第46卷第3期 中国科学技术大学学报 Vol. 46, No. 3
2016年3月 JOURNAL OF UNIVERSITY OF SCIENCE AND TECHNOLOGY OF CHINA Mar. 2016

文章编号:0253-2778(2016)3-0188-12

BDCode:一种面向大数据存储系统的纠删码算法

殷 超[1],王健宗[2,3],吕海涛[1],崔宗敏[1],程良伦[2],李同芳[1],刘 妍[1]

(1.九江学院信息科学与技术学院,九江 332005;2.广东工业大学计算机学院,广州 510006;
3.平安科技(深圳)有限公司,深圳 518029)

摘要:针对含有大量数据的大数据存储系统,提出了一种基于编码技术的面向大数据备份的优化算法(BDCode).该算法通过对不同编解码服务器设置不同的虚拟节点存储组来保证系统的可用性,节点和数据块的并行解码计算提高了系统中数据损坏时的恢复效率.实验表明,所提出基于编码的大数据系统备份机制可以提高系统的存储利用率,并行解码方式的引入能加速减少数据损坏时的恢复时间,并能达到零号的系统负载均衡;此外不同的用户设置不同的编码参数,增加了大数据系统的鲁棒性.实验通过设置不同的数据块 m 和校验块 k 的比例来提升利用率,并行解码速度相比以前的串行提高近两倍.使用 BDCode 比 CRS 编码效率平均高 36.1%,解码效率平均高 19.3%;比 RS 码编码效率平均高 58.2%,解码效率平均高 33.1%.

关键词:分布式存储系统;纠删码;大数据;鲁棒性;可用性;云存储

中图分类号:TP311 **文献标识码**:A doi:10.3969/j.issn.0253-2778.2016.03.003

引用格式:YIN Chao, WANG Jianzong, LV Haitao, et al. BDCode: An erasure code algorithm for big data storage systems[J]. Journal of University of Science and Technology of China, 2016,46(3):188-199.
殷 超,王健宗,吕海涛,等. BDCode:一种面向大数据存储系统的纠删码算法[J]. 中国科学技术大学学报,2016,46(3):188-199.

BDCode: An erasure code algorithm for big data storage systems

YIN Chao[1], WANG Jianzong[2,3], LV Haitao[1], CUI Zongmin[1],
CHENG Lianglun[2], LI Tongfang[1], LIU Yan[1]

(1. *School of Information Science and Technology, Jiujiang University, Jiujiang* 332005, *China*;
2. *School of Computer Science and Technology, Guangdong University of Technology, Guangzhou* 510006, *China*;
3. *Ping An Technology (Shenzhen) Co., Ltd, Shenzhen* 518029, *China*)

Abstract: An optimized algorithm, based on erasure coding technology towards the big data storage system that contains a lot of data, was proposed. By studying existing coding technologies and big data systems, this algorithm, named BDCode (big data code) can not only protect system reliability, but also improve the security and the utilization of storage space. Due to the high reliability and space saving rate of coding technology, coding mechanisms were introduced into big data systems. The storage nodes are divided into many virtual nodes to realize load balancing. By setting different virtual nodes' storage groups for different codec servers , ensure system availability. And by using the parallel decoding computing of the nodes and

收稿日期:2015-08-27;**修回日期**:2015-12-01

基金项目:国家自然科学基金(61462048),国家自然科学基金广东省联合基金重点项目(U2012A002D01),江西省科技厅科技项目(GJJ151081)资助.

作者简介:殷超,男,1980年生,博士/副教授.研究方向:云存储,大数据,纠删码,节能. E-mail: david_yin@jju.edu.cn.

通讯作者:王健宗,博士/工程师. E-mail: jzwang@188.com

the block of data, we can be ensured the recovery efficiency of the system can be proved when data is corrupted. Additionally, different users setting different coding parameters can improve the robustness of big data storage systems. We configured various data block m and calibration block k to improve the utilization rate in the quantitative experiments. The results show that parallel decoding speed can be nearly two times faster than the past serial decoding speed. The encoding efficiency with BDCode coding is, on average, 36.1% higher than using CRS and 58.2% higer than using RS coding. The decoding rate by using BDCode averages 19.3% higher than using CRS and 33.1% higher than using RS.

Key words: distributed storage system; erasure coding; big data; robustness; availability; cloud storage

0　引言

随着数据量不断爆炸式增长，增加了对“数据宝藏”挖掘的需求，甲骨文、IBM、微软和 SAP 等公司花了超过 15 亿美元投入的在大数据数据管理与分析应用. 数据存储规模已达到从 TB 级别到 PB 级，乃至 ZB 级，呈海量飞跃式发展[1]. 图灵奖获得者 Gray 指出，网络环境下每 18 个月产生的新数据量等于有史以来所有老数据量之和，因此，将数据集中采用存储服务器存放的传统方式已经不能满足现在的海量数据存储需求，本地存储在扩容方面存在很多限制，如成本、技术、通讯等，使得数据存储成为制约信息技术发展的主要瓶颈. 在此种情况下，数据以分布式形式进行存储，由本地存储的形式延伸到云存储成为数据存储一个必然的发展方向. 分布式存储利用了存储技术和传输技术的优势，在数据安全性，存储容量，容灾备份方面有着本地存储无法比拟的优势. 大数据的存储，目前存在以下一些问题需要解决[2-5]：

容量问题：通常可达到 PB/ZB 级的数据规模，要求海量数据存储系统也一定要有相应等级的扩展能力. 与此同时，存储系统的扩展一定要简便，仅通过增加模块或磁盘柜来增加容量，甚至实现在线扩容，减少降级服务时间并保证服务性能.

延迟问题：“大数据”应用还存在实时性的问题. 特别是涉及与网上交易或者金融类相关的应用. 有很多“大数据”应用需要较高的 IOPS 性能，如 HPC 高性能和数据依赖的计算.

安全和隐私问题：大数据分析往往需要收集各方面的数据，如美国“棱镜门”便是对数据隐私的严重侵犯，对于公司/个人而言，保证数据的隐私和安全性不受到侵犯是其考虑的首要问题.

成本问题：想控制成本，就意味着我们要让每一台设备都实现更高的“效率”，同时还要以最少的成本保证最佳的可靠性. 目前，像纠删码、重复数据删除、云存储等技术已经进入到主流存储市场，可以处理更多的数据类型，这都可以减小大数据存储应用的成本，提升存储效率.

大数据可靠性与空间利用率优化依靠的主流技术包括纠删码和重复数据删除. 目前，我们所知道的云系统，如 GFS[6]、HDFS[7]、Amazon S3[8] 和 Ceph[9]，基本都使用多副本技术. 对比多副本和纠删码两种冗余技术，纠删码因为其在给系统提供相同可靠性时需要的空间更少，因而更适合大数据备份系统. 本文提出一种基于纠错码的大数据备份优化算法，称为 BDCode (big data code)来存储备份数据，该算法在原先的 CRS[10] 和 Reed-Solomon [11]算法上进行改进，通过矩阵行列式的优化，减少 CRS 矩阵里面的单位元素 1 的数量，通过提高矩阵运算的速度来提升整个系统的性能.

