**四川师范大学本科毕业论文（设计）**

|  |
| --- |
| 分布式存储系统中基于EVENODD码的容错技术研究与实现 |

|  |  |
| --- | --- |
| **学生姓名** | **江小玉** |
| **院系名称** | **计算机科学学院** |
| **专业名称** | **计算机科学与技术** |
| **班 级** | **2013 级 1 班** |
| **学 号** | **2013110116** |
| **指导教师** | **韩鸿宇** |
| **完成时间** | **2017年 5 月 05 日** |

分布式存储系统中基于EVENODD码的容错技术

研究与实现

学生：江小玉 指导教师：韩鸿宇

内容摘要：2014年，大数据作为一项新兴的技术初次进入政府工作陈述。大数据具有海量化、多样化等的特征，而这样的特征是双刃剑，既带来了新的发展机遇，也带来新的技术挑战。大数据早已经步入了人们的生活，例如“城市计算”就是运用计算机科学技术，以发展中的城市为规划蓝图，与城市前景规划、交通运输、能源动力、环境保护、社会经济学等学科结合的新兴领域。城市计算是利用城市感知技术不停的获取、分析和整合城市中不同领域的各种数据来解决城市所面临的挑战（例如环境恶化、交通拥挤、能耗剧增、规划布局落后等）的过程[[[1]](#endnote-0)][[[2]](#endnote-1)]。数据驱动的新型业务形式日趋常见，大数据时代的技术环境等各方面都有新的变动，传统的数据可靠性实施也需要与时俱进。传统存储技术已经难以适应大数据时代。

传统存储技术依靠本地集中式存储，硬盘一旦发生损坏或者出现操作系统故障，数据都将陷入丢失或者失效的困境。当庞杂的数据进行存储时，传统存储技术无法确保数据的高安全性及高可靠性。为了提升性能，早期研究人员采用备份技术作为冗余手段，当一个节点的数据流失时，可以从其他的备份节点复制数据恢复原数据，以此重构故障盘符中丢失的数据。但是节点可靠性较低，复制冗余策略造成了巨大的存储浪费，在庞大的数据量面前显得十分低效。

RAID的出现，将多个磁盘组织成为一个逻辑盘，一度有效的提高了磁盘的

性能的存储容量。但是随着时代的发展，RAID也不能完全满足海量数据存储的需求。

为了降低冗余度，减少成本，提高存储效率，研究人员们在RAID的基础上又采用纠删码对文件进行编码存储。编码的方式非常多，例如EVENODD[[[3]](#endnote-2)]就是RAID-6常用的一种编码策略。适用于恢复任意俩个磁盘信息失效的情况。Hadoop内置的HDFS文件系统的NameNode/DataNode 可以提供数据切片并冗余存储, 平衡数据的功能。Hadoop的分布式存储系统，可在低价的服务端上搭建大规模的结构化存储集群。Facebook在Hadoop上已经实现了纠删码模块HDFS-RAID，本论文在此研究结果基础上设计实现HDFS-EVENODD模块。

关键词：大数据 城市计算 冗余手段 纠删码 HADOOP HDFS RAID EVENODD

**Research and implementation of fault tolerance technology based on EVENODD code in distributed storage system**

**Abstract:** In2014, the government work report for the first time entered the big data, as a new technology. Big data has the characteristics of mass, diversification and so on, but everything has two sides , not only brings new opportunities for development, but also brings new technical challenges. Big data have entered people's life, for example, "city computing" is the use of the computer science and technology, with the development of city planning background, the emerging field of integration and city planning, transportation, energy, environment, sociology and economics. The city is calculated in different areas through the city data sensing technology to constantly acquire, integrate and analyze the city to address the challenges faced by the city (such as the deterioration of the environment, traffic congestion, increase energy consumption and backward process planning). The new business model driven by data is becoming more and more common, the technical environment and other aspects of the big data era have new changes. Traditional storage technology is difficult to adapt to the era of big data.

Traditional storage technologies rely on local centralized storage, once the disk is damaged or operating system failure, the data will face the loss or failure of the dilemma. When the complex data is stored, the traditional storage technology can not guarantee the security and reliability of the data. In order to improve the storage of high reliability and high availability, early researchers used redundancy backup technology as a means, when a node of data loss, can copy the data from the backup node other to restore the original data, missing data in order to reconstruct the fault in the letter. However, the reliability of the node is low, and the replication strategy increases the storage cost, which is very inefficient in the face of huge amount of data.

The emergence of RAID, will be a number of disk organization into a logical disk, once effectively improved the disk performance storage capacity. However, with the development of the times, RAID can not fully meet the needs of massive data storage.

In order to reduce the redundancy, reduce the cost and improve the storage efficiency, the researchers use the erasure code to store the files on the basis of RAID. The way of coding is very much, such as EVENODD is a commonly used RAID-6 coding strategy. Applicable to restore any two disk information failure. Hadoop built-in HDFS file system NameNode/DataNode can provide data slicing and redundant storage, balance the function of data. HDFS, can be built on a cheap server on a large scale structured storage cluster. Facebook has been implemented in Hadoop erasure code module HDFS-RAID, this paper based on the results of the design and implementation of HDFS-EVENODD module.

**Key words**: Big-Data; City-Computing; Redundancy; Erasure-code; Hadoop; HDFS; RAID; EVENODD

目录

[1 绪论 5](#_Toc15612)

[1.1研究目的和意义 5](#_Toc31341)

[1.2国内外研究现状 6](#_Toc1631)

[1.3主要贡献 6](#_Toc21735)

[1.4文章的结构 7](#_Toc5285)

[2 Hadoop基本架构 8](#_Toc3096)

[2.1 Hadoop 生态系统框架 8](#_Toc6057)

[2.2 Hadoop核心组件HDFS简介 9](#_Toc7529)

[2.3 Hadoop相比RAID 11](#_Toc22593)

[2.4 Hadoop集群部署方法 12](#_Toc13088)

[3 EVENODD算法详解 13](#_Toc28990)

[3.1 有限域的介绍 14](#_Toc14310)

[3.2 EVENODD算法 15](#_Toc22867)

[3.2.1 编码算法 15](#_Toc27531)

[3.2.2 任意两个磁盘失效修复算法 16](#_Toc3416)

[3.2.3 单个磁盘出错修复过程 20](#_Toc10946)

[4 EVENODD基于Hadoop平台的实现 21](#_Toc25989)

[4.1 HDFS-RAID实现 21](#_Toc27116)

[4.2 HDFS-EVENODD设计实现 21](#_Toc13715)

[4.2.1 在WINDOWS系统下的实现 21](#_Toc9017)

[4.2.2 在HADOOP系统下的实现 25](#_Toc5868)

[5 总 结 26](#_Toc14995)

[5.1 性能分析 26](#_Toc30030)

[5.2 总结 26](#_Toc11834)

[6 致谢 28](#_Toc24833)

[参考文献 29](#_Toc2900)

