计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2016.20150558
53(9):1906-1917, 2016

一种基于磁盘内和磁盘间冗余的混合编码方案

卞建超　查雅行　罗守山　李　伟
(北京邮电大学计算机学院　北京　100876)
(灾备技术国家工程实验室(北京邮电大学)　北京　100876)
(bianjianchao@bupt.edu.cn)

A Hybrid Coding Scheme Based on Intra- and Inter-Device Redundancy

Bian Jianchao, Zha Yaxing, Luo Shoushan, and Li Wei
(*School of Computer Science, Beijing University of Posts and Telecommunications, Beijing* 100876)
(*National Engineering Laboratory for Disaster Backup and Recovery (Beijing University of Posts and Telecommunications), Beijing* 100876)

Abstract The development and application of cloud computing set higher requirement for the fault-tolerant capability of the storage systems. Erasure code has been widely used to generate device-level redundancy to protect against device failures, while has less space efficiency when resisting the sector failures. Current optimization schemes for the sector failures only resist the failures of small amounts of the sectors or specific sectors. In this paper, we propose a hybrid coding scheme (intra- and inter-device redundancy, IIDR) combining inter-device redundancy with intra-device redundancy based on the homomorphism property of MDS (maximum distance separable) codes, which employs global parity sector against sector failures in the data disks when adding parity device against device failures, and optimizes the ability to process single-sector errors taking advantage of intra-device coding to generate local parity sectors. In the end, the correctness proof and performance analysis are shown in this paper, and the results indicate that our scheme can protect against device failures and sector failures of any distribution, and the computing cost of recovering single-sector errors is much lower, and the update performance is better. Compared with traditional intra-device coding schemes, our scheme comes with less space usage.

Key words cloud computing; device failures; latent sector errors (LSEs); erasure code; intra-disk redundancy

摘　要　云计算的发展和应用对存储系统的容错能力提出了更高要求.纠删码被广泛运用于计算磁盘级冗余以抵抗磁盘错误,但抵抗扇区错误时磁盘利用率较低,现有针对扇区错误的优化方案只能抵抗较小数目或者特定分布的扇区错误.为此,利用MDS(maximum distance separable)码的同态性质,提出了一种将磁盘间冗余与磁盘内冗余相结合的混合编码(intra- and inter-device redundancy, IIDR).添加校验磁盘抵抗磁盘错误的同时,在数据磁盘中添加全局校验扇区以抵抗扇区错误,利用磁盘内编码添加本地校验扇区,以优化处理单个扇区错误的性能.最后,正确性证明及性能分析的结果表明所提方案能抵抗磁盘错误和任意分布的扇区错误,并且恢复单个扇区错误计算开销低,更新性能较好,同时与传统的磁盘内编码方案相比,有较高的磁盘利用率.

收稿日期:2015-06-16;**修回日期**:2015-10-13
基金项目:国家"八六三"高技术研究发展计划基金项目(2015AA016005)
This work was supported by the National High Technology Research and Development Program of China (863 Program) (2015AA016005).

关键词 云计算;磁盘错误;潜在扇区错误;纠删码;磁盘内冗余

中图法分类号 TP302.8

随着云计算相关技术的发展,各大厂商也陆续推出了商用的云服务,用户量的激增、业务模式的扩展、服务保障要求的提高,都对数据的可用性提出了更高要求,现有数据容错机制的改进加强迫在眉睫.造成数据丢失的设备故障可以分为磁盘错误(device failures)和扇区错误(sector failures)[1],磁盘错误会导致出错磁盘上的数据全部丢失,扇区错误则会导致单个扇区上的数据不可用.造成设备故障的因素有很多,介质材料不合格、松动微粒引起的介质划痕、高飞写入(high fly writes)造成的异常位模式、振动、磁头触及介质、偏离磁道的读写都有可能造成设备故障[2].随着使用垂直记录技术的几千兆级磁盘打入市场,其更高的磁录密度、更窄的磁道宽度、更贴近磁盘的浮动磁头,也更容易被软颗粒污染物划伤[2].此外,随着固态硬盘(solid state drive, SSD)生产成本降低,大规模应用成为可能,但与机械硬盘(hard disk drive, HDD)不同的是,SSD 的单个扇区错误发生的概率是随着写操作的次数增长而增大的[3].

相对于设备错误,潜在扇区错误(latent sector errors, LSEs)在读取该扇区的数据之前无法检测,这种类型故障对数据的可用性造成了很大风险.当设备错误出现后,需要使用独立冗余磁盘阵列(redundant arrays of independent disks, RAID)重建数据,但此时有磁盘出现 LSEs 就有可能造成数据重建失败[4].

1 相关工作

针对扇区错误开展的研究有很多,目前主要有2种解决方案:1)scrubbing 方案[5],该方案是周期性地在后台对各磁盘进行扫描,发现 LSEs 后使用 RAID 冗余对其进行修复,该方案已经在 NetApp 的系统上部署商用了;2)IDR(intra-disk redundancy)方案,该方案在每个磁盘内添加校验冗余,并配合具有磁盘间冗余的 RAID 一起应用.一些文件系统[6]在磁盘中对部分元数据进行备份,IRON(internal robustness)文件系统[7]对每个文件添加校验单元.Dholakia 等人[8]提出了对1个磁盘中所有数据块添加校验冗余的方案,其实就是纠删码在单个磁盘内的应用,但以上方案都需要与磁盘间容错的 RAID 搭配应用.Iliadis 等人[9]对比分析了 scrubbing 方案与 IDR 方案,指出 scrubbing 方案在数据量极为庞大的今天,周期地扫描每块磁盘的做法显然效率低下.然而,IDR 方案在每个磁盘添加冗余的方法,其磁盘利用率也很低.特别是在同时出现磁盘错误和 LSEs 的混合错误模式下,以上方案都无法独立解决,但传统的纠删码为了容忍额外的扇区错误需要添加整块的冗余磁盘,容错效率同样低下.

针对混合错误模式,2013 年 Plank 等人[10-11]提出了 SD Codes,该方案在校验磁盘的基础上添加全局校验扇区对抗扇区错误,如在添加 1 块校验磁盘和 2 个全局校验扇区的情况下,SD Codes 是介于 RAID5 与 RAID6 之间的解决方案.相对于传统纠删码,该方案能够容忍 1 个磁盘错误、2 个扇区错误,而不用添加第 2 块冗余校验磁盘.相对于 IDR 方案在每块磁盘上添加同等数量的校验扇区,该方案在磁盘利用率上有很大优势.但该方案最多只能容忍每个条带 3 个扇区错误,而且还有其他参数限制.

2014 年 Li 等人[1,12]提出了 STAIR Codes 方案,该方案提出了一种更一般化的应对混合错误模式的编码方法,使用向量 $\boldsymbol{e}$ 表示扇区错误分布,在数据磁盘上添加相应分布的全局校验扇区以抵抗扇区错误.该方案需要 2 种 MDS(maximum distance separable)码分别承担行编码和列编码,相比于 SD Codes,该方案能够容忍更多扇区错误,并能灵活套用任意系统 MDS 码,相比于传统的 IDR 方案,STAIR Codes 有更高的磁盘利用率.但 STAIR Codes 需要向量来表示错误分布情况,即 STAIR Codes 只能容忍特定分布的 s 个扇区错误,在实际部署时有一定局限性.Schroeder 等人[4]的扇区错误分析指出,不同类型的磁盘 90%~98%的扇区错误是单独出现,即存储系统面临的风险绝大多数来自孤立的扇区错误.但在处理单个扇区错误时,STAIR Codes 仍需读取多个磁盘以恢复数据,I/O 及计算开销比传统 IDR 方案大.