基于 BDCode 的大数据备份系统的底层使用类似点对点(peer to peer, P2P)的无中心节点的存储系统，主要原因包括以下两点. ①BDCode 的主要目标是瞄准不需要频繁更新的备份系统应用；②BDCode 是为了保证保证系统里面 99.9%的读写操作能够快速完成. 与请求路由经过多节点架构的系统[112-14]不同，BDCode 的每个节点本地维护足够的路由信息，直接路由请求到其他精确的节点，因此它增加了路由的响应时间，从而相对增加了延时.

我们使用一致性哈希函数将数据存储在分布式系统中. 相比用其他方式来存储数据，一致性哈希函数更适合目前这种状况. 首先这种方式适合存储相对小的数据，一般的都是小于 1M；其次一致性哈希函数易于实现系统鲁棒性. 哈希的使用不仅提高了查找的效率并且能够保证节点的地址不冲突，例如 Amazon Dynamo[15]. 目前，备份数据的保存需要在保证可靠性的前提下尽量减少存储消耗. 本研究目标是最大程度降低存储消耗和提高系统性能，本

文主要贡献包括：

(Ⅰ)设计名字节点和数据节点这种无中心的端到端架构，这种架构比基于中心控制的设计更加对称；将P2P框架引入大数据环境中，使得系统不会因为其中某一个(或几个)节点损坏的影响而导致系统崩溃；

(Ⅱ)提出大数据系统纠错码算法BDCode，该算法是在基础的CRS方案上经过修改形成的，利用矩阵计算加速编解码和修复时间.

实验表明，使用BDCode编码使得系统性能有很大提升，BDCode比CRS编码效率平均高34.2%，解码效率平均高18.1%；比RS码编码效率平均高56.5%，解码效率平均高31.1%.

1 相关工作

1.1 编码技术

目前在分布式备份存储系统中，纠删码被广泛应用于大规模存储系统，如Facebook[16-17]和谷歌系统.谷歌文件系统升级版(Colossus)[18]已经引入RS编码，并且很多相关纠删码的工作已经实施并从各个方面提升了系统性能，但是相比CRS编码，RS编码的性能要低的多.

LRC编码[19]是在Windows Azure系统里面使用的编码.它的基本思想是使用额外的数据块来构造额外的校验链，这样每条校验链上的块数比用RS和CRS编码的要少.这样一条校验链上的fragments相对于BDCode来说会比较少，使得能在离线的情况下恢复数据的时候读取数据比较少，但是也增加了很多额外的校验盘，增加了存储开销.

RS编码、CRS编码及BDCode编码，都属于最大距离可分码(maximum distance separable, MDS)[20].MDS码是使得对于某个给定容错只能达到最小存储的编码.目前为止还有很多也属于这个范畴的编码，如Hover编码[21]，Weaver编码[22]和X码[23]都属于MDS码，但是这些编码主要适用于阵列结构的系统.

此外，Yin等[24]通过对云系统的研究，提出分布式的功能性修复方式，大大降低了系统出错时候读取数据量，同时也简化了系统的维护工作.Rashmiet等[25]用最小存储再生码(MSR)作为分布式系统的编码，在减少磁盘I/O消耗的同时保留最优的存储、可靠性和网络带宽.

1.2 重复数据删除

重复数据删除[26-27]对于分布式备份系统是一种目前主流且非常热门的技术，能够对存储容量进行有效优化.它通过删除数据集中重复的数据，消除冗余数据.

一类重复数据删除是基于引用计数[26]，另一类是当系统出现错误环境时，在存储单元里实现垃圾回收.MAD2[28]就是在删除过程中引用计数的一个系统，而Dedupv1[13]就是第二类使用垃圾回收机制来标记未被引用的块，从而进行重复数据删除工作.我们在大数据存储系统中采用Hash算法，是重复数据删除技术是实际应用，目的是为了能够去除重复相同的哈希值.

1.3 分级存储系统

HACFS[14]中提出一种自适应编码，通过RAID结构分层.AutoRAID[5]在存储控制器内提供了二级存储结构.它在不同RAID等级之间自动迁移数据，提供高I/O性能的活跃数据和低存储开销的不活跃数据.同样，HACFS在快码和简码之间迁移数据，在分布式存储中减少额外恢复的网络传输.DiskReduce[15]提出了通过在复制和纠删码的存储级之间异步迁移的分级存储.BDCode也借鉴分级存储的技术优化存储开销和恢复成本.

2 RS、CRS和BDCode理论分析

2.1 RS编码

RS码全称里德所罗门码.由Reed和Solomon于1960年提出并构造，是一类纠错能力很强的特殊的非二进制BCH码[28].最初用于通信领域，具有较强的容错能力.RS码是一种具有MDS性质的编码，具有理论上最优秀的容错能力.由于码长为n信息长度为k的码的最大汉明距离为$n-k+1$，所以在这种意义下RS码是一种最优的编码方法，其编码矩阵是范德蒙码.

定义2.1 若选取编码矩阵为$\boldsymbol{G}_{k\times n}$，使得$\boldsymbol{G}^{\mathrm{T}}=(g_{i,j})$，其中$g_{i,j}=g_i^{j-1}$，$g_i\in \boldsymbol{G}(p^r)$，$p$为素数，$r$为正整数，则称所得纠删码码为范德蒙码，$\boldsymbol{G}$的任意$k$列组成方矩阵的转置矩阵为范德蒙矩阵，若$x_k(i=1,2,\cdots,k)$互不相同，则$|(\boldsymbol{G}')^{\mathrm{T}}|\neq 0$，即$\boldsymbol{G}$的任意$k$列组成的方矩阵为非奇异的，因此这样的矩阵满足纠删码中生成矩阵的特性.范德蒙矩阵形如下：

$$\boldsymbol{V}_n=\begin{pmatrix}1 & 1 & \cdots & 1\\ a_1 & a_2 & \cdots & a_n\\ a_1^2 & a_2^2 & \cdots & a_n^2\\ \cdots & \cdots & \cdots & \cdots\\ a_1^{n-1} & a_2^{n-1} & \cdots & a_n^{n-1}\end{pmatrix}.$$

由以上的生成矩阵 $\boldsymbol{G}$ 进行编码，$y=x\boldsymbol{G}$，其中 $x=(x_0,x_1,\cdots,x_{k-1})$ 为源数据，这种编码一定程度上可以增加信息的保密性，但也增加了编解码的复杂度，并不十分适用，所以线性码的生成矩阵 $\boldsymbol{G}$ 的构造很多时候并不直接适用范德蒙矩阵作为生成矩阵，而是常常使用 $k\times k$ 单位矩阵和 $k\times(n-k)$ 的范德蒙矩阵联合构成.