分布式存储系统中基于EVENODD码的容错技术

研究与实现

# **1 绪论**

## 1.1研究目的和意义

数据的最大价值不在量有多么大，而在于对数据的完整分析和充分的应用，但是如何存储，如何提取数据是一个急需解决的问题。

随着科技发展，各种传感器技术以及云计算、云存储单位日趋成熟，我们有了社交媒体、交通流量、气象、地理等多种数据。数据存储的方式不仅限于文字、图片、视频...数据的多元性和多源性绕不开对数据的存储、管理、提取和分析。当前，大数据已经从互联网领域延伸至通信、金融服务、房产、贸易活动等行业，基于大数据领域产生的新兴技术、新媒体产品、新服务形式、新兴业态不断涌现，并且不断的融入到社会公众生活。各行各业通过新媒体形式迅速产生庞大复杂的数据。汹涌而来的大数据时代，数据在传统技术下难以存储和传输。在基于大数据的网络用户行为数据分析中可以发现，百度、Goolge、阿里巴巴、Facebook等大型网站的总数据量早已超过数千PB，未来将以ZB为单位计算[[[4]](#endnote-3)]。而据2011年的IDC研究报告预测，全球的数据总量在未来十年里将增长近50倍[[[5]](#endnote-4)]。

数据存储最简单的一种冗余机制是复制策略，以存储原始数据的N个副本来保证N-1个擦除的容错性，牺牲硬盘的存储空间而保证系统的性能。巨大的存储容量必然要求更多高性能的服务器，这无疑将增加昂贵的开发成本，所以在保证可用性下如何优化成本是迫切需要解决的问题。

传统的技术和方式无法满足海量数据增长的挖掘应用、分析处理以及存储需求。Hadoop作为研究大数据的一项重要技术，拥有存储和处理千兆字节以上数据的扩容能力，可以将普通的计算机组成数千个节点的大型服务器集群来分发数据，并在数据存在的节点上同时处理数据，保障了低成本、高效率和可靠性的优势。Hadoop集群的优势由其分布式存储文件系统— HDFS的分布式特点而来，通过网络在多台主机上实现共享，即便系统中的某些数据节点不小心被损坏，利用数据副本恢复源数据维持系统整体运作而不会有数据损失。HDFS通过复制保存副本保障系统的容错能力，底层数据存储默认使用三副本冗余以保障数据的可靠性，存储浪费与数据总量成正比，造成巨大的存储浪费，降低系统的性能。 HDFS也以牺牲存储空间来提高系统可靠性，因为它的设计原理主要是基于复制冗余和镜像冗余，而针对HDFS数据多副本容灾设计而带来的存储空间利用率低，恢复效率低等问题，国内很多学者结合纠删码的思想，引入编码译码模块，保证了集群数据的安全性，提高数据恢复的速度，减少了空间代价，降低了整体的存储成本。

为了降低冗余度，提高效率，许多学者开始使用纠删码技术编码生成校验文件块，使之成为HDFS系统中的一种冗余机制。纠删码技术的基本思想就是：1、将需要存储的文件根据系统节点的个数设置分成文件块，2、使用合适的编码算法对文件块进行编码得到校验文件块，3、将这些文件碎片部署到系统节点和校验节点上。当系统节点失效或者文件块丢失时，只需要从系统中下载相应的帮助文件块，通过译码技术就可以恢复数据，得到原始文件。使用纠删码算法对文件进行编码存储，可以减少存储副本数目，提供合理的的额外存储，避免了过度的存储开销，如果部分磁盘损坏或丢失，可以通过译码恢复源文件，给系统的性能带来了数量级的提高。本论文研究的便是分布式存储系统中基于EVENODD码的纠删码容错技术。

### 1.2国内外研究现状

1988年，RAID（Redundant Array of Inexpersive Disks），即“廉价磁盘冗余矩阵”概念的提出[[[6]](#endnote-5)]，有效的解决了磁盘存取速度与CPU处理速度不匹配对计算机整体性能的掣肘。 RAID的设计逻辑是通过对多个独立的磁盘并行操作，提高输入输出速率，提供更大的存储量。由于RAID采用分布式存储，将数据存储在不同的磁盘，节点可靠性较低，在成百上千个节点同时工作时，必须有良好的容错性。

2012年，Goplan等人提出的局部修复性(repair locality)[[[7]](#endnote-6)]，表示如果一个编码满足任意位局部性，那么这个编码可以非常有效的降低修改过程中协助恢复节点的访问数目，以此降低数据修复的复杂性。

2014年，付园[[[8]](#endnote-7)]针对HDFS系统底层设定的复制三副本冗余机制的弊端，设计实现了将复制策略和RS纠删码两种冗余方法结合优化的数据冗余算法，基于分布式文件系统，修改了Hadoop的底层代码，应用了这种优化后的算法。最后在自行部署的分布式系统环境中完成了对系统性能和算法功能的验证。

## 1.3主要贡献

（1）根据国内外研究发展现状分析大数据的现实应用，以及在大数据时代传统存储技术的弊端和纠删码在冗余存贮技术中的优势。

（2）详细介绍了Hadoop平台的组成和结构，分布式文件系统的工作原理，和RAID技术及RAID的发展，以及搭建Hadoop伪分布式环境的配置详细步骤。

（3）对EVENODD的算法进行了详细的讲解，并举例演绎了编码和译码的过程。对EVENODD编码技术中涉及的有限域，MDS阵列等概念都做了介绍。并在Windows环境下实现单个矩阵的编码译码以及实现丢失图片的恢复。

（4）在Facebook的HDFS-RAID模块上设计实现HFDS-EVENODD模块。

## 1.4文章的结构

主要内容：

（1）学习Hadoop体系结构以及周边框架；搭建Linux系统下Ubantu环境中的Hadoop伪分布式集群环境，运行实例测试Hadoop环境是否搭建成功，包括从本地上传、下载、修改、删除HDFS文件系统中的文件等；重点学习HDFS模块的工作原理。

（2）阅读Facebook版本的Hadoop源码，源代码实现过程复杂，结构十分庞大，需要继承实现的类比较多，需理解项目的编译方式、部署方式和配置文件的结构，着重学习RAID模块的运行流程以及相关类的实现。

（3）编码基于有限域和素数域，补充线性代数，有限域的相关知识。

（4）学习纠删码的基础理论知识，包括MDS阵列的概念及构造，RAID技术的起源、成熟和发展等。理解纠删码编码译码的思想，例如X码，B码[[[9]](#endnote-8)]，S码[[[10]](#endnote-9)]等这样与EVENODD编码一样可以容许两个存储设备同时故障的数据分布策略的编码。

（5）学习EVENODD编码及译码的算法原理，并使用java语言实现。首先在windows下实现基本的单个矩阵编码，丢失列后恢复的功能，再实现文件分块，编码，恢复原图的功能，最后嵌入HadoopUSC源码中，实现基于HDFS文件系统下的恢复文件的功能。