针对现有方案的缺陷,本文提出了一种基于磁盘内和磁盘间冗余的混合编码算法(intra- and inter-device redundancy, IIDR)方案.本方案结合了磁盘内编码在修复小规模扇区错误时 I/O、计算开销上的优势,又解决了其对抗大规模扇区错误时磁盘利用率急剧下降的缺点.相对于 STAIR Codes 方案,

本方案牺牲了一定的磁盘利用率,解决了其不能抵抗任意扇区错误的问题,并优化了在应对小规模特别是单个扇区错误时的I/O及计算性能.

2 混合编码方案

本节首先给出预备知识及基本定义,然后分别介绍本方案编码及数据恢复过程.

2.1 预备知识

一个由 n 块磁盘组成的存储系统,包括一定数量的数据磁盘和校验磁盘,其中校验磁盘所存数据由数据磁盘编码计算而来,与同一次编码计算相关的所有数据称为条带(stripe),存储系统也可以看作为多个条带的组合.同一磁盘上同属一个条带的数据集合称为条块(strip/chunk),条块又由若干扇区(sector/element/symbol/block)组成,个数记为 r,扇区为编码基本计算单元[13].如图1所示,条块由 r 个扇区组成,条带由 n 个条块组成,条带可以看成由扇区组成的 $r\times n$ 矩阵.

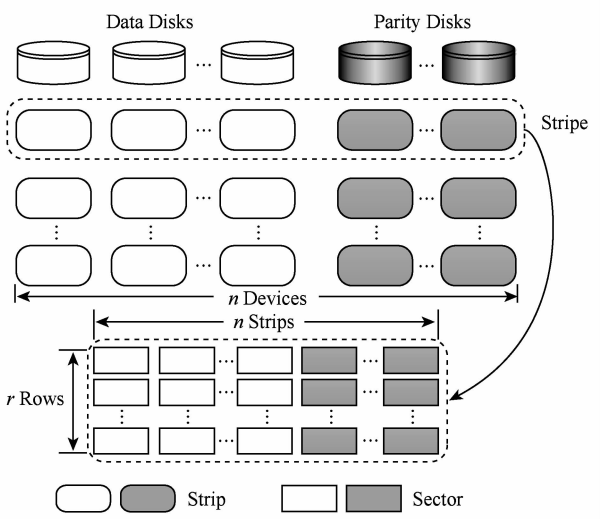


Fig. 1 Relationship between sector, strip and stripe.

图1 扇区、条块、条带三者关系

如上所述,各条带在编码时相互独立,可以以条带为单位来进行讨论.存储系统面临的威胁有磁盘错误和扇区错误,可以把磁盘错误映射为条带中的条块错误,即整个条块数据丢失或损坏.本文提出的方案能抵抗条带中同时出现 $m(m<n)$ 个条块错误(即一个存储系统出现 m 个磁盘错误)和 s 个扇区错误,为此需添加 m 个校验条块,在剩余 $n-m$ 个数据条块中添加相应的校验扇区,数量记为 s',分布在 m' 个条块中.

2.1.1 校验扇区分布

本方案的目标是能抵抗 s 个扇区错误,需添加相同分布的校验扇区(证明见第3节),而 s 个扇区错误的分布有很多种可能,所以添加的校验扇区应覆盖其所有可能分布.如 $s=4$ 时,可能的分布就有(4),(3,1),(2,2),(1,1,1,1),添加的校验扇区分布应为(4,2,1,1).若要抵抗 s 个扇区错误,需添加的校验扇区的分布为

$$(s,\lfloor s/2\rfloor,\lfloor s/3\rfloor,\cdots,\lfloor s/s-1\rfloor,\lfloor s/s\rfloor)$$

(证明见第3节).为了优化单个扇区错误恢复性能,本方案在每个条块中添加一个本地校验扇区,所以本方案校验扇区分布为

$$\underbrace{(s,\lfloor s/2\rfloor,\lfloor s/3\rfloor,\cdots,\lfloor s/s-1\rfloor,\lfloor s/s\rfloor,1,\cdots,1)}_{n},$$

则全局校验扇区分布实际为

$$(s-1,\lfloor s/2\rfloor-1,\lfloor s/3\rfloor-1,\cdots,\lfloor s/m'\rfloor-1),$$

其中 $m'=\lfloor s/2\rfloor$,s' 与 s 的关系就为

$$s'=(s-1)+(\lfloor s/2\rfloor-1)+(\lfloor s/3\rfloor-1)+\cdots+(\lfloor s/m'\rfloor-1),$$

本方案总计校验扇区个数为 $s'+n$.

2.1.2 基本定义

本编码可以用参数 (n,r,m,s) 来表示能够抵抗 m 个磁盘错误和 s 个扇区错误同时出现的情况,条带中添加了 m 个校验条块,数据条块中添加了分布为 $\underbrace{(s,\lfloor s/2\rfloor,\lfloor s/3\rfloor,\cdots,\lfloor s/s-1\rfloor,\lfloor s/s\rfloor,1,\cdots,1)}_{n}$ 的校验扇区.根据算法要求,本方案计算过程中需添加相同分布的外部全局校验扇区,每行添加 m' 个临时扇区、每列添加 $s-1$ 个临时扇区,即将条带由 $r\times n$ 扩展为 $(r+s-1)\times(n+m')$,称为计算条带.

设 $D_{i,j}$,$P_{i,k}$,$D'_{t,j}$,$P'_{t,l}$,P^*_h,$\hat{G}_{t,l}$,$G_{t,l}$ 分别表示数据扇区、行校验扇区、临时列数据扇区、临时列校验扇区、本地校验扇区、全局校验扇区、外部全局校验扇区,其中 $0\leqslant i\leqslant r-2$,$0\leqslant j\leqslant n-m-1$,$0\leqslant k\leqslant m-1$,$0\leqslant l\leqslant m'-1$,$0\leqslant h\leqslant n-1$,$0\leqslant t\leqslant s-2$.

本方案涉及使用2种系统MDS码,一种为磁盘间编码(行编码),采用 $(n+m',n-m)$ 码作为行编码算法,记为 C_{row};另一种为磁盘内编码(列编码),采用 $(r+s-1,r-1)$ 码作为列编码算法,记为 C_{col}.所以将本方案命名为磁盘内和磁盘间冗余编码.

2.1.3 同态性质

设符号 $\overset{C}{\Rightarrow}$ 表示左边输入的数据利用算法 c 进行编码计算,并从右侧输出编码后的数据.