RS 码在伽罗华域(GF)中运算. 其编码方法为：有 k 个数据块 $D_1,D_2,D_3,\cdots,D_7$，使用一个 $(k+m)*k$ 的生成矩阵来乘以数据块列向量，这里的乘法是 GF 域上的乘法. 生成矩阵中，前 k 行是一个单位矩阵，用以保留数据块，剩余的 m 行构成校验块，如图 1 所示. 编码后的码组可以表示成一个 $k+m$ 的列向量 $\boldsymbol{D}|\boldsymbol{C}$，其中 $C_1,C_2,\cdots,C_m$ 为校验块. 解码过程是将 $\boldsymbol{D}|\boldsymbol{C}$ 中损坏的块删除，同时删除其对应的生成矩阵中那一行. 再对删除行后的生成矩阵求逆，与删除损坏块的 $\boldsymbol{D}|\boldsymbol{C}$ 相乘，如此可得原始数据的数据块信息. 解码过程可以修复数据块和校验块.

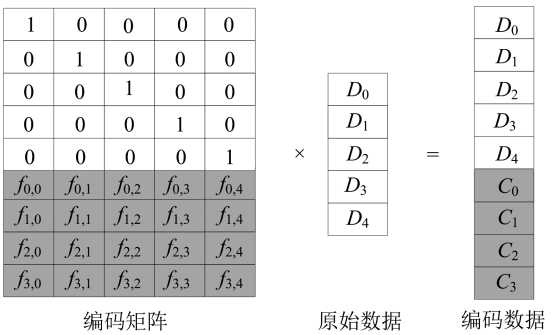

图 1　数据编码示意图

Fig. 1　Data coding

2.2　CRS 编码

使用柯西矩阵作为编码矩阵的 RS 编码称作柯西 RS 编码. 柯西矩阵有助于提高编解码的效率.

柯西矩阵的数学定义如下：

定义 2.2　设 F 是一个有限域，$\{x_1,\cdots,x_m\}$ 和 $\{y_1,\cdots,y_n\}$ 是 F 中元素集合的两个子集，且满足：

①$\forall i\in\{1,\cdots,m\},\forall j\in\{1,\cdots,n\}$：$x_i+x_j\neq 0$.

②$\forall i,j\in\{1,\cdots,m\}$，$i\neq j$：$x_i\neq x_j$ 且 $\forall i,j\in\{1,\cdots,n\}$，$i\neq j$：$y_i\neq y_j$.

那么，矩阵

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

称作在有限域 F 上的柯西矩阵.

CRS 编码对 RS 编码进行了改进. 采用了不同结构的生成矩阵，使用柯西矩阵而不是范德蒙矩阵[3]. CRS 编码消除了复杂的乘法运算，转化用异或操作[13].

2.3　BDCode 编码

本文在 CRS 的基础上对编码矩阵进行变换，希望得到效率更高的矩阵. 假设在 GF(2^w)中，令 $w=3$，n 为数据块的个数，k 为校验块的个数.

定义 2.3　定义一个 k 行 n 列的柯西矩阵，k 行的数据表示为 $X_1,X_2,\cdots,X_n$，n 列的数据表示为 $Y_1,Y_2,\cdots,Y_k$. 这个柯西矩阵可以定义为

$$\boldsymbol{M}(X,Y)=\sum_{i=1}^{k}\sum_{j=1}^{n}\boldsymbol{V}(X_i,Y_j).$$

其中，$\boldsymbol{V}(x,y)$ 表示矩阵 $\boldsymbol{M}(1/(x+y))$，在伽罗华域($w=3$)时矩阵如图 2 所示.

	0	1	2	3	4	5	6	7
0	-	7	3	4	2	3	4	1
1	7	-	4	3	3	2	1	4
2	3	4	-	7	4	1	2	3
3	4	3	7	-	1	4	3	2
4	2	3	4	1	-	7	3	4
5	3	2	1	4	7	-	4	3
6	4	1	2	3	3	4	-	7
7	1	4	3	2	4	3	7	-

图 2　柯西矩阵示意图

Fig. 2　Cauchy matrix

图 2 中，$X=\{1,2\}$ 代表阴影部分的列，$Y=\{0,3,4,5,6\}$ 代表阴影部分的行. 整个柯西矩阵就是通过矩阵变换实际得到的 1 的数量就是该矩阵中所有阴影部分格子的数量.

本文的目标实际是确定 X 和 Y 的个数，即 n 和 k 的值，使得整个矩阵的值 $\boldsymbol{M}(X,Y)$ 最小；因此在 GF(2^3)中，采用枚举法来证明. 枚举的具体值如表 1

所示.

表1 枚举的相关 n 值和 k 值

Tab.1 Enumeration of related n and k values

n 值	k 值
6	2
5	2
5	3
4	2
4	3
4	4
3	2
3	3
2	2

表1结果表明,当 $n=k$ 时整个矩阵的1数量最少,因此BDCode系统会比CRS系统的性能好.

3 BDCode大数据存储介绍

3.1 BDCode编码实现

编码矩阵不同会导致CRS码的编解码时间不同.本模块中用到BDCode编码,是在普通的CRS编码上进行优化得到的,CRS由于其编码矩阵可以通过求最少"1"的优化方法大幅降低编码时间,因此采用额外的矩阵变换来改造矩阵,生成的BDCode的柯西矩阵能够使系统具有更快的修复时间,并且计算次数减少.

BDCode系统矩阵的生成过程分成三步:

(Ⅰ)生成最基本的柯西矩阵 $\boldsymbol{M}$

使 $\boldsymbol{M}[i,j]=1/(i\oplus(m+j))$,其中,除法操作是在有限域中进行,加法操作是常规的整数加法.

(Ⅱ)让编码矩阵的首行为全1

对于每一列 j,用该列所有的元素除以列首元素 $\boldsymbol{M}[0,j]$,其中除法操作在有限域中进行.

(Ⅲ)依次优化编码矩阵的每一行

优化矩阵的目的是降低每一行对应的位矩阵中1元素的个数.首先,统计每一行对应的位矩阵中1元素的个数;然后,将该行依次试着除以每一个元素,并统计经过除法变换后位矩阵表示中1元素的个数;在所有变换后的矩阵中选择1元素最少的矩阵.再对矩阵的每一行都进行这样的优化,降低位矩阵1元素个数,从而减少系统的编解码开销.

例如,假设 $k=m=w=3$,经过第一步得到的柯西矩阵为

$$\begin{pmatrix}3&5&4\\2&4&5\\6&8&1\end{pmatrix} \tag{1}$$

按照第二步的优化,用每一列除以列首元素,可得到首行全1的编码矩阵为

$$\begin{pmatrix}1&1&1\\5&9&4\\2&3&6\end{pmatrix} \tag{2}$$

接下来按照第三步进行优化.

统计次行在位矩阵表示中1元素的个数为7,8和5.现在分别将该行的数据除以5、9和4,并分别统计位矩阵表示中1元素的个数,得到15、18和14;那么我们选择对该行除以4,得到更优的矩阵表示.

同理,最后一行除以2,得到更优的矩阵表示.

$$\begin{pmatrix}1&1&1\\1&7&3\\6&5&1\end{pmatrix} \tag{3}$$

这个编码矩阵的位矩阵表示中有29个1元素,相比较原始矩阵的43个,有明显的改善.采用BDCode编码模式会比传统的CRS具有更高的编解码效率,数据块的读写效率更高.