（6）结合搭建的Hadoop伪分布式环境测试EVENODD代码模块的功能，是否能恢复删除的文件块。

（7）对EVENODD编码的性能经行分析，与其他纠删码的性能进行比较。

安排：本论文一共分为5章

第1章 绪论。主要分析论文研究的背景和意义，国内外发展的情况，以及本论文的行文思路和内容安排。

第2章 Hadoop平台介绍。主要介绍了研究大数据的方法，着重研究Hadoop平台，讲解Hadoop基础框架结构，包含的组件；介绍了RAID的工作原理，以及部署伪分布式集群环境的流程。

第3章 EVENODD算法详解。主要介绍了EVENODD的算法逻辑，举例演绎此算法编码译码的过程。

第4章 EVENODD基于Hadoop平台的实现。在基于HDFS-RAID模块上，研究代码的层次，设计实现EVENODD；在windoWS下先实现了对单个矩阵的编码、擦除和恢复后，再实现了一个图片分块、编码以及擦除任意俩块和恢复后，将代码移植到了Hadoop平台上，实现HDFS-EVENODD模块。

第5章 总结。分析EVENODD编码的性能并进行总结。

# 2 **Hadoop基本架构**

在商业应用中，常使用Hadoop框架研究大数据。Hadoop是一个分布式的软件框架，它也是一个基于JAVA语言的框架，能够进行海量数据的分布式存储、数据处理和数据分析。与之相比较的是Apache Spark大数据框架，但Spark是一个专门用来对分布式存储的大数据进行处理的工具，不能进行分布式数据的存储，需要集成其他的文件系统。所以对于企业来说，Hadoop不需要购买和维护昂贵的服务器硬件，成本更低。

## 2.1 Hadoop 生态系统框架

Hadoop的核心设计为分布式文件系统HDFS和MapReduce计算框架。Hadoop的生态系统架构如图2.1。

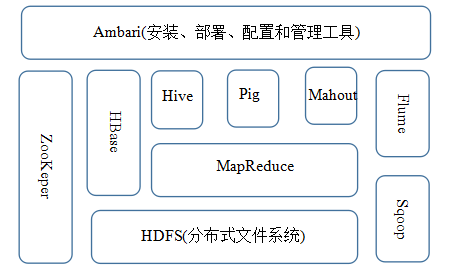


图 2.1 Hadoop的生态系统架构

下面介绍Hadoop生态框架中的主要组件：

Pig：基于Hadoop的数据流系统。Pig是基于MapReduce的一种数据分析工具，可以加载数据，转换数据格式在Hadoop上运行，以及存储最终结果等一系列过程。

Hive：基于Hadoop的数据仓库。将数据化的数据文件映射为一张数据库表，可向HDFS添加数据，并允许使用类似SQL的语言进行数据查询，可以将SQL语句转换为MapReduce任务运行。

Zookeeper：分布式协作服务。解决分布式环境下的数据管理问题，统一命名，状态同步，集群管理为HBase提供稳定服务和failover机制。

Avro：数据系列化系统，用于支持大批量数据交互的应用。

MapReduce：分布式计算框架，超大型数据集的并行计算框架，负责计算。

Hbase：分布式列存数据库，面向列的分布式存储系统，用于在Hadoop中支持大型稀疏表的列存储数据环境，数据库集群，位于结构化存储层，是一个开源的非关系型分布式数据库（NoSQL）,基于谷歌的BigTable建模，是一个高可靠性、高性能、高伸缩的分布式存储系统。

HDFS：分布式文件系统，负责存储数据。

## 2.2 Hadoop核心组件HDFS简介

HDFS可以管理多台机器上的文件，位于系统底层，存储管理Hadoop集群所有节点上的数据，是Hadoop体系中分布式数据存储和管理的基础部件。HDFS能检测和应对系统中发生的硬件故障，在廉价的硬件上运行,通过网络管理分布在不同主机存储的文件系统，让不同主机上的用户共享文件和存储空间，程序和底层用户就感觉像访问本地的磁盘一样，系统在遇到故障时能继续运行，用户不会明显察觉到中断。HDFS以流式数据模式访问和存储超大文件，这样的模式能提高系统吞吐量，并且以文件块（Block）的形式操纵文件。HDFS的安全性和可靠性也成为目前研究的热点难点。HDFS的整体结构如图2.2。

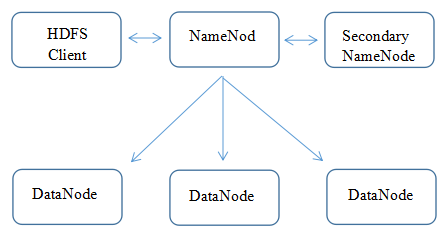


图 2.2 HDFS结构图

HDFS适合大文件存储，不适合小文件。NameNode将Block分布到不同的

DataNode节点上，小文件的 Block 越多，对 NameNode 的内存压力越大。

HDFS为主从式结构，系统中主节点，即名字节点(NameNode)只有一个，从节点，即数据节点(DataNode)可以有多个。 当程序运行时，可以在文件目录下找到in\_use.lock文件，in\_use.lock 文件中没有什么实际内容，但是这个文件表示 name 目录已经被 NameNode 进程占用了，如果在此时再启动 NameNode，进程就会报错，无法进入。这意味着如果多个进程同时编辑数据会出现问题，in\_use.lock文件的存在，表示锁定NameNode节点，其他 NameNode进程无法进入,这也表示HDFS不支持并发写情况，所以只能允许一个 NameNode存在。

下面介绍一些基础的概念：

Block：数据块是最基本的存储单位，是linux文件系统划分的一个概念。对于文件内容而言，一个文件的长度大小是Size，那么从文件的 0 偏移量开始，按照固定的大小，顺序对于文件进行划块分并编号，每一个块称为一个 Block。 HDFS默认 Block 大小是 64MB，以一个大小为256M的文件为例，共有256/64=4个Block。与普通文件系统不同的是，HDFS中一个比单个Block块小的文件不会占据整个Block的大小。HDFS可以根据实际情况将Block的大小设置为128M，256M，或者更大的单位。具体设置方法：复制

<br><name>dfs.block.size</name></br>

<br><value>67108864</value></br>

这段代码到 hdfs-site.xml 文件中并修改大小。

HDFS-Client：客户端可以切分文件，并且访问HDFS。客户端与名字节点进行交互，获取文件在数据节点上的位置信息；客户端还与数据节点进行交互，读取数据。文件具体的读写流程：从HDFS中读取数据时，首先会连接到客户端，客户端通过访问名字节点获取到文件包含的所有的文件块以及这些块所在的数据节点，然后客户端从各个数据节点上下载数据块。向HDFS写入数据时，首先也要连接到客户端，客户端通过访问名字节点获得文件块被分配到的数据节点，最后由客户端使用流式数据接口向数据节点中写入数据，并从该数据节点向被选为备份节点的数据节点中复制数据作为冗余数据，降低数据损失风险。