根据定义,各列进行编码计算有:

当 $0\leqslant j\leqslant n-m-1$ 时,

$$D_{0,j},D_{1,j},\cdots,D_{r-2,j}\overset{C_{\text{col}}}{\Rightarrow}P_j^*,D'_{0,j},D'_{1,j},\cdots,D'_{s-2,j};$$

当 $0\leqslant k\leqslant m-1$ 时,

$$P_{0,k},P_{1,k},\cdots,P_{r-2,k},\overset{C_{\text{col}}}{\Rightarrow}P^*_{k+n-m},P'_{0,k},P'_{1,k},\cdots,P'_{s-2,k}.$$

将编码产生的临时数据扇区 $D'_{t,j}$ 各行进行编码计算:

当 $0\leqslant t\leqslant s-2$ 时,

$$D'_{t,0},D'_{t,1},D'_{t,2},\cdots,D'_{t,n-m-1}\overset{C_{\text{row}}}{\Rightarrow}P'_{t,0},P'_{t,1},\cdots,P'_{t,m-1},$$

则由列编码计算的 $P'_{t,l}$ 应等于行编码计算的 $P'_{t,l}$,称为同态性质,该性质也是本方案正确性的基础,文献[1]中给出了具体证明.

2.2 编码过程

2.2.1 示例

首先通过实例来介绍本方案编码过程,图 2 中取值 $n=8,m=2,r=6,s=4$,即有 6 个数据条块、2 个校验条块,每个条带添加分布为(3,1)的全局校验扇区,此外每个条块添加 1 个扇区作为本地校验扇区.行编码、列编码分别需采用(10,6)码、(9,5)码.具体步骤如表 1 所述,本方案编码思想是利用 MDS 码的同态性质,配合置 0 的外部全局校验扇区完成编码计算.

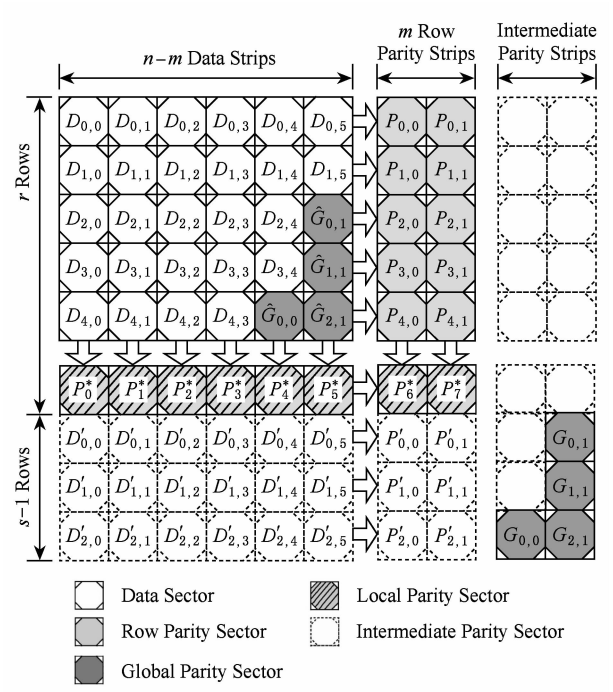


Fig. 2 Encoding algorithm.

图 2 编码算法

2.2.2 一般情况

根据定义以计算条带为单位,每行需要计算行校验扇区,每列需要计算本地校验扇区,此外还需要计算分布为($s-1$, $\lfloor s/2\rfloor-1$, $\lfloor s/3\rfloor-1$,…, $\lfloor s/m'\rfloor-1$)的全局校验扇区.

Table 1 Detailed Steps for Encoding

表 1 编码步骤

Step	Detailed Descriptions
1	$D_{0,0},D_{0,1},D_{0,2},D_{0,3},D_{0,4},D_{0,5}\Rightarrow P_{0,0},P_{0,1}$
2	$D_{1,0},D_{1,1},D_{1,2},D_{1,3},D_{1,4},D_{1,5}\Rightarrow P_{1,0},P_{1,1}$
3	$D_{0,0},D_{1,0},D_{2,0},D_{3,0},D_{4,0}\Rightarrow P_0^*,D'_{0,0},D'_{1,0},D'_{2,0}$
4	$D_{0,1},D_{1,1},D_{2,1},D_{3,1},D_{4,1}\Rightarrow P_1^*,D'_{0,1},D'_{1,1},D'_{2,1}$
5	$D_{0,2},D_{1,2},D_{2,2},D_{3,2},D_{4,2}\Rightarrow P_2^*,D'_{0,2},D'_{1,2},D'_{2,2},$
6	$D_{0,3},D_{1,3},D_{2,3},D_{3,3},D_{4,3}\Rightarrow P_3^*,D'_{0,3},D'_{1,3},D'_{2,3}$
7	$D'_{2,0},D'_{2,1},D'_{2,2},D'_{2,3},G_{0,0},G_{2,1}\Rightarrow D'_{2,4},D'_{2,5},P'_{2,0},P'_{2,1}$
8	$D_{0,4},D_{1,4},D_{2,4},D_{3,4},D'_{2,4}\Rightarrow\hat{G}_{0,0},P_4^*,D'_{0,4},D'_{1,4}$
9	$D'_{0,0},D'_{0,1},D'_{0,2},D'_{0,3},D'_{0,4},G_{0,1}\Rightarrow D'_{0,5},P'_{0,0},P'_{0,1}$
10	$D'_{1,0},D'_{1,1},D'_{1,2},D'_{1,3},D'_{1,4},G_{1,1}\Rightarrow D'_{1,5},P'_{1,0},P'_{1,1}$
11	$D_{0,5},D_{1,5},D'_{0,5},D'_{1,5},D'_{2,5}\Rightarrow\hat{G}_{0,1},\hat{G}_{1,1},\hat{G}_{2,1},P_5^*$
12	$P_{0,0},P_{1,0},P'_{0,0},P'_{1,0},P'_{2,0}\Rightarrow P_{2,0},P_{3,0},P_{4,0},P_6^*$
13	$P_{0,1},P_{1,1},P'_{0,1},P'_{1,1},P'_{2,1}\Rightarrow P_{2,1},P_{3,1},P_{4,1},P_7^*$

根据全局校验扇区的分布,本方案编码由以下 4 步组成:

1) 计算 0 至 $n-m-m'-1$ 列各校验扇区

根据定义,计算条带中 0 至 $r-s-1$ 行、0 至 $n-m-m'-1$ 列不包含全局校验扇区.此类行和列可以直接进行编码计算:

① 计算行校验扇区(不包含全局校验扇区的行),

$$D_{i,0},D_{i,1},\cdots,D_{i,n-m-1}\overset{C_{\text{row}}}{\Rightarrow}P_{i,0},P_{i,1},\cdots,P_{i,m-1},$$

其中 $0\leqslant i\leqslant r-s-1$;

② 计算本地校验扇区及相应的临时列数据扇区(不包含全局校验扇区的列),

$$D_{0,j},D_{1,j},\cdots,D_{r-2,j}\overset{C_{\text{col}}}{\Rightarrow}P_j^*,D'_{0,j},D'_{1,j},\cdots,D'_{s-2,j},$$

其中 $0\leqslant j\leqslant n-m-m'-1$.