3.2 BDCode系统设计

本文的设计结合备份存储系统的特点,在保持编码方法高存储利用率,数据隐私保护等特性的同时,通过对系统结构进行设计来修正编码方法响应时间等性能的不足.

如图3所示,系统中主要有两种节点:名字节点和数据节点.名字节点中存放着所有文件及文件路径的元数据,并且会根据策略对数据进行编码并选取使用的数据节点,控制整个系统的负载均衡.数据节点中保存着编码后的用户数据.系统中的数据节点均被分割成多个微型节点,这样可以使得系统的负载更易均衡,同时可以提升系统的并行能力以及用户间的独立性.数据节点中的每个微型节点拥有一个唯一的标记(ID)等信息.每个微型节点可以存放一个数据块或多个数据,微型节点的ID可用于记录微型节点属于哪一个编码集合.用户与系统的交互主要通过名字节点进行.

用户将自己的数据以及要求提交给名字节点后,名字节点会根据用户的要求对用户的数据进行分块并编码,生成带有冗余的编码块,所有的编码块

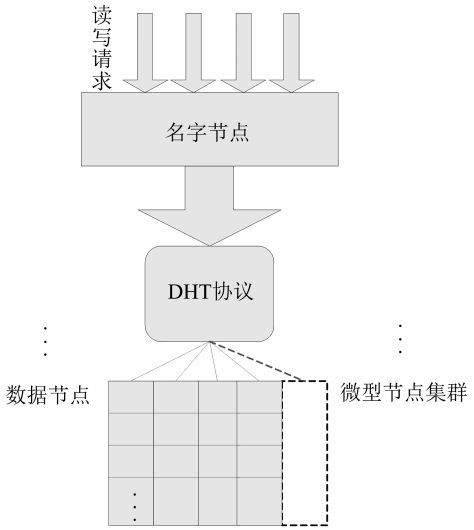

图 3　系统架构图

Fig. 3　System frame

构成一个编码集合. 编码集合是非常重要的信息，在检索用户存放数据的位置，修复损坏的数据等操作中都起着很重要的作用，因此被存放在名字节点中. 在编码完成后，名字节点会在系统中的不同的数据节点上选取空闲的微型节点存放编码块. 出于数据安全性以及负载的均衡，每一个编码集合在每一个数据节点上仅会选取一个微型节点用于存放数据，这样即使发生数据节点整个崩溃的情况数据也不会丢失，同时可以使系统在各数据节点上的负载平衡. 选择完成后，名字节点会将被选择的微型节点的信息以及编码块在微型节点中的位置记录在该文件的编码集合中. 这样用户就可以通过名字节点访问相应的数据节点来获得自己存放的文件. 系统在发生数据损坏时，也可以通过名字节点获得损坏文件编码集合中幸存的编码块以修复损坏的数据.

3.3　节点设计

本文提出的系统，对名字节点和数据节点都进行了精心的设计. 相对与使用备份方法和常规的编码方法的分布式系统，名字节点和数据节点的结构更为复杂. 系统对数据节点进行了划分，将其虚拟为多个独立的微型节点，数据节点为自己的微型节点分配唯一的 ID、地址等参数，使它们可以被独立访问. 这使得数据节点相对于常规系统需要具备以下能力：

（Ⅰ）数据节点具备能够自动对自身进行划分的能力. 系统对加入的数据节点进行初始化时，数据节点会对自己进行划分，将自己按照整个节点的存储空间和计算能力划分为若干个微型节点.

（Ⅱ）由于每个微型节点具备唯一的 ID、地址等参数，可以被独立访问. 故数据节点还需要能够为属于自己的微型节点生成唯一的 ID，同时为其分配地址和相应端口，使得名字节点可以直接访问该微型节点.

（Ⅲ）对数据节点进行划分有一个很重要的理由就是划分后各微型节点可以并行修复，这就要求每一微型节点的健康状态都可以随时被系统得知，因此数据节点还要具备对微型节点进行校验的能力.

由于在系统中对数据节点的结构进行了分割，将其分为多个微型节点. 相应地为了支持数据节点的改变，名字节点也需要对微型节点进行支持，同时应具备以下能力：

（Ⅰ）出于对于数据安全的考虑，系统在选择微型节点时有固定的策略，即同一个编码集合的编码块在同一个数据节点上的仅会选择一个微型节点；在此基础上随机选择微型节点以均衡系统的负载.

（Ⅱ）由于编码集合中以微型节点为最基本单位，所以系统要求名字节点不仅能够记录数据节点的信息，还要求能够记录每个单独微型节点的信息并能直接访问它们. 数据节点为微型节点分配了独立的 ID 以及地址等参数，名字节点要能够在编码集合中记录属于集合的微型节点的各项信息.

（Ⅲ）系统能够根据用户的要求提供不同的编码，这就要求名字节点兼容不同种类的编码. 同时要求能够根据要求自动选择适应的编码，拥有编码选择策略.

（Ⅳ）当数据丢失时，系统应该具备找回丢失数据的能力. 这就要求名字节点具备错误发生时定位错误所属编码集合，由数据节点提供的幸存数据进行解码操作获得丢失的数据并将其写回数据节点的数据修复能力.

（Ⅴ）系统需要能够对每个微型节点进行单独访问的能力，这就要求了系统能够依据唯一 ID 访问对应的微型节点的能力，可以通过 DHT 来实现.

明确了系统对名字节点和数据节点额外的要求，需要对名字节点和数据节点的软件结构进行设计. 编码方法本身会带来额外的计算和带宽消耗，系统的性能与备份方法相比仍然存在一定的差距. 为了解决这些问题，满足上述对名字节点和数据节点的要求，对系统进行优化，使得系统能够通过软件设计弥补性能损失. 在软件上，系统主要有图 4 中几个模块来各司其责，分别提供系统对于名字节点和数

据节点的要求.

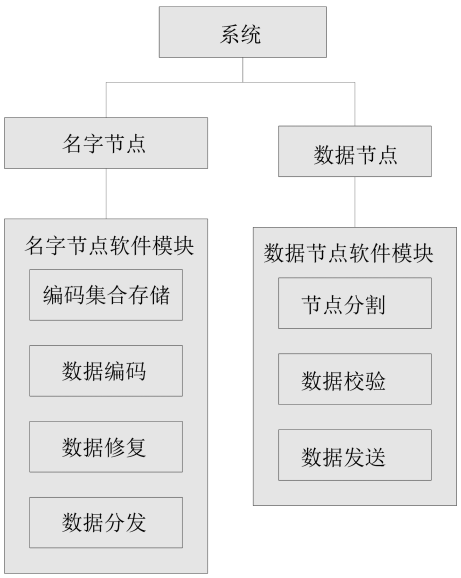

图 4 节点设计图

Fig. 4 Node design

4 BDCode 运行机制

4.1 编解集合存储

用户的数据被分割成块,编码后发送到不同的数据节点,一个文件经过编码后生成的所有块(数据块,校验块)被称为一个编码集合. 编码集合是系统中一个虚拟的概念,编码集合的信息存储在名字节点中. 在系统中,数据节点被分为很多个微型节点. 每个微型节点可以存放一个数据块或多个数据. 无论是 RAID 5/6 纠删码或其他网络编码,都是对若干个数据块进行编码计算,以生成校验块或编码后数据块的方法对数据添加冗余,使得系统具有容错能力. 在任意 N 个数据块(N 为一个固定值,视编码技术的不同而变化)的数据丢失时,由所有其他的块为基础进行解码操作即可重新生成丢失的数据块,使得系统恢复到数据丢失前的健康状态.