NameNode：系统的名字节点，管理系统的命名空间。接收Client的操作请求；维护系统的目录结构和文件；管理文件与数据块之间的关系。为了提升检索速度，文件目录树一般放在内存中，为了持久保存，则写入硬盘中保存。故存储在硬盘上，但运行时在内存中。

DataNode：系统的工作节点，存储实际的数据，文件被分成多个数据块存储在硬盘上，反馈存储信息给NameNode，默认存储多个副本以保障安全性，HDFS可以存储海量数据，实际上指的就是 DataNode 节点的快速扩展，副本越多会占用磁盘空间。副本默认是三个，三个副本一般存在三个机器上，一般就近放在最近机架上的电脑上(确定)，同一机架上的另一台电脑(随机)，另一机架的另一台电脑(随机))。目录没有副本，副本越多会占用磁盘空间，可以根据实际情况在配置文件中设定。具体配置方法：进入hadoop/conf 目录下，找到hdfs-site.xml配置文件，可以看到

<br><name>dfs.replication</name> </br>

<value>1</value></b></br>

在这段代码里，则可以设置副本数。

SecondaryNameNode：次名字节点，辅助NameNode节点，分担名字节点的工作量。执行过程：首先从名字节点上下载fsimage和edits元数据信息，然后将两个信息进行合并，生成新的fsimage文件并保存在本地，再将其推送到名字节点，同时重置名字节点中的edits信息。名字节点是集中的单一故障点，次名字节点为名字节点创建检查点，在紧急情况下，可利用次名字节点恢复名字节点，降低名字节点数据丢失的风险。

## 2.3 Hadoop相比RAID

RAID（Redundant Array of Independent Disks），即“独立磁盘冗余阵列”。Katz、Gibson、和Patterson这三位工程师在1987年发表了题为《A Case of Redundant Array of Inexpensive Disks》的论文，文中提出了RAID的概念。RAID的设计原理是将多个独立的容量小而廉价的物理硬盘按照不同的方式有机组合成一个硬盘组（逻辑盘），让这个硬盘的性能可以超过单个大而昂贵的硬盘，组合的硬盘会拥有更高的存储性能，更大的存储容量和提供数据备份技术，能通过资源冗余来提高服务质量。而从用户看来，组合的磁盘组就是一个完整的磁盘，对组合磁盘的操作就像操作单个磁盘。RAID通过数据分割、多通道并行读/写来提高I/O速率，磁盘阵列的存储速度就比单个磁盘的要高很多。RAID还通过保存冗余的数据、校验信息以及简单的容错编码来提高存储的可靠性，在用户数据一旦发生损坏后，利用这些冗余信息可以使损坏的数据得以恢复。因此，RAID技术在提高系统I/O速度、数据传输率、数据可靠性和数据存储容量方面取得了很好的效果。早期的RAID系统使用最简单的奇偶校验技术来保障数据的可靠性，至多允许一张磁盘失效。但是当系统的规模较大时，同时发生多张磁盘失效的概率比较高，存储系统的可靠性迅速降低。

RAID经过多年的突破和发展，现在可分为许多等级。RAID技术主要包含了从RAID0到RAID7等多个规范等级，并且各个等级的技术侧重点不相同，比较常见的几种规范如下:

RAID 0: 传输速率高，但是无法保障数据的安全性。  
 RAID 1：强调数据的安全性，但缺点是比较浪费空间。

RAID 10：RAID 0+1，即RAID0和RAID1标准结合的产物。它同时拥有RAID0的读写速度快和RAID1的数据安全性等优点，但同样的继承了RAID 1的缺点，CPU占用率高，并且磁盘的利用率比较低。（值得注意的是：RAID 0+1 最少需要4块硬盘，且做RAID 1+0时要先作RAID 1，再把数个RAID 1做成RAID 0，这样比先做RAID 0，再做RAID 1有更高的可靠性。）

RAID 2：技术实施比较复杂，商业环境中很少使用RAID 2技术。

RAID 3：可以很好的传输海量连续的数据，但是奇偶盘会成为随机数据写操作的掣肘。

RAID 4：由于技术原因，RAID 4技术也很少在商业环境中使用。  
 RAID 5：也相当于RAID 0和RAID 1标准的综合，更适应于读写小的[数据块](http://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E5%9D%97" \t "http://baike.baidu.com/_blank)和随机数据。

RAID 6：数据的可靠性比RAID 5更高，但是所浪费的空间也更大，“写性能”比较差。因为性能不佳并且实施方式比较复杂，RAID 6的实际应用也不多。

RAID 7：可以不使用主机的CPU资源，完全独立运行。RAID 7是一种新型的RAID标准，与其他的几种RAID标准有很大区别，可以看作是一种存储型计算机。

那么在[存储系统](http://baike.baidu.com/item/%E5%AD%98%E5%82%A8%E7%B3%BB%E7%BB%9F" \t "http://baike.baidu.com/_blank)中使用RAID技术的优势可以体现在：第一是通过把数个小的、廉价的磁盘有机组织成一个逻辑磁盘卷提供磁盘的跨越功能；第二通过把需要读写的数据分成多个小的[数据块](http://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E5%9D%97" \t "http://baike.baidu.com/_blank)（Block），同时访问多个磁盘以提高磁盘的读写速度；第三是通过[镜像](http://baike.baidu.com/item/%E9%95%9C%E5%83%8F" \t "http://baike.baidu.com/_blank)技术或者校验冗余提供磁盘的容错能力。RAID技术和Hadoop的设计原理与思路一样，但是两者不在一个量级上。RAID只能在一台机器上，Hadoop可以部署在成千上万台机器上。简单点来说，即RAID的并发读写速度提高最多只能是单张磁盘的十几倍，但是Hadoop可以是数万倍。RAID的可靠性最大程度能做到单张磁盘损坏时数据不丢失，但是Hadoop可以做到很多机器损坏而数据不丢失。

## 2.4 Hadoop集群部署方法

这里使用FackBook的包含RAID模块的源码，在主机上部署为伪分布式环境。在此系统中能进行EVENODD编码的仿真实现。

准备工具：VMware Workstation、Ubantu系统、HadoopUSC源代码

步骤：

（1）、按照Linux系统的虚拟机Ubantu，网上有许多详细教程，难度不大，这里就不进行详细讲解。

（2）、在Ubantu系统中安装Hadoop系统，进入HadoopUSC项目的根目录下，输入命令“ant-Dversion 0.20”即可。RAID模块需要单独编译，具体的编译和配置方法可见教程[[[11]](#endnote-10)]。

（3）、配置完之后可以启动Hadoop系统集群，输入“bin/hadoop namenode -format”命令初始化HDFS文件系统，然后输入“bin/start-all.sh”命令启动所有的守护进程，输入“bin/start-raidnode.sh”命令把raid节点启动起来。启动成功后，输入JPS查看系统运行的进程，出现JobTracker、SecondaryNameNode、DataNode、TaskTracker、NameNode和RaidNode六个进程，则表示包含RAID模块的Hadoop系统部署成功了。