2) 计算 $n-m-m'$ 至 $n-m-2$ 列各校验扇区

令外部全局校验扇区 $G_{t,l}=0$,其中 $0\leqslant t\leqslant s-2,0\leqslant l\leqslant m'-1$.经过上一轮运算,计算条带第 $r+s-2$ 行(最后 1 行)包含 $n-m-m'$ 个已知扇区,结合外部全局校验扇区 $G_{s-2,m'-1}=0,G_{x,l}=0$,其中 $0\leqslant l\leqslant m'-1,x=\lfloor s/(m'-l)\rfloor-2$,该行已知扇区数为 $n-m$,可调用行编码算法 $C_{\text{row}}(n+m',n-m)$

计算该行剩余扇区.使得第 $n-m-m'$ 列(包含全局校验扇区 $\hat{G}_{0,0}$)存在 $r-1$ 个已知扇区,可调用列编码算法 $C_{\text{col}}(r+s-1,r-1)$ 计算 $\hat{G}_{0,0}$ 及该列其他校验扇区.依照以上方法,自下而上、自左至右依次调用行、列编码算法计算 $r+s-2$ 至 $r+s-m'-1$ 行,$n-m-m'$ 至 $n-m-2$ 列包含的各校验扇区,具体算法如算法 1 所示.

算法 1. 编码循环算法.

初始化:

R_r; /* 行中数据完好的扇区数目 */

R_c; /* 列中数据完好的扇区数目 */

Set $t=2,i=0,q=0,j=0,p=0$.

For $i=0$ to $\lfloor s/2 \rfloor-2$

$t=t+i$;

搜索第 $r+s-t$ 行;

记录 R_r;

If $R_r=n-m$ Then

对 $r+s-t$ 行进行编码;

Else break;

End If

For $j=q$ to $m'-2$

搜索第 $n-m-m'+i$ 列;

记录 R_c;

If $R_c=r-1$ Then

对 $n-m-m'+i$ 列进行编码;

$p=p+1$;

Else break;

End If

End For

$q=j+p$;

$p=0$;

End For

输出:从 $n-m-m'$ 到 $n-m-2$ 列.

3) 计算 $n-m-1$ 列各校验扇区

经过第 2 步运算,r 至 $r+s-m'-1$ 行都存在 $n-m$ 个已知扇区,可调用行编码算法进行计算,

$$D'_{t,0},D'_{t,1},\cdots,D'_{t,n-m-2},G_{t,m'-1}=0 \overset{C_{\text{row}}}{\Rightarrow} D'_{t,n-m-1},P'_{t,0},P'_{t,1},\cdots,P'_{t,m-1},$$

其中 $0\leqslant t\leqslant s-m'-1$;

使得全局校验扇区 $\hat{G}_{t,m'-1}$($0\leqslant t\leqslant s-2$)所在的列($n-m-1$ 列)存在 $r-1$ 个已知扇区,可以调用列编码,

$$D_{0,n-m-1},D_{1,n-m-1},\cdots,D_{r-s-1,n-m-1},D'_{0,n-m-1},D'_{1,n-m-1},\cdots,D'_{s-2,n-m-1} \overset{C_{\text{col}}}{\Rightarrow} \hat{G}_{0,m'-1},\hat{G}_{1,m'-1},\cdots,\hat{G}_{s-2,m'-1},P^*_{n-m-1};$$

全局校验扇区全部计算完毕.

4) 计算 $n-m$ 至 $n-1$ 列(磁盘校验条块)剩余扇区

$$P_{0,k},P_{1,k},\cdots,P_{r-s-1,k},P'_{0,k},P'_{1,k},\cdots,P'_{s-2,k} \Rightarrow P_{r-s,k},P_{r-s,k},\cdots,P_{r-2,k},P^*_{k+n-m},$$

其中 $0\leqslant k\leqslant m-1$.

至此,条带中的校验扇区全部计算完毕.

2.3 数据恢复过程

2.3.1 示例

图 3 为本方案数据恢复示例,其中取值 $n=8$,$m=2$,$r=6$,$s=4$,图 3 中黑色表示出错扇区,即出现 2 个条块错误及分布为(4,2,1,1,1,1)的扇区错误,即该参数下本方案能抵抗最严重错误情况,通过这个示例来阐述本方案数据恢复算法,具体操作如表 2 所示.

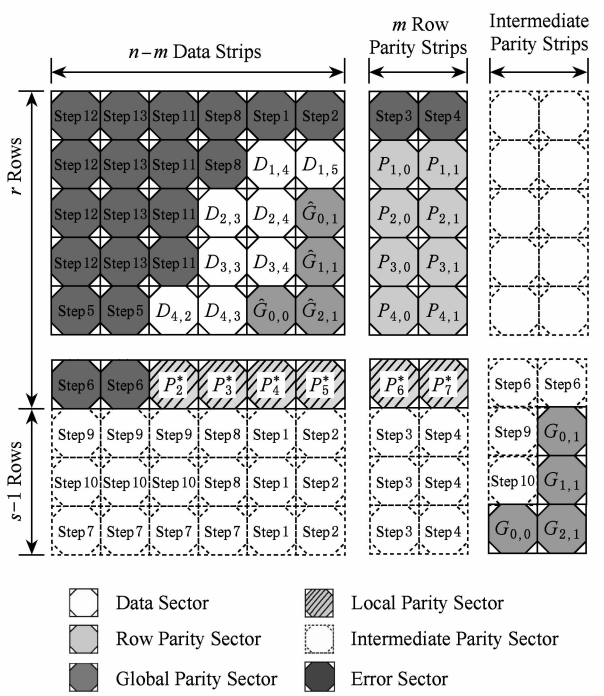


Fig. 3 Data recovery algorithm.

图 3 数据恢复算法

本方案数据恢复思路是先使用本地校验扇区修复单个扇区错误,再结合置 0 的外部全局校验扇区计算出部分临时行扇区,使得部分列达到列编码条件并恢复,依次类推直到数据完全恢复.

Table 2 Detailed Steps for Data Recovery

表 2 数据恢复步骤

Step	Detailed Descriptions
1	$D_{1,4},D_{2,4},D_{3,4},\hat{G}_{0,0},P_4^*\Rightarrow D_{3,2},D'_{0,4},D'_{1,4},D'_{2,4}$
2	$D_{1,5},\hat{G}_{0,1},\hat{G}_{1,1},\hat{G}_{2,1},P_5^*\Rightarrow D_{0,5},D'_{0,5},D'_{1,5},D'_{2,5}$
3	$P_{1,0},P_{2,0},P_{3,0},P_{4,0},P_6^*\Rightarrow P_{0,0},P'_{0,0},P'_{1,0},P'_{2,0}$
4	$P_{1,1},P_{2,1},P_{3,1},P_{4,1},P_7^*\Rightarrow P_{0,1},P'_{0,1},P'_{1,1},P'_{2,1}$
5	$D_{4,2},D_{4,3},\hat{G}_{0,0},\hat{G}_{2,1},P_{4,0},P_{4,1}\Rightarrow D_{4,0},D_{4,1}$
6	$P_2^*,P_3^*,P_4^*,P_5^*,P_6^*,P_7^*\Rightarrow P_0^*,P_1^*$
7	$D'_{2,4},D'_{2,5},P'_{2,0},P'_{2,1},G_{0,0},G_{2,1}\Rightarrow D'_{2,0},D'_{2,1},D'_{2,2},D'_{2,3}$
8	$D_{2,3},D_{3,3},D_{4,3},P_3^*,D'_{2,3}\Rightarrow D_{0,3},D_{1,3},D'_{0,3},D'_{1,3}$
9	$D'_{0,3},D'_{0,4},D'_{0,5},P'_{0,0},P'_{0,1},G_{0,1}\Rightarrow D'_{0,0},D'_{0,1},D'_{0,2}$
10	$D'_{1,3},D'_{1,4},D'_{1,5},P'_{1,0},P'_{1,1},G_{1,1}\Rightarrow D'_{1,0},D'_{1,1},D'_{1,2}$
11	$D_{4,2},P_2^*,D'_{0,2},D'_{1,2},D'_{2,2}\Rightarrow D_{0,2},D_{1,2},D_{2,2},D_{3,2}$
12	$D_{4,0},P_0^*,D'_{0,0},D'_{1,0},D'_{2,0}\Rightarrow D_{0,0},D_{1,0},D_{2,0},D_{3,0}$
13	$D_{4,1},P_1^*,D'_{0,1},D'_{1,1},D'_{2,1}\Rightarrow D_{0,1},D_{1,1},D_{2,1},D_{3,1}$