在用户数据被分块,编码并且发送到被选择的微型节点中,名字节点会将所有的微型节点信息记录写在一个存储集合中. 存储集合中记录以下信息:微型节点列表 slots;微型节点个数 N;文件大小 S.

根据编解码服务器大小 M 和名称 Name 选取一组虚拟节点作为数据节点,然后加上校验的冗余节点构成 N 个虚拟节点组成的存储组. 存储组内的节点不能在同一物理节点上,所以编解码服务器的创建主要是选取存储组过程的设计.

如图 5 所示,虚拟节点经过 IP 和端口号的 Hash 已经能均匀随机的分布到环上,整个节点结构已经比较随机,所以在选取存储时可以根据编解码服务器的名称进行 Hash,根据 Hash 值按照 Chord 的方式选取对应的虚拟节点;然后在此虚拟节点下依次选取 N 个未被占用的后继结点,在选取的过程中比较该虚拟节点与已选取的存储组的节点是否属于同一物理节点,如果是则继续向后查找,直到找到 N 个虚拟节点为止.

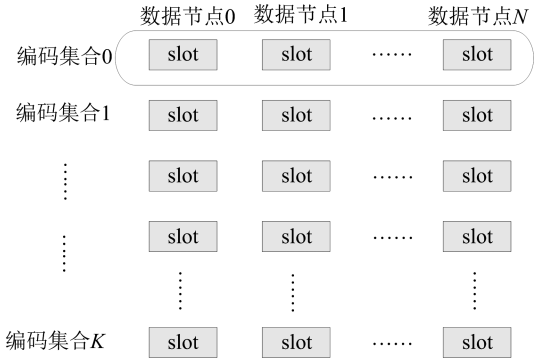

图 5 存储集合的选取

Fig. 5 Selection of storage sets

编码集合的相关信息存放在名字节点中(集合内微型节点的 ID 和地址等信息),由名字节点维护. 当数据节点发现存在不健康的微型节点时,名字节点会根据编码需要向编码集合内其他的微型节点发出数据请求命令来修复丢失的数据并写入损坏的微型节点中. 同样,在用户访问自己的数据时,也需要通过编码集合找到所需要的数据块所处的位置,因此编码集合对于系统非常重要,是用户以及系统操作文件的接口. 编码集合存储模块主要负责存储并维护编码集合的信息. 同时在用户提交请求时能够快速地提供相关数据块所在微型节点的 ID 以及地址等.

假设用 $\mathrm{Cap}_{\mathrm{name}}$ 表示名字节点的存储数据量大小,$\mathrm{Cap}_{\mathrm{block}}$ 表示数据块的大小,$\mathrm{Num}_{\mathrm{block}}$ 名字节点内数据块的个数,$\mathrm{Num}_{\mathrm{parity}}$ 为编码组内校验数目,则编码次数为

$$\mathrm{Num}_{\mathrm{ecode}} = \sum_{i=1}^{n} \mathrm{Cap}_{\mathrm{name}} / (\mathrm{Cap}_{\mathrm{block}} * \mathrm{Num}_{\mathrm{block}}) \quad (4)$$

所需的总存储空间为

$$\mathrm{Cap}_{\mathrm{total}} = \mathrm{Cap}_{\mathrm{name}} * \mathrm{Num}_{\mathrm{parity}} / \mathrm{Num}_{\mathrm{block}} \quad (5)$$

4.2 数据分割

对数据节点划分后,各微型节点可以并行修复,这就要求了每一微型节点的健康状态都可以随时被系统得知,因此,数据节点还要具备对微型节点进行

校验的能力.

微型节点是系统中一个虚拟的概念，用以提升系统的并行能力、灵活性以及性能. 在本系统中，每个数据节点会被划分成若干个微型节点，每个微型节点相当于一个通用的容器，该容器可以成为所有系统支持编码的基本单位，存放一个数据块或多个大小的数据. 微型节点中除存放有数据块的数据外，还存放有 ID、地址、是否使用标记、健康状况、校验、使用百分比等信息.

由于每个微型节点具备唯一的 ID、地址等参数，可以被独立访问，因此数据节点还需要能够为属于自己的微型节点生成唯一的 ID，同时为其分配地址和相应端口，使得名字节点可以直接访问该微型节点.

ID 对于每一个微型节点都是唯一的，用于标示系统中所有的槽. 地址是为了方便系统访问所需的微型节点，提高微型节点的独立性，为系统提供了独立访问每个微型节点的能力，可以说每个微型节点都是一个微型的数据节点. 是否使用标记用来记录微型节点是否已经被使用，避免覆盖数据造成数据丢失. 健康状况是为了使得系统能够迅速地获取系统失效的情况，确定修复策略(微修复或全面修复)，实现修复的并行化，系统为每一个微型节点添加了一个简况标记，用以监视系统中所有微型节点的健康状态，迅速地选取修复策略，并且能够实现多微型节点并行修复，降低系统的修复时间，提高系统可用性. 校验就是系统会对每个微型节点中的数据添加一个校验值来监视微型节点中数据的健康状况. 在检查操作中，系统会利用校验值对系统中的数据进行完整性检查. 若数据不合法则将健康状况标记置为“失效”，触发修复机制对微型节点中的数据进行修复.

微型节点的设计使得系统对于每个数据节点的负载分配更为科学，可以使得数据节点的负载均衡. 对数据节点的健康状况了解更为清晰，能够精确到某一部分而不是整个节点. 同时系统对数据节点的访问粒度更精细，地址使得系统可以单独访问微型节点中的数据. 校验使得系统可以获得每个单独的微型节点的健康状况. 微型节点对数据节点的虚拟化使得系统对数据节点的操作更为灵活，不同的微型节点可以选择不同的编码技术对其进行编码. 在整个数据节点发生损坏或多个微型节点丢失数据时，系统可以同时对多个微型节点进行并行修复，这大大提升了系统的修复速度，因此也提升了系统的可用性.

图 6 为一个数据节点被分割为多个微型节点的过程. 首先数据节点按照存储空间以及计算能力分割为多个微型节点，每个微型节点包括 ID、地址、是否使用标记、健康状况、校验、使用百分比等关键信息. 在数据节点第一次被划分的同时，所有微型节点的各项参数都被初始化.