## 3 EVENODD算法详解

EVENODD是基于RAID阵列的一种纠删码算法，由简单的异或操作产生两个冗余磁盘，可以容忍两个磁盘故障。这种冗余存储是最佳的，非常高效。EVENODD编码的小规模写操作也非常的高效，当一个磁盘扇区被修改时，只需要同时修改俩个额外的磁盘扇区。

在我们的假设中的一个明显的约束条件是，信息磁盘的数量M是一个素数。如果所需的磁盘数量不是素数，可以简单地假设有更多的磁盘，而不影响的编码和解码程序。假设有M + 2个磁盘，原始数据存储在第一个M磁盘中，而冗余数据存储在最后两个磁盘中。M，信息磁盘的数量，这里假设是一个素数，这个要求是很重要的，因为没有这个假设，该算法将失效。但是在实际情况中，M的素性不是很好保证，如果我们想存储任意数量的磁盘，这个数不一定是素数，到时可以采取这个任意数的下一个素数，并假设有没有信息的磁盘（即所有的信息位是0）。

EVENODD和针对两个磁盘故障恢复提出的其他方法想比较，有明显的优势：

（1）EVENODD编码基于简单的水平异或运算和对角线异或运算产生校验数据，不涉及递归运算。

（2）EVENODD编码只需要奇偶校验的硬件，通常是在标准的RAID-5控制器中就可实现，因此对其可实施的STA标准RAID-5没有任何硬件的变化。而基于RS代码的方案需要特殊的硬件，以支持有限域类型的计算，因此RS不能被纳入标准的RAID-5控制器。

（3）它可以被纳入到已知的RAID技术。例如，奇偶校验技术可以部署在任何磁盘上，避免瓶颈效应，当涉及重复写操作的时候（RAID-5）。

（4）操作符号可以有任意大小，从比特到多个扇区，没有限制的位或字节。

（5）在涉及多个磁盘的系统中，经常会遇到需要许多小规模写操作的情况。一个小规模写操作更新一个单一的磁盘扇区（一个符号）。EVENODD编码提供了极大的灵活性，因为所涉及的符号可以具有任意大小。通常情况下，我们将一个符号作为一个磁盘扇区。而且大多数小规模写操作只会影响两个冗余符号，也就是说，每一个写我们需要三读和三写操作。只有当受影响的符号是在对角线（M-2，1）、（M-3，2）...（0，M-1）时，我们要修改列M + 1和列M上的一个符号。在任何情况下，所有的校验符号都是独立的。

（6）传统的已知方案中，采用的最佳冗余存储（两个额外的磁盘）是基于Reed Solomon（RS）纠错码，需要在有限域上计算并导致一个更复杂的实现。例如，我们发现在15个磁盘的磁盘阵列实现，EVENODD编码的复杂性是使用RS编码的50%。

## 3.1 有限域的介绍

有限域，即只含有限个元素的域，由数学家E.[伽罗瓦](http://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=183204&ss_c=ssc.citiao.link" \t "http://baike.sogou.com/_blank)最先发现，因此以他的名字命名，又称为伽罗瓦域，记为符号GF。GF和[有理数](http://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=268048&ss_c=ssc.citiao.link" \t "http://baike.sogou.com/_blank)域、实数域等其他域有着许多不相同的性质，有限域的代表性应用是编码理论、开关理论、纠错码和AES。有限域中元素的个数称为F的阶，记为，例如GF(2)。最简单的有限域是整数环Z模一个素数p得到的余环Z/(p),由p个元素0,1,…,p-1组成,按模p相加和相乘。

纠删码中的计算是基于某个有限域讨论的，比如素数域、二进制域，这俩种有限域上都包括了模加、模减、模乘、模平方、模逆和模的约减这几种运算。素域和二进制域被人们广泛采用，例如AES是1997年美国政府世界公开征集的密码算法，2000年称为美国国家标准。它利用扩成有限域，8次不可约多项式。一个字节就可视为一个多项式，成为中的一个元素。

EVENODD算法基于素数域，M为一个素数。在算法中，需要做有限域上的模运算，与实数域上的模运算有不一样的计算方式。一般情况下，模运算经常与取余运算混为一谈，因为在许多编程语言里，都用符号“%”表示取模运算或者求余运算。但是值得注意的是运算中有负数的情况下，计算结果是不一样的。，a和b都为整数，取模和求余运算的计算方法相同：1、求整数商：c = a / b；2、计算模或者余数：r = a - c \* b。但是求模和求余运算在计算的第一步有些不同: 取余运算在计算取整数c的值时，向0 的方向舍入(fix()函数)，而取模运算在计算整数c的值时，向负无穷的方向舍入(floor()函数)。

例如：计算 -7 mod 4，那么：a = -7；b = 4，计算过程如下：

先求整数商c，求模运算时c = -2（向负无穷方向舍入），求余运算时c = -1（向0方向舍入）；再因c的值不同，虽然计算模和余数的公式相同，但求模运算时r = 1，求余运算时r = -3。

总结可得规律：当整数a和整数b符号相同时，求模运算和求余运算得到的c值相同，因此结果相同，例如11%3 = 2，-11%（-3） = -2。但是当a和b符号不相同时，结果不同，并且求模运算结果的符号和b相同，求余运算结果的符号和a相同。

EVENODD编码算法中需要做有限域上的模运算，符号，则

，并且，例如，。可得运算公式：a % b = ( a + b ) % b。

## 3.2 EVENODD算法

## 3.2.1 编码算法

为了简化演绎算法，这里假设每个磁盘只有M - 1个符号的信息。程序的工作原理与任意容量的磁盘，分别处理每一个块上的M - 1个符号。为了简单明了，在以下的一些例子中，假设每个符号的大小都是一个位。但是在一些实际应用中，一个符号可能像一个512字节的磁盘扇区一样大。

这里假设共有M+2张磁盘，其中M张磁盘存储原始数据信息，最后的两张磁盘存储奇偶校验信息。在(M-1)×(M+2)磁盘阵列中，表示第j张盘上的第i个码元，其中，。在下面的译码过程中，会给矩阵加上0行，即在矩阵的最后一行加上全为0的一行，表示为，这个假设不是必要的，但它是有用的标记，便于计算。这时阵列表示为M×（M+2）

水平奇偶校验相关的符号集如下所示:

表 3.1 水平校验相关符号

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ◊ | ◊ | ◊ | ◊ | ◊ | ◊ |  |
| ♣ | ♣ | ♣ | ♣ | ♣ | ♣ |  |
| □ | □ | □ | □ | □ | □ |  |
| ♥ | ♥ | ♥ | ♥ | ♥ | ♥ |  |

对角线奇偶校验相关的符号的集合如下：（请注意，∞是特殊的对角线，对角线奇偶校验产生S符号，S为偶数表示偶校验，S为奇数表示奇校验）。

表 3.2 对角线校验相关符号

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ◊ | ♣ | □ | ♥ | ∞ |  | ◊ |
| ♣ | □ | ♥ | ∞ | ◊ |  | ♣ |
| □ | ♥ | ∞ | ◊ | ♣ |  | □ |
| ♥ | ∞ | ◊ | ♣ | □ |  | ♥ |