2.3.2 一般情况

本方案为了能抵抗 s 个任意分布的扇区错误,添加了大于 s 的校验扇区,实际容错上限为校验扇区相同分布的扇区错误(证明见第 3 节),即出现 m 个条块错误和分布为

$$\underbrace{(s,\lfloor s/2\rfloor,\lfloor s/3\rfloor,\cdots,\lfloor s/s-1\rfloor,\lfloor s/s\rfloor,1,\cdots,1)}_{n}$$

的扇区错误.

本方案只限定每列扇区错误数,错误扇区在该列的位置及扇区错误数排列顺序与恢复算法无关,为了便于算法描述,设定错误自上而下、自左至右顺序分布.假设发生最严重错误情况,即 0 至 $m-1$ 列数据全部丢失,m 至 $m+m'-1$ 列依次丢失 s,$\lfloor s/2\rfloor,\cdots,\lfloor s/m'\rfloor$ 个扇区数据,$m+m'$ 至 $n-1$ 各列均丢失 1 个扇区数据.

1) 修复简单错误

列内只存在单个扇区错误或者行内扇区错误数为 m 的情况,即恢复数据只涉及 1 次解码计算,称为简单错误.首先处理列内只存在单个扇区错误的情况,很显然此类列存在 $r-1$ 个完好扇区,可调用列编码算法 $C_{\text{col}}(r+s-1,r-1)$ 恢复数据,如表 2 的 Step1~4;接着处理扇区错误数为 m 的行,可知该行完好扇区个数为 $n-m$,可调用行编码算法 $C_{\text{row}}(n+m',n-m)$ 恢复数据,如表 2 中的 Step5~6.

2) 修复连续扇区错误

由编码算法可知外部全局校验扇区 $G_{t,l}=0$,其中 $0\leqslant t\leqslant s-2$,$0\leqslant l\leqslant m'-1$,已知外部校验扇区分布为 $(\lfloor s/m'\rfloor-1,\lfloor s/m'-1\rfloor-1,\cdots,\lfloor s/2\rfloor-1,s-1)$.

① 修复 $m+1$ 至 $m+m'-1$ 列

经过上一轮运算,计算条带第 $r+s-2$ 行包含 $n-m-m'$ 个已知扇区,结合外部全局校验扇区 $G_{s-2,m'-1}=G_{x,l}=0$,其中 $0\leqslant l\leqslant m'-1$,$x=\lfloor s/(m'-l)\rfloor-2$,该行已知扇区数为 $n-m$,可调用行编码算法 $C_{\text{row}}(n+m',n-m)$ 恢复该行剩余扇区.使得存在 2 个扇区错误的列($m+m'-1$ 列)包含 $r-1$ 个已知扇区,可调用列编码算法 $C_{\text{col}}(r+s-1,r-1)$ 恢复该列剩余扇区.依照以上方法,依次恢复 $m+m'-1$ 至 $m+1$ 列所有错误扇区,即恢复存在 2 至 $\lfloor s/2\rfloor$ 扇区错误的列.

② 修复第 m 列(s 个扇区错误)

此时,r 至 $r+s-m'-1$ 行都存在 $n-m$ 个已知扇区,可调用行编码算法进行计算,

$$D'_{t,0},D'_{t,1},\cdots,D'_{t,n-m-2},G_{t,m'-1}=0\overset{C_{\text{row}}}{\Rightarrow}D'_{t,n-m-1},P'_{t,0},P'_{t,1},\cdots,P'_{t,m-1},$$

其中 $0\leqslant t\leqslant s-m'-1$.

使得第 m 列拥有 $r-1$ 个已知扇区,可调用列编码算法 $C_{\text{col}}(r+s-1,r-1)$ 恢复该列剩余扇区,

$$D_{s,m},D_{s+1,m},\cdots,D_{r-2,m},P_m^*,D'_{0,m},D'_{1,m},\cdots,D'_{s-2,m}\overset{C_{\text{row}}}{\Rightarrow}D_{0,m},D_{1,m},\cdots,D_{s-1,m}.$$

3) 修复 0 至 $m-1$ 列(条块错误)

经过步骤 1,条块错误所在列仍有 s 个扇区错误,经过步骤 2 计算此类列存在 $r-1$ 个已知扇区,可调用列编码算法恢复所有扇区错误:

$$D_{s,j},D_{s+1,j},\cdots,D_{r-2,j},P_j^*,D'_{0,j},D'_{1,j},\cdots,D'_{s-2,j}\overset{C_{\text{row}}}{\Rightarrow}D_{0,j},D_{1,j},\cdots,D_{s-1,j},$$

其中 $0\leqslant j\leqslant m-1$.

至此,全部数据均被修复.

上述算法为本方案能够容忍的最严重错误情况,很显然针对一般错误情况(m 个磁盘错误,任意分布的 s 个扇区错误)本方案仍能胜任,具体正确性证明会在第 3 节进行阐述.

3 正确性证明

引理 1. s 个扇区错误所有分布的叠加为

$$(s,\lfloor s/2\rfloor,\lfloor s/3\rfloor,\cdots,\lfloor s/s-1\rfloor,\lfloor s/s\rfloor).$$

证明. 设 s 个扇区错误分布为 $(e_0,e_1,\cdots,e_{m'-1})$,其中 $1\leqslant m'\leqslant s$,且有 $e_0+e_1+\cdots+e_{m'-1}=s$,

为便于讨论设 $e_0 \geqslant e_1 \geqslant \cdots \geqslant e_{m'-1}$. 按照 m' 不同的取值来进行讨论验证:

1) $m'=1$, 即 s 个扇区错误分布在同一磁盘, $e_0=s$;

2) $m'=2$, 则分布为 (e_0, e_1), 可知 $e_0+e_1=s$ 且 $e_0 \geqslant e_1$, 则当 $e_0=e_1=s/2$ 时 e_1 取值最大, 由于 e_0, e_1 为整数且 $e_0 \geqslant e_1$, 则 e_1 的取值上限为 $\lfloor s/2 \rfloor$;

3) $m'=3$, 则分布为 (e_0, e_1, e_2), 可知 $e_0+e_1+e_2=s$ 且 $e_0 \geqslant e_1 \geqslant e_2$, 则当 $e_0=e_1=e_2=s/3$ 时 e_3 取值最大, 由于 e_0, e_1, e_2 为整数且 $e_0 \geqslant e_1 \geqslant e_2$, 则 e_2 的取值上限为 $\lfloor s/3 \rfloor$.