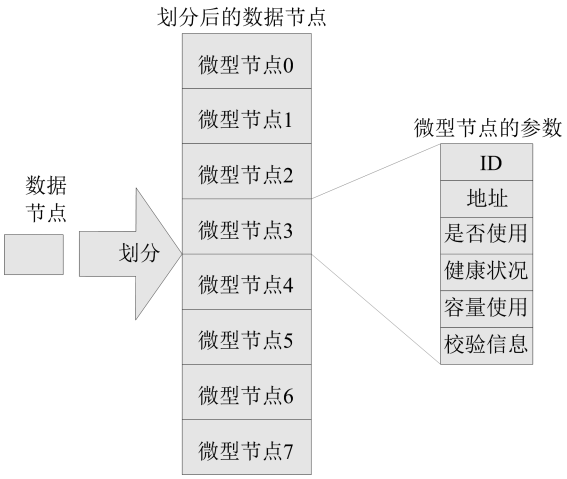

图 6　节点分割流程

Fig. 6　Process of node segmentation

4.3　错误修复

数据修复模块主要负责在系统发生数据丢失时对数据进行修复. 在发生系统错误时，数据修复模块首先会尝试微修复方法，在发现微修复不可行后采用全节点修复. 修复时，数据修复模块从编码集合存储模块读取编码集合信息，从数据发送模块读取所需数据块，最后调用解码 API 对获得的数据块进行解码操作即可修复丢失的数据.

每个数据节点都被分割为多个微型节点，微型节点中的健康状态表明了当前微型节点中数据的健康状态. 在系统发现错误时，会先检查数据节点中所有微型节点的健康状态，对健康状态正常的微型节点会忽略，对不健康的微型节点进行修复. 这样就避免了全节点修复.

当微型节点中健康状态无法确认其正确性时，会默认为微型节点中数据已经丢失，视情况而定选用全节点修复的策略.

通过微型节点中健康状态这一项实现微修复技术，系统可以大大减少修复时间，降低修复消耗，提升系统可用性.

当系统判定数据丢失不可微修复时，将会触发

全节点修复流程. 在DHT协议的设计中采用了同一编码集合中的微型节点必定在同一物理节点上有且仅有一个. 根据用户的设置,系统可以容忍两个或更多物理节点完全失效的情况. 在触发全节点修复流程后,系统会调查所有编码集合中对应失效节点的微型节点,再对所有被波及的编码集合进行解码操作,修复失效微型节点中的数据. 所有微型节点都被恢复后写入到数据节点中,系统修复完毕.

图7是一个简单的数据修复的例子,图中微型节点0发生了损坏,微型节点0中的数据丢失. 此时系统从损坏文件所属的编码集合中获得微型节点1～6的数据,将其放入缓冲区中进行解码操作,重新计算或生成微型节点0中的数据,并将其写入原微型节点0中,至此数据修复操作完成.

节点的修复时间为

$$\text{time}_{\text{node}} = \text{time}_{\text{decode}} * \text{Num} \tag{6}$$

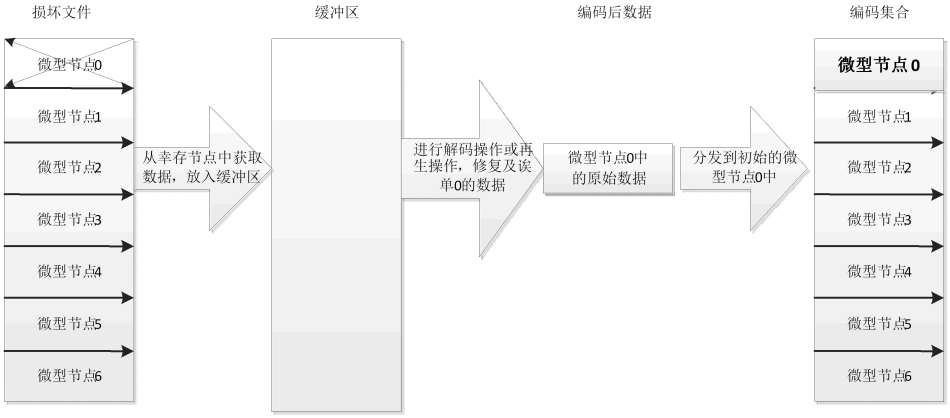

图7 节点失效时的恢复流程

Fig. 7 Restore process when node failure

5 BDCode实验评价

5.1 实验平台

测试工作采用的硬件平台是Intel(R) Xeon(R) CPU E5606 @ 2.13GHz,内存16GB DDR3,测试方式采用填充数据和修复数据修复的方式. 整个系统测试为仿真测试,数据的编解码过程都在仿真模拟实现,节点的测试过程也是仿真模拟实现.

5.2 数据块的编解码时间

名字节点在用户写入数据后,会对用户数据进行编码,然后写入到选择的微型节点中. 如果数据丢失,则通过解码方法修复. 在编解码过程中,范德蒙码和柯西RS码,这两种编码由于其容错率和分散性适合用于分布式系统,所以测试过程中采用了这两种编码和BDCode编码进行对比测试.

用户的数据是随机生成的单位大小的数据块. 在确认选用的编码种类后,将数据分块送入缓冲区. 全部数据块进入缓冲区后,系统开始对数据进行编码. 从三个方面测试了编码的性能:数据块大小,编码集合中数据块的个数,编码集合中校验快个数.

图8表示当数据块为5,校验块为2时的一个编码集合的编解码时间的编码规律图. 图中其他参数不变,数据块大小从1M变化到10M. 由图8可知,随着数据块大小的增加,编码时间也随之增加,解码时间随着数据块的大小增加而增加,但相较编码时间,数据块的单块解码时间效率显然比较高. 同时,BDCode和CRS由于采用bitmap将乘法操作转为异或操作,所以解码时间都较RS短,而BDCode由于经过编解码优化,解码时间又明显比CRS短,因此BDCode更适合用于大数据备份系统的解码工作. 同时由图8还可以看出,解码时间要少于编码时间,因此对系统解码时间的优化更能够提供整个系统的编码效率.

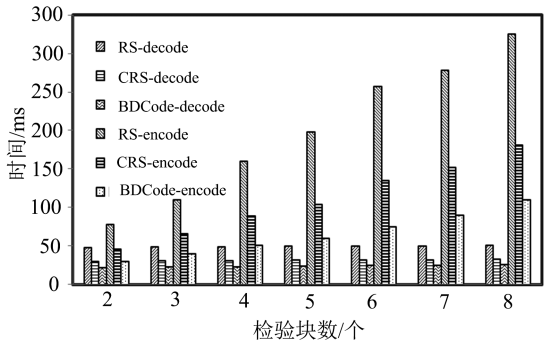

图8 数据块大小对编解码时间的影响

Fig. 8 Effect of data block size on encoding and decoding time

图 9 表示数据块大小为 1M，编码集合内数据块个数为 5，校验块个数从 2 到 10 编码时的编解码时间. 从图 9 可以看出，随着校验块个数的增加，编码时间也急剧上升. 由于 RS、CRS 和 BDCode 满足 MDS 性质，越多的校验块代表着越高的容错能力. 同时可以看出 RS 码的增长速率高于 CRS 和 BDCode. 这是由于 RS 码的编码矩阵相对于 CRS 和 BDCode 并未进行优化，所以每生成一个校验块，RS 的编码时间增加量会比 CRS 和 BDCode 更多；而 RS 、CRS 和 BDCode 的解码时间并未显著变化，时间基本持平，这说明校验块的增加并不能引起单数据块解码时间的增加. 在编解码的对比中，BDCode 的编解码性能都比 CRS 和 RS 好，说明 BDCode 更适合对编解码时间要求高的系统.