在(M-1)×M 磁盘阵列中，通过行校验和对角线校验产生两列冗余数据充当校验集来保证任意两个磁盘同时故障时（位置已知）数据不会丢失，或者纠错一列出错的情况。

编码过程：

 (3-1)

当0 ≦ l ≦ m-2时

 (3-2)

 (3-3)

例：当m=5时，按照公式，如图方式进行编码，D0~D4是数据盘，D5和D6为冗余数据，充当校验盘。

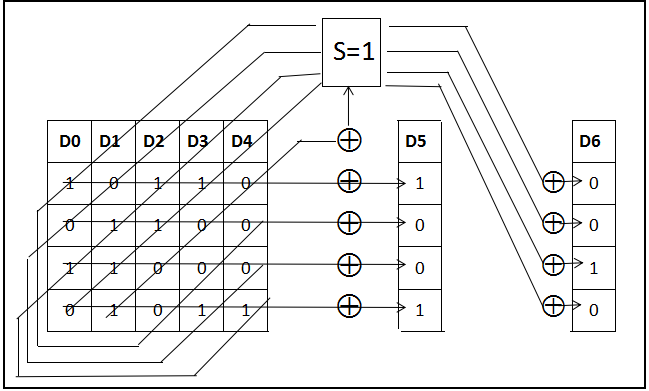


图 3.1 编码过程图

## 3.2.2 任意两个磁盘失效修复算法

修复失效磁盘的方法即译码算法，任意两个磁盘失效，需要知道具体错误位置。假设存在故障的是Di和Dj列，则 0 ≦ i < j ≦ m+1，这里有4种情况，可以分别进行讨论。

1. i = m , j = m + 1

两个冗余磁盘已经失效，重建就相当于编码的情况。使用上述（3-1）、（3-2）、（3-3）公式即可恢复。

2. i < m , j = m

即一个冗余磁盘和一个数据磁盘已经失效，此处假设D1和D5出现故障。

译码过程：

1、先为所有磁盘增加一个0的数据，即令 ，其中0 ≦ l ≦ m+1，再通过公式（3-4）计算奇偶符号S：

 （3-4）

可得：



2、通过公式（3-5）计算D1

 （3-5）

如图可得D1：

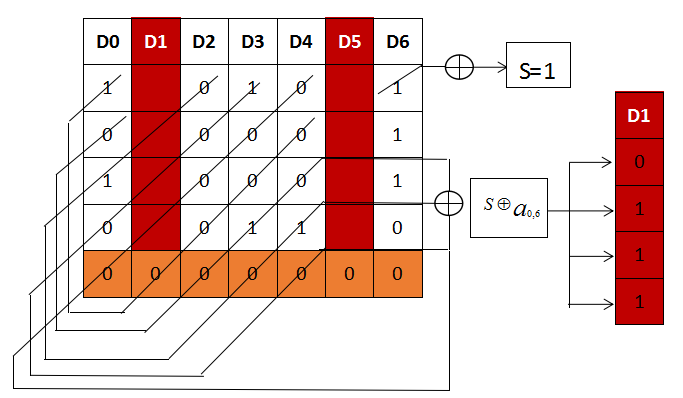


图 3.2 译码D1列过程图

3、最后可用公式（3-2）得到D5：

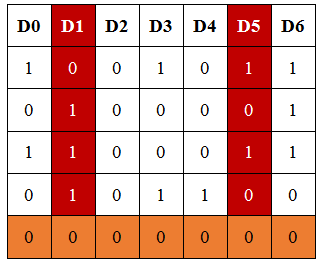


图 3.3 译码D5列过程图

3.i < m , j = m+1

一个冗余磁盘和一个数据磁盘已经失效，此处假设D1和D6出现故障。

译码过程：先通过水平奇偶公式（3-2）计算D1，通过公式（3-1）和（3-3）计算出 D6。

4. i < m , j < m

两个数据磁盘已经失效，这是主要的情况，这里假设 D0 和 D2 出现了故障。

译码过程：

1、先为所有磁盘增加一个0的数据，即 ，其中0 ≦ l ≦ m+1，我们首先需要一个入口，在那里我们可以开始。例如，对角线（3，1），（2，2），（1，3），（0，4）相交在条目的列2（2，2）只：这是特殊的对角线，它有奇数奇偶性。由于在这个对角线上的唯一的位丢失位（2，2），通过检索它使用其他位再通过公式（3-6）计算奇偶符号S：

 （3-6）

如图可得S=1：

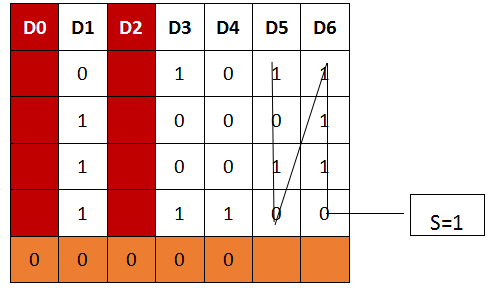


图 3.4 求符号S图

2、再通过公式（3-7）计算：

 （3-7）

可得：



又可得：





最终可得：



3、再通过公式（3-8）计算

 （3-8）

可得：



又可得：





最终可得：



4、再通过以下步骤计算：

（1）令S ← <-(j-i)-1>m，

（2）令





（3）令S ← <S-(j-i)-1>m，若S = m-1停止，否则返回（2），

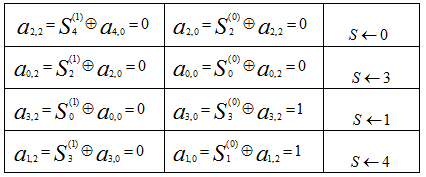


图 3.5 计算步骤图

通过如图3.5的计算步骤，可得译码结果如图3.6所示：

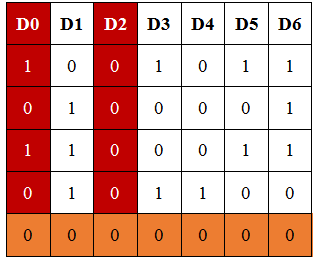


图 3.6 译码结果图

## 3.2.3 单个磁盘出错修复过程

当前的算法可以纠算出一个磁盘出现错误的情况，并且进行修改。解码仍然分为四种情况。

首先通过公式（3-9）和（3-10）计算水平综合征和对角线综合征：

 （3-9）

易知：



同理：

 （3-10）

易知：



这里也分4种情况讨论：

1、

这种情况说明磁盘阵列没有数据错误。

2、

这种情况说明磁盘阵列的 m 列出现错误，只需要用公式（3-2）就可以恢复m列的数据。

3、

这种情况说明磁盘阵列的 m+1 列出现错误，只需要用公式（3-1）和（3-3）就可以恢复 m+1 列的数据。

4、

这种情况下说明一个数据盘 j (0 ≦ j ≦ m-1) 出现错误，需要计算出确定的位置。

（1）首先为各个盘虚拟添加一行数据0，然后通过公式（3-9），（3-10）得出：





（2）然后对向量里的数据右移直到 ，（其中 j 是右移的次数），经计算， j=2，即D2出现错误，此时只需要

的前 m-1 个数据与 D2 的数据逐一相加即可恢复。



# 4 EVENODD基于Hadoop平台的实现

## 4.1 HDFS-RAID实现

目前，HDFS-RAID模块采用了XOR和RS两种纠删编码方式。XOR编码比较简单，只采取异或算法产生一个校验块，纠错能力也相对较弱，只能纠错一个块。RS编码实现复杂，但是纠错能力强。