同理, 当 $m'=i$ 时, 其中 $2 \leqslant i \leqslant s$, e_{i-1} 的取值上限为 $\lfloor s/i \rfloor$, 则 s 个扇区错误所有分布的叠加为 $(s, \lfloor s/2 \rfloor, \lfloor s/3 \rfloor, \cdots, \lfloor s/s-1 \rfloor, \lfloor s/s \rfloor)$. 证毕.

引理 2. 本方案能抵抗与校验扇区相同分布的扇区错误.

证明. 本方案校验扇区包括本地校验扇区及全局校验扇区, 设全局校验扇区分布为 $(\lambda_0, \lambda_1, \cdots, \lambda_{m'-1})$, 其中 $\lambda_0 \leqslant \lambda_1 \leqslant \cdots \leqslant \lambda_{m'-1}$, 方案还会添加相同分布的外部全局校验扇区并置 0, 则本方案校验扇区分布为 $\underbrace{(1,1,\cdots,1,\lambda_0+1,\lambda_1+1,\cdots,\lambda_{m'-1}+1)}_{n}$, 使用行编码算法 $C_{\text{row}}(n+m', n-m)$, 列编码算法 $C_{\text{col}}(r+\lambda_{m'-1}, r-1)$ 来进行编解码操作.

根据命题, 此时存在同样分布的扇区错误, 其中单个扇区错误可以用本地校验扇区独立恢复, 剩余需要恢复分布为 $(\lambda_0+1, \lambda_1+1, \cdots, \lambda_{m'-1}+1)$ 的错误扇区. 可知此时计算条带中每行都存在 $n-m-m'$ 个可用扇区, 最后 λ_0 行都存在 m' 个外部全局校验扇区(置 0), 满足行编码条件, 调用 C_{row} 恢复最后 λ_0 行的剩余扇区(临时列校验扇区). λ_0+1 个错误扇区所在列存在 $r-1$ 个可用扇区($r-\lambda_0-1$ 个数据扇区、λ_0 个临时列校验扇区), 满足列编码条件, 调用 C_{col} 能够恢复该列 λ_0+1 个错误扇区.

此时, 剩余行存在 $n-m-m'+1$ 个可用扇区, 配合 $\lambda_1-\lambda_0$ 行都存在 $m'-1$ 个外部全局校验扇区, 满足行编码条件, 调用 C_{row} 恢复 $\lambda_1-\lambda_0$ 行所有扇区. 计算后 λ_1+1 个错误扇区所在列满足列编码条件, 能够恢复该列 λ_1+1 个错误扇区.

恢复完 λ_{i-1} $(1 \leqslant i \leqslant m'-1)$ 个错误扇区所在列后, 配合 $\lambda_i-\lambda_{i-1}$ 行都存在 $m'-i$ 个外部全局校验扇区, 可以调用 C_{row} 计算 $\lambda_i-\lambda_{i-1}$ 行所有扇区, 由此可以调用 C_{col} 恢复 λ_i+1 个扇区错误所在列的所有数据. 依次循环计算可以将全部错误扇区恢复.

因此, 本方案能抵抗与校验扇区相同分布的扇区错误. 证毕.

定理 1. 本方案若添加分布为 $\underbrace{(s, \lfloor s/2 \rfloor, \lfloor s/3 \rfloor, \cdots, \lfloor s/s-1 \rfloor, \lfloor s/s \rfloor, 1, \cdots, 1)}_{n}$ 的校验扇区, 则能抵抗 s 个任意分布的扇区错误.

证明. 根据引理 2, 若想抵抗 s 个扇区错误, 需添加相同分布的 s 个校验扇区. 根据引理 1 可知 s 个扇区错误所有分布的叠加为 $(s, \lfloor s/2 \rfloor, \lfloor s/3 \rfloor, \cdots, \lfloor s/s-1 \rfloor, \lfloor s/s \rfloor)$, 很显然分布本方案添加的校验扇区包含该值.

因此, 本方案若添加分布为 $\underbrace{(s, \lfloor s/2 \rfloor, \lfloor s/3 \rfloor, \cdots, \lfloor s/s-1 \rfloor, \lfloor s/s \rfloor, 1, \cdots, 1)}_{n}$ 的校验扇区, 则能抵抗 s 个任意分布的扇区错误. 证毕.

4 性能分析

如第 1 节所述, 本文提出了一种基本磁盘间和磁盘内冗余的混合编码算法(IIDR). 本节从磁盘利用率、计算开销、更新性能 3 个方面, 通过与经典的磁盘内冗余方案 IDR 及混合编码方案 SD Codes, STAIR Codes 做对比, 来分析本方案各方面性能的优劣.

4.1 磁盘利用率

随着数据爆炸性的增长, 磁盘利用率成为衡量编码算法优劣的重要指标. 由于 STAIR Codes, IDR 及本方案在同等容错能力下校验磁盘数目一致, 所以数据磁盘中校验扇区的多少决定了一个方案磁盘利用率的高低, 所需校验扇区越多则磁盘利用率越低. 图 4 中取值 $n=8$, $r=6$, 分别计算了抵抗 4 个扇区错误的 5 种分布情况下各方案单个条带的校验扇区数目.

如图 4 所示, 抵抗特定分布的扇区错误, STAIR Codes 只需添加对应分布的校验扇区, 即只需 4 个校验扇区, 该方案所需添加的校验扇区最少; 本方案除了添加全局校验扇区, 还在每个条块中都添加了本地校验扇区, 所以本方案校验扇区数略大于 STAIR Codes 方案; IDR 方案需要在每个条块添加最大错误数的本地校验扇区, 即校验扇区数目为最大错误数与 n 之积, 所以随着最大错误数的增大, IDR 方案变化十分明显. 可以看出在抵抗特定分布扇区错误时, 本方案磁盘利用率介于 STAIR Codes 与 IDR 方案之间.

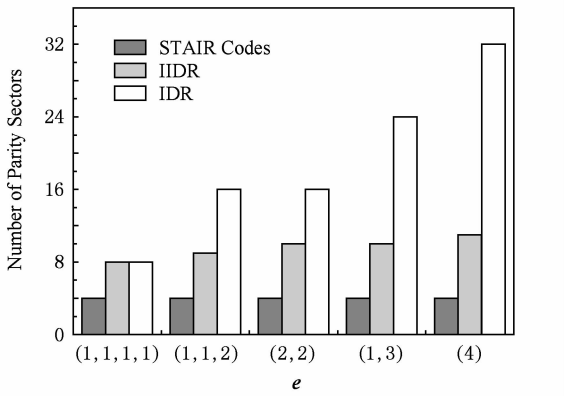


Fig. 4 The number of parity sectors for different e.