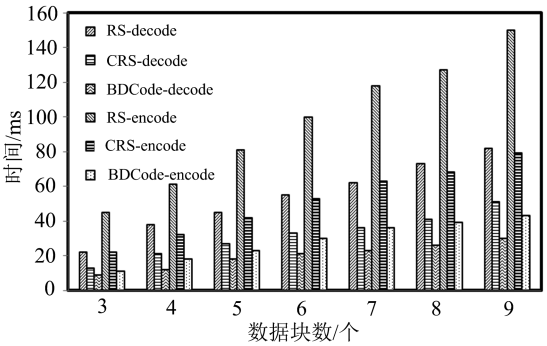

图 9　校验块个数对编解码时间的影响

Fig. 9　Effect of number of check blocks on encoding and decoding time

图 10 表示校验块为 2，数据块大小为 4M，编码集合内数据块从 3 变化到 10 的编码时间. 从图 10 可知，编码组内随着数据块个数的增加编码时间也随之加长，从数十秒到数百秒，同时比较 RS、CRS 和 BDCode 三种编码方式，BDCode 整体编码效果都比 RS 和 CRS 编码时间短. 从图 10 还可以看出，数据块的增加对解码时间的影响并没有对编码时间的影响大. 这是因为在解码时要求的数据量受数据块数量的影响较小，因此数据块数量对于编码性能有着较为明显的直接影响，对于解码性能虽然影响不大，但也不可忽略. 同时 BDCode 码对于编码时的解码时间基本保持稳定，且整体解码效率要高于 RS 和 CRS 码.

图 8～10 显示了数据块数变化，校验块数变化，数据块大小，编码集合内数据块变化的编解码时间的不同结果. 通过对比测试可以看出，在整个系统的编码过程中，编码时间随着数据块数，校验块数，数据块大小这些参数变化而变化. 数据块的恢复时间同数据块大小关系最密切，数据块大小越大解码时间越长；编码集合内数据块个数增长时，解码恢复时间缓慢增长，但涨幅没有数据块大小变化时大；与编码集合内校验块个数关系不大. 同时解码时间较编码时间波动小. 此外 BDCode 码的整体编解码效率比 CRS 和 RS 码高，更适合用于分布式系统中. 由图 8～10 得知，BDCode 比 CRS 编码效率平均高 34.2%，解码效率平均高 18.1%；比 RS 码编码效率平均高 56.5%，解码效率平均高 31.1%.

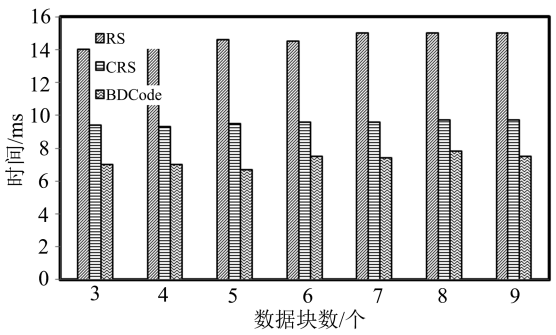

图 10　数据块个数对编解码时间的影响

Fig. 10　Effect of number of data blocks on encoding and decoding time

5.3　节点的编解码时间

数据块经过编码后存储到数据节点上，如果数据块出现损坏则调用解码算法进行解码，对节点的编解码测试分为对大量数据的编码时间的测试和虚拟节点、物理节点失效时的测试.

图 11 为数据块大小为 1M，个数为 600 时编码时间图，图中编码集合中数据块数从 3 变化到 10，测试中生成随机数据的时间均算在编码时间内. 由图 11 可以看出，两种编码的编码时间并未随着 k 的变化大幅度波动. 这是由于编码集合内数据块数多则编码组数少，虽然单编码组编码时间增加，总编码组数却减少，整体效果达到综合，编码时间并未显著增加. 同时在图中可以看出，BDCode 的编码效率比 RS 和 CRS 好.

5.4　系统读性能测试

首先测试系统的正常读性能. 实验使用 TPC-C 基准测试与 1GB 的工作负载来测试系统正常的读性能，使用的纠删码分别是 RS、CRS 和 BDCode，并与三副本时的系统读性能相比较，使用的是四种不同大小的查询结果集，测试结果从如图 12 所示. 从图 12 可以看出，使用纠删码的系统在一定程度上优于三副本. 产生这种现象的原因是，纠删码中只保持

原始数据的一个副本，整个系统的读性能的优越得益于轻负载.

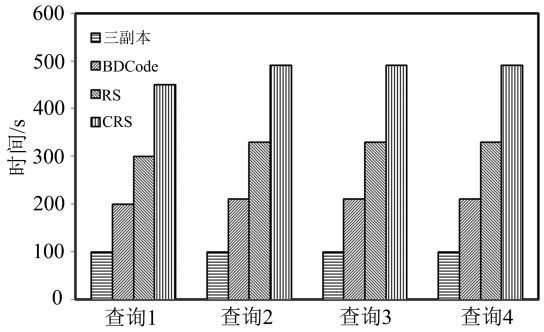

图 11 数据块为 1M，个数为 600 个时编码时间图

Fig. 11 Encoding and decoding time at 600 data blocks, each one equal 1 M

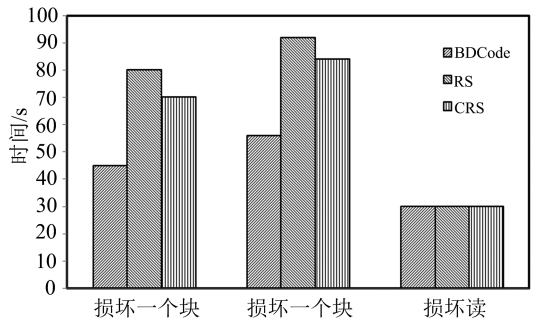

图 12 系统正常读性能测试图

Fig. 12 System normal read performance test

接下来测试系统的降级读性能. 我们使用一个256MB大小的文件作为轻负荷来测试系统的降级读性能，测试结果如图 13 所示. 前三根柱形条代表读一个损坏的块时，系统有且仅有一个块损坏的时候，采用不同纠删码的系统读性能；中间三根柱形条表示读一个损坏的块时，系统此时有两个块损坏；最后三根柱形条显示的时间读一个恢复块. 从测试结果可以看出，系统使用 BDCode 纠删码在降级读的时候，比使用 RS 和 CRS 时，读性能分别高出32.2%和 21.4%，并且在恢复数据后，系统读性能基本相同，因此 BDCode 更加适合大数据存储系统.