源代码里负责对文件进行编码和对文件失效数据块进行修复的类：Encoder表示编码类，实现了对文件块编码的框架；Decoder表示译码类，实现了对文件失效数据块进行恢复的框架；ErasureCode表示纠删码类，具体纠删码的底层实现模块。论文设计实现的EVENODD算法需继承这些类实现恢复功能。

HDFS-RAID模块主要由三个部分组成，一个是封装了DFS的DRFS，一个是RaidNode进程，另外一个RaidShell命令行工具。HDFS-RAID工作流程如图4.1所示：

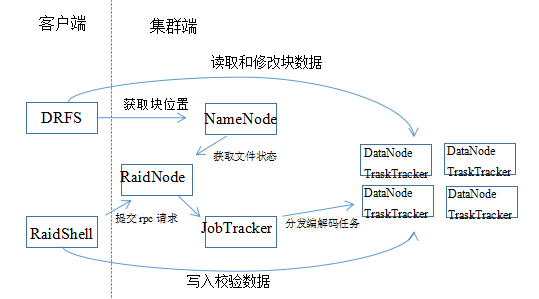


图 4.1 HDFS-RAID工作流程图

## 4.2 HDFS-EVENODD设计实现

### **4.2.1 在WINDOWS系统下的实现**

1、实现单个的矩阵丢失任意两列的恢复

其中实现矩阵水平校验的核心代码：将数组的每行异或存储为额外的一列。

**public** **static** **byte**[] horiExclusive\_OR(**byte**[][] dataCache) {

// **TODO** Auto-generated method stub

**int** l=***M***-1;

**byte**[] horiExculsive=**new** **byte**[l];

**for**(**int** i=0;i<l;i++){

**byte** temp=0;

**for**(**int** j=0;j<***M***;j++)

temp=(**byte**) (temp^dataCache[i][j]);

horiExculsive[i]=temp;

}

**return** horiExculsive;

}

实现矩阵对角线校验的核心代码：将数据的对角线的数据异或然后存储为额外的一列。

**public** **static** **byte**[] diagExclusive\_OR(**byte**[][] dataCache, **byte** commonFactor){

// **TODO** Auto-generated method stub

**int** l=***M***-1;

**byte**[] diagExclusive = **new** **byte**[l];

**for**(**int** i=0;i<l;i++){

**byte** temp=commonFactor;

**for**(**int** j=0; j<***M***;j++){

**int** t=*getMod*((i-j),***M***);

**if**(t>=l)

**continue**;

temp=(**byte**) (temp^dataCache[t][j]);

}

diagExclusive[i]=temp;

}

**return** diagExclusive;

}

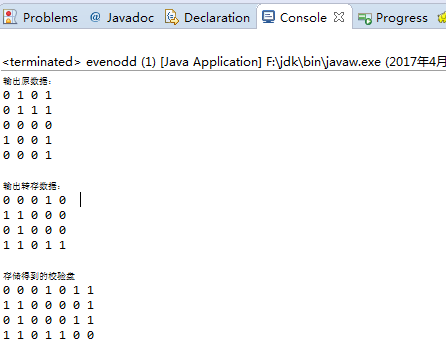


图 4.3 编码结果图

这里假设错的是D0和D5：

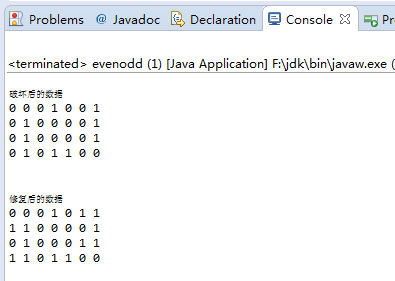


图 4.4 译码结果图

2、图片碎块丢失后恢复原图

其中实现文件分块的核心代码：用java自带的文件流读写文件，一次读取DISK\_SIZE(一个文件块的大小)大小的字节，放入ReadBuffer，并写入文件流，为文件碎块编号。

**public** **static** **void** split() **throws** IOException {

**int** blockNo=0;

String filePath="./1.jpg";

**byte**[] ReadBuffer=**new** **byte**[(**int**) *DISK\_SIZE*];//一次性读取 DISK\_SIZE 大小字节

//以二进制打开文件

DataInputStream fpr=**new** DataInputStream(**new** BufferedInputStream(**new** FileInputStream(filePath)));

**try**{

**while**(**true**){

DataOutputStream fpw=**new** DataOutputStream(**new** FileOutputStream("./2-"+blockNo+".jpg"));

fpr.read(ReadBuffer);

fpw.write(ReadBuffer);

fpw.close();

blockNo++;

**if**(blockNo>=***M***)

**break**;

}

}**catch**(EOFException e){

fpr.close();

}

}

实现文件碎块合并的核心代码：同样用java自带的文件流读写文件，遍历读取文件碎块，依次写入文件流，最后一块特殊化处理，将做运算补齐的字节去掉后再读入文件流，最后合并成的文件就是原文件。

**public** **static** **void** merge() **throws** FileNotFoundException,IOException {

String filePath="./3.jpg";

**int** blockNo=0;

**byte**[] WriteBuffer=**new** **byte**[(**int**) *DISK\_SIZE*];

**int** differ=(**int**) (*DISK\_SIZE* \* ***M*** - *length*); //去除补充的字节数

**byte**[] WriteBufferDiffer=**new** **byte**[(**int**)(*DISK\_SIZE*-differ)];

DataOutputStream fpw=**new** DataOutputStream(**new** FileOutputStream(filePath));

**try**{

**for**(**int** k=0;k<***M***-1;k++){

DataInputStream fpr=**new** DataInputStream(**new** BufferedInputStream(**new** FileInputStream("./2-"+blockNo+".jpg")));

fpr.read(WriteBuffer);

fpw.write(WriteBuffer);

fpr.close();

blockNo++;

}

//最后一块特殊对待

DataInputStream fpr=**new** DataInputStream(**new** BufferedInputStream(**new** FileInputStream("./2-"+blockNo+".jpg")));

fpr.read(WriteBufferDiffer);

fpw.write(WriteBufferDiffer);

fpr.close();

blockNo++;

}**catch**(EOFException e){

fpw.close();