图 4 校验扇区数目

由于 STAIR Codes 方案不支持任意扇区错误分布,只能对比 IDR 方案与本方案在面对任意错误分布时的磁盘利用率.本方案添加的校验扇区分布为

$$\underbrace{(s,\lfloor s/2\rfloor,\lfloor s/3\rfloor,\cdots,\lfloor s/s-1\rfloor,\lfloor s/s\rfloor,1,\cdots,1)}_{n},$$

则数目可以表示为

$$s'_{\mathrm{IIDR}}=s+\lfloor s/2\rfloor+\lfloor s/3\rfloor+\cdots+\lfloor s/s-1\rfloor+\lfloor s/s\rfloor+n-s,$$

IDR 方案校验扇区则为 $s'_{\mathrm{IDR}}=n\times s$,可以看出,在 n 相同取值情况下,关系如下:

$$\begin{cases}s'_{\mathrm{IIDR}}=s'_{\mathrm{IDR}}, & s=1;\\ s'_{\mathrm{IIDR}}<s'_{\mathrm{IDR}}, & s>1.\end{cases}$$

图 5 中分别取值 $n=8,16,32$,分析对比 2 种方案在 s 不同的取值下校验扇区数量的变化.

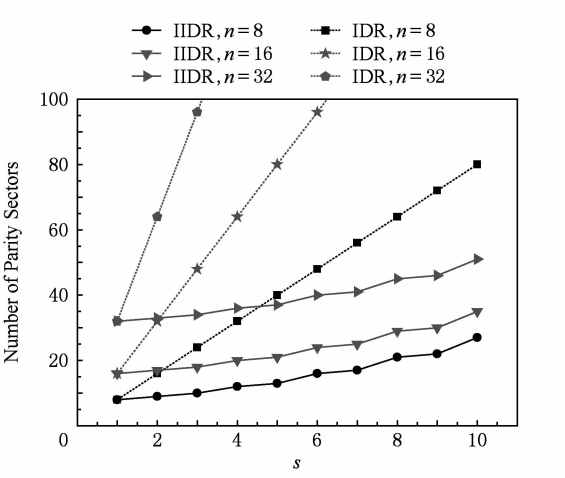


Fig. 5 The number of parity sectors for different s.

图 5 s 不同取值下校验扇区数目

如图 5 所示,同一 n 取值下,2 方案所需校验扇区数量与 s 成正比,IDR 方案所需校验扇区更多,且增长速率也大于本方案.由于 2 个方案都是在每个磁盘上添加校验扇区,在同一 s 取值下,2 方案所需校验扇区数量与 n 成正比,同样 IDR 方案所需校验扇区多于本方案,且增长速率大于本方案.综上所述,在抵抗任意错误分布的情况下,本方案的磁盘利用率优于 IDR 方案.

4.2 计算开销

为了量化分析编解码算法的计算开销,将算法分解成若干步基本操作 $Mult\_XORs$[14],以步骤的多少来衡量一个算法计算开销的大小 ,很显然分解的操作越少,则计算开销较小.基本操作 $Mult_XORs(R_1,R_2,\alpha)$定义为若干字节数据 R_1 与常量 α 相乘,所得的值与 R_2 异或求和.如:(n,k)码的编码过程可以分解成 $k\times(n-k)$步基本操作.

4.2.1 修复混合错误的计算开销

本方案在编解码过程中,共调用了 n 次纵向编码算法 $C_{\mathrm{col}}(r+s-1,r-1)$,计算产生了 n 个本地校验扇区、s'个全局校验扇区、$(n-m)\times(s-1)-s'$个临时校验扇区、$m\times(s-1)$个行校验扇区.调用 $r-1$ 次横向编码算法 $C_{\mathrm{row}}(n+m',n-m)$,计算产生了 $m\times(r-s)$个行校验扇区、$m\times(s-1)+s'$个临时校验扇区.分解成基本操作 $Mult\_XORs$,步骤数 β 为

$$\beta=\overbrace{(r-1)\times(n\times s)}^{\text{column}}+\overbrace{(n-m)\times[m\times(r-1)+s']}^{\text{row}}.$$

图 6 为 $s=4$ 时 5 中不同错误分布情况下本方案与 STAIR Codes 方案计算开销对比,其中 $n=8$,$m=2$,$r=8,16,24$.如图 6 所示,本方案总体性能与 STAIR-Upstairs 算法相近,由于本方案添加了本地校验扇区,在修复单个扇区错误时不需要调用全局校验扇区,只需进行 1 次解码运算,所以在抵抗单个扇区错误较多的分布时,本方案 β 值略小于 STAIR-Upstairs 算法.

图 7 为 s,n,r 不同取值时 β 的变化情况.如图 7 所示,β 与 s 成正比,且随着 n,r 的取值增大,β 的增长速率变大.可以看出抵抗 s 个扇区错误,编码条带的规模也影响了计算性能,规模越大,开销越大.

4.2.2 恢复单个扇区错误的计算开销

假设条带中出现 m 个条块错误(对应磁盘错误)及单个扇区错误,对比分析 STAIR Codes 方案的 Upstairs,Downstairs 两种算法与本方案恢复错误数据的计算开销.各方案恢复条块错误的操作基本一致,不同之处在于恢复单个扇区错误.本方案只需要扇区错误所在条块调用 1 次列编码算法即可完成扇区数据恢复,总计操作数为

$$\beta_{\mathrm{IIDR}}=\overbrace{(r-1)}^{\text{column}}+\overbrace{(n-m)\times(m\times r)}^{\text{row}}.$$

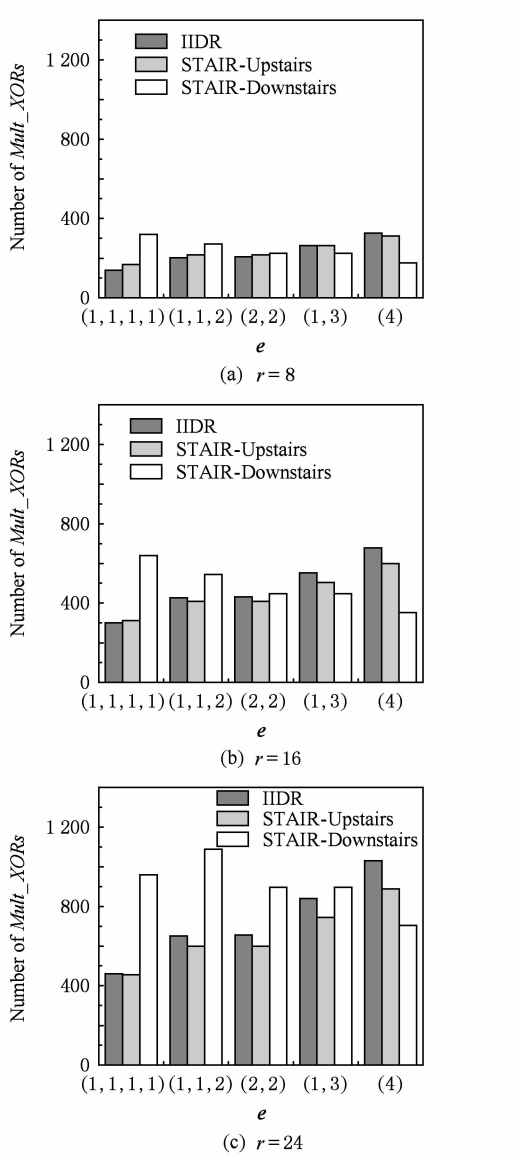


Fig. 6 The number of *Mult_XORs* of STAIR Codes, IIDR versus different ***e***.