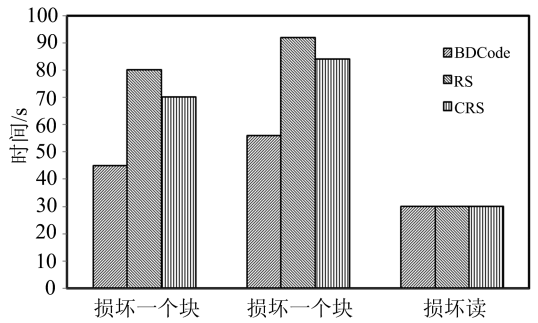

图 13 系统降级读性能测试图

Fig. 13 System demoted read performance test

6 结论

本文分析研究了现有大数据存储系统的状况，结合纠删码技术来提高系统存储利用率. 提出了一种基于编码的分布式备份机制，该机制将编码技术应用于包含大量冷数据的分布式系统中，对分布式系统进行数据保护和负载均衡. 对 CRS 编码进行了改进，提出了 BDCode 编码，利用其编解码速度快的优势来提高 BDCode. 对系统的整体架构和软件模块进行了分析设计，采用并行恢复的方式减少系统修复时间，采用虚拟节点提高系统的负载均衡. 测试结果证明系统使用 BDCode 能更快地完成编解码工作.

参考文献(References)

[1] MORRIS R J T, TRUSKOWSKI B J. The evolution of storage systems[J]. IBM Systems Journal, 2003, 42(2): 205-217.

[2] WANG Y, KAPRITSOS M, REN Z, et al. Robustness in the Salus scalable block store[C]// Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation. Lombard, USA: ACM Press, 2013: 357-370.

[3] LI Y, DHOTRE N, OHARA Y, et al. Horus: Fine-grained encryption-based security for large-scale storage[C]// Proceedings of the 11th USENIX Conference on File and Storage Systems. San Jose, USA: IEEE Press, 2013: 147-160.

[4] TIWARI D, BOBOILA S, VAZHKUDAI S S, et al. Active flash: Towards energy-efficient, in-situ data analytics on extreme-scale machines[C]// Proceedings of the 11th USENIX Conference on File and Storage Systems. San Jose, USA: IEEE Press, 2013: 119-132.

[5] WILKES J, GOLDING R, STAELIN C, et al. The HP AutoRAID hierarchical storage system[J]. ACM Transactions on Computer System, 1996, 14(1): 108-136.

[6] GHEMAWAT S, GOBIOFF H, LEUNG S T. File and storage systems: The Google file system[C]// Proceedings of the 9th ACM Symposium on Operating Systems Principles. Bolton Landing, USA: IEEE Press, 2003: 29-43.

[7] SHVACHKO K, KUANG H, RADIA S, et al. The Hadoop distributed file system[C]// Proceedings of

Symposium on Mass Storage Systems & Technologies. Incline Village, USA: IEEE Press, 2010: 1-10.

[8] Amazon simple storage service (S3)[EB/OL]. http://www.amazon.com/s3.

[9] WEIL S A, BRANDT S A, MILLER E L, et al. Ceph: A scalable, high-performance distributed file system[C]// Proceedings of the 7th Conference on Operating Systems Design and Implementation. Seattle, USA: ACM Press, 2006: 307-320.

[10] BLOMER J, KALFANE M, KARP R, et al. An XOR-based erasure-resilient coding scheme [R]. Technical Report TR-95-048, International Computer Science Institute, 1995.

[11] PLANK J S. A tutorial on Reed-Solomon coding for fault-tolerance in RAID-like systems[J]. Software-Practice & Experience, 1997, 27(9): 995-1012.

[12] STOICA I, MORRIS R, KARGER D, et al. Chord: A scalable peer-to-peer lookup service for Internet applications[J]. Proceedings of the ACM SIGCOMM Computer Communication Review, 2001, 31(4): 149-160.

[13] Meister D, Brinkmann A. dedupv1: Improving deduplication throughput using solid state drives (SSD)[C]// Proceedings of the 26th Symposium on Mass Storage systems and Technology. Incline Village, NV: IEEE, 2010: 1-6.

[14] XIA M Y, SAXENA M, BLAUM M et al. A tale of two erasure codes in HDFS[C]// Proceedings of the 13th USENIX Conference on File and Storage Technologies. Santa Clara, USA: ACM Press, 2015: 213-226.

[15] FAN B, TANTISIRIROJ W, XIAO L, et al. Diskreduce: RAID for data-intensive scalable computing[C]// Proceedings of the 4th Annual Workshop on Petatascale Data Storage. Portlang, USA: ACM Press, 2009: 6-10.

[16] VENKATARAMANI V, AMSDEN Z, BRONSON N, Get al. Tao: How Facebook serves the social graph [C]// Proceedings of the ACM SIGMOD International Conference on Management of Data. New York, USA: ACM Press, 2012: 791-792.

[17] NISHTALA R, FUGAL H, GRIMM S, et al. Scaling memcache at Facebook[C]// Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation. Berkeley, USA: ACM Press, 2013: 385-398.

[18] Google-GFS2 Colossus [EB/OL]. http://www.quora.com/Colossus-Google- GFS2" Google, 2012.

[19] TAMO I, BARG A. A family of optimal locally recoverable codes [J]. IEEE Transactions on Information Theory , 2014, 60(8): 4661-4676.

[20] FENG G L, DENG R H, BAO F, et al. New efficient MDS array codes for RAID part I: Reed-Solomon-like codes for tolerating three disk failures[J]. IEEE Transactions on Computers, 2005, 54(9):1071-1080.

[21] HAFNER J L. WEAVER Codes: Highly fault tolerant erasure codes for storage systems[C]// Proceedings of the 4th USENIX Conference on File and Storage Technology. San Francisco, USA: ACM Press, 2005: 211-224.

[22] HAFNER J L. HoVer erasure codes for disk arrays [C]// International Conference on Dependable Systems and Networks. Philadelphia, USA: IEEE Press, 2006: 217-226.

[23] XU L H, BRUCK J. X-Code: MDS array codes with optimal encoding [J]. IEEE Transactions on Information Theory, 1999, 45(1): 272-276.

[24] YIN C, XIE C S, WAN J G, et al. BMCloud: Minimizing repair bandwidth and maintenance cost in cloud storage [J]. Mathematical Problems in Engineering, 2013, 2013(6): 1-11.

[25] RASHMI K V, NAKKIRAN P, WANG J Y, et al. Having your cake and eating it too: Jointly optimal erasure codes for I/O, storage, and network-bandwidth[C]// Proceedings of the 13th USENIX Conference on File and Storage Technologies. Santa Clara, USA: ACM Press, 2015: 81-94.

[26] STRZELCZAK P, ADAMCZYK E, HERMAN-IZYCKA U, et al. Concurrent Deletion in a Distributed Content-Addressable Storage System with Global Deduplication[C]// Proceedings of the 11th USENIX Conference on File and Storage Technologies. 2013: 161-174.

[27] FU M, FENG D, HUA Y, et al. Design Tradeoffs for data deduplication performance in backup workloads [C]// Proceedings of the 13th USENIX Conference on File and Storage Technologies. Santa Clara, USA: ACM Press, 2015: 331-344.

[28] REED I S, SOLOMON G. Polynomial codes over certain finite fields[J]. Journal of the Society for Industrial and Applied Mathematics, 1960, 8(2): 300-304.