}

}

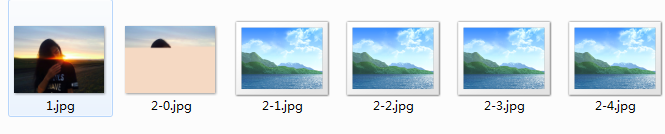


图 4.7 文件分块结果图

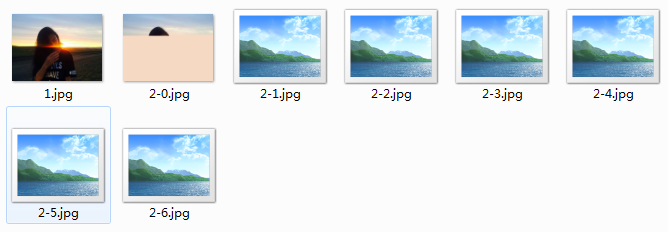


图 4.8 编码结果图

假设丢失2-2.jpg和2-6.jpg文件碎块。

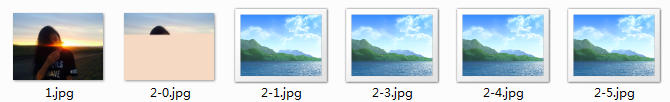


图 4.9 译码结果图

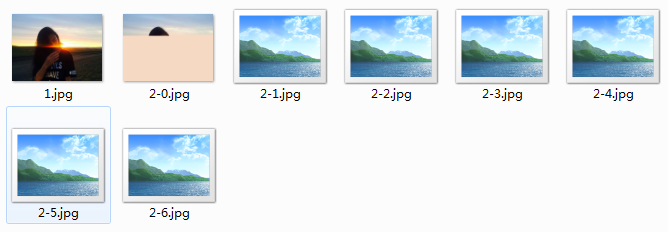


图 4.10 恢复丢失的文件块图

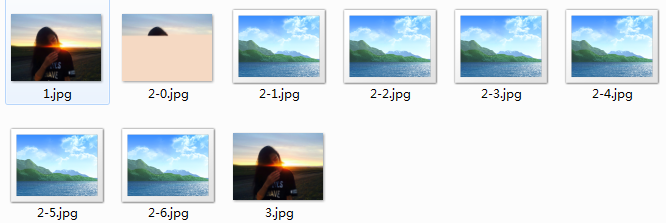


图 4.11 合并文件块恢复文件图

## 4.2.2 在HADOOP系统下的实现

Hadoop系统的底层机制在检测到坏块的时候，会返回错误，所以目前系统只能恢复一个坏块。因此，HDFS-EVENODD模块会基于EVENODD算法进行简化，只进行水平校验，因为只纠一个错，对角线校验已经失去了意义。

HDFS底层的数据存取结构如图4-1，每个数据文件块对应一个校验文件。stripeLength表示系统节点个数，parityLength表示校验节点个数。例如：stripeLength=4，parityLength=2，数据文件有8个block，那么对这个数据文件做RAID会产生4个parity block，这四个parity block被连接起来组成校验文件。数据文件和校验文件根据一定的规则存放，具体的根据配置决定。

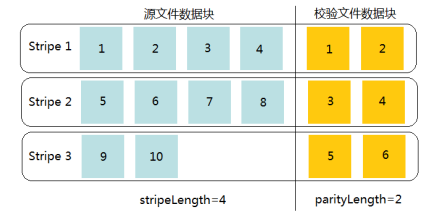


图 4.1 HDFS文件分块存放示意图

EVENODD编码在Hadoop中的实现需要继承类生成encodeBulk，decodeBulk类进行编码和解码。核心实现代码如下：

# 5 总 结

## 5.1 性能分析

## 5.2 总结

在整个毕业论文和设计的过程中，收获了很多。由于论题和本科所学知识结合度不高，所以在新的知识方面做了很多的功夫，比如了解基本的纠删码专业名字概念，码距、码集、最小码距等，还有纠删码的一些性质，例如MDS阵列，以及纠删码涉及的数学知识，有限域的分析等。在论文研究期间，经过很多瓶颈，但幸运的是最后能成功的完成毕业设计。

# 致 谢

# 在此要感谢我的导师韩鸿宇老师在这期间对我的指导，从帮助找论题资料，到修改论文，一直都非常的耐心负责。还有西南交大的老师杨彬老师，在专业上给了我非常大的帮助，讲解Hadoop的文件系统存取方式，以及HDFS-RAID模块是实现流程等

# 参考文献

1. [] Yu Zheng, Licia Capra, Ouri Wolfson, Hai Yang. [Urban Computing: concepts, methodologies, and applications](https://www.microsoft.com/en-us/research/publication/urban-computing-concepts-methodologies-and-applications/). ACM Transaction on Intelligent Systems and Technology. 5(3), 2014 [↑](#endnote-ref-0)
2. [] 城市计算概述 郑宇. [城市计算概述](http://rmc-stage/pubs/211950/%E5%9F%8E%E5%B8%82%E8%AE%A1%E7%AE%97%E6%A6%82%E8%BF%B0-%E9%83%91%E5%AE%87.pdf" \t "https://www.microsoft.com/en-us/research/project/%e5%9f%8e%e5%b8%82%e8%ae%a1%e7%ae%97/_self)，武汉大学学报. 2015年1月，40卷第一期 [↑](#endnote-ref-1)
3. [] BLAUM M，BRADY J，et al. EVENODD:an efficient scheme for tolerating double sisk failures in RAID archeitectures[J].IEEE Trans.Comput,1995,44(2):192-202 [↑](#endnote-ref-2)
4. [] 基于大数据的网络用户行为分析 左军，[《软件工程师》.2014年10期](http://www.cnki.com.cn/Journal/I-I2-ZGGC-2014-10.htm" \t "http://www.cnki.com.cn/Article/_blank)  [↑](#endnote-ref-3)
5. [] IDC：2011年全球数据产生量达到1.8ZB 到2020年将增长50倍 新浪科技IDC.2011年06月29日 [↑](#endnote-ref-4)
6. [] D.A.Patterson,G.A.Gibson,R.H.Katz.”A Case for Redundant Arrays of Inexpensive Disks(RAID)” In:Proc.of ACM SICMOD,June 1998.109-116 [↑](#endnote-ref-5)
7. [] Gopalan, P., Huang, et al. On the Locality of Codeword Symbol[J]. IEEE Transactions on Information Theory, 2012, 58(11): 6925-6934. [↑](#endnote-ref-6)
8. [] 付园. 基于HDFS的优化数据冗余策略的研究[D]. 硕士论文. 吉林大学，2014 [↑](#endnote-ref-7)
9. [] XU L. Highly available distributed storge systems[D]. Pasadena, California: California Institute of Technology, 1997. [↑](#endnote-ref-8)
10. [] KATTI R, RUAN Xiao-yu. S-code: New distance-3 MDS array codes with optimal encoding[C]//In: Proceedings of IEEE ICASSP’05. Piscataway, NJ: IEEE Signal Processing Society,2005. [↑](#endnote-ref-9)
11. [] https://wiki.apache.org/hadoop/HDFS-RAID [↑](#endnote-ref-10)