图 6 抵抗不同分布错误时 STAIR Codes, IIDR 方案 *Mult_XORs* 操作数

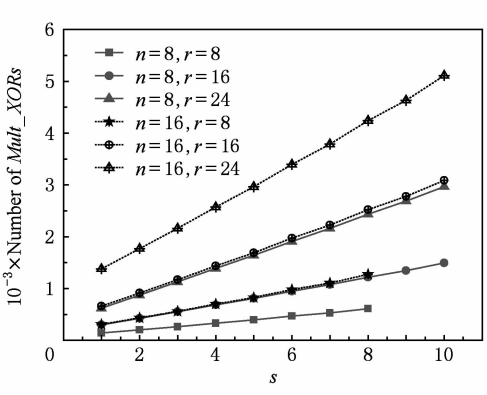


Fig. 7 The number of *Mult_XORs* of IIDR.

图 7 IIDR 方案 *Mult_XORs* 操作数变化图

而 STAIR Codes 方案的 Upstairs, Downstairs 两种算法都需要结合外部全局校验扇区来恢复扇区数据, 总计操作数分别为

$$\beta_{\text{STAIR-Upstairs}}=\overbrace{r\times(n-m)}^{\text{column}}+\overbrace{(n-m)\times(m\times r)}^{\text{row}},$$

$$\beta_{\text{STAIR-Downstairs}}=\overbrace{r}^{\text{column}}+\overbrace{(n-m)\times[(m+1)\times r]}^{\text{row}}.$$

从公式上来看, 本方案计算开销小于其他 2 种算法. 图 8 是 n, r 取不同值时本方案与 STAIR Codes 方案 3 种算法恢复计算操作数对比.

如图 8 所示, 由于本方案添加了本地磁盘校验, 修复单个扇区错误时只需在本地进行 1 次解码操

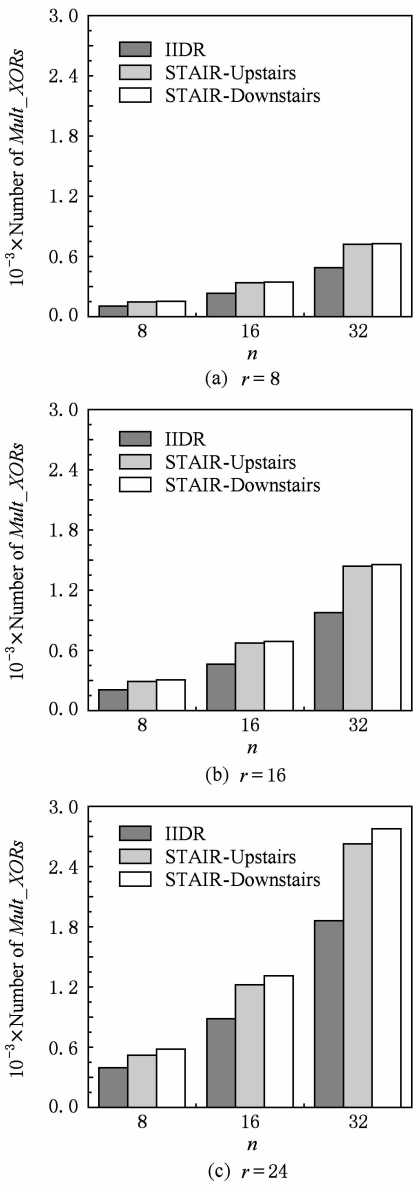


Fig. 8 Compute overhead of a single sector error recovery.

图 8 单个扇区错误恢复计算开销

作,所以在不同的取值组合下本方案操作数均小于其他 2 种算法,特别是随着 n,r 取值增大时,STAIR Codes 行、列编码矩阵增大,计算复杂度进一步增大,本方案优势更加明显. 综上所述,本方案在抵抗单个扇区错误时计算开销小于 STAIR Codes 方案.

再考虑只发生单个扇区错误的情形,本方案仍通过磁盘内冗余进行修复,只需接入读取 1 块磁盘;而 STAIR Codes 则需要利用磁盘间冗余修复,需要读取 $n-m$ 块磁盘. 此种情形下,本方案 I/O 性能也优于 STAIR Codes 方案.

4.3 更新性能

当数据扇区更新时,相应的校验扇区也需要更新,可以通过所需更新的校验扇区数目来衡量方案的更新性能. 更新开销定义为在 1 个条带中更新 1 个数据扇区时,所需更新的校验扇区数目的平均值.

图 9 为 $n=8$,$r=6$,s,m 不同的取值情况下,SD Codes,IDR,IIDR 三个方案更新开销的对比. 如图 9 所示,3 个方案中,SD Codes 方案所需更新的全局校验扇区仅为 s,行校验扇区仅为 m,所以 SD Codes 方案有较好的更新性能. 当 s 较小时,3 个方案相差不大;当 s 较大时,本方案全局校验扇区数目增大,IDR 方案则需要更新较多的行校验扇区,所以 SD Codes 方案优势更为明显,但 SD Codes 方案只能抵抗 $s\leqslant 3$ 的任意错误分布. 本方案开销整体略高于

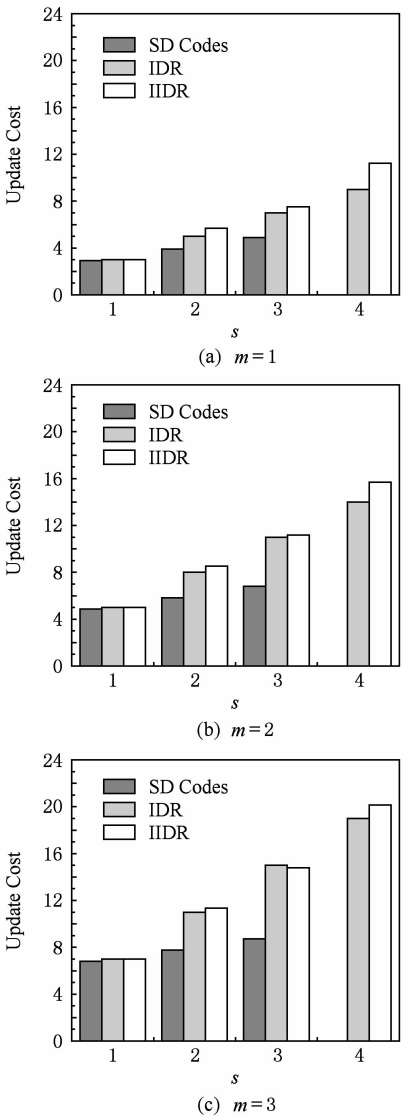


Fig. 9 Update cost of SD Codes, IDR, IIDR for different s.

图 9 s 不同取值下 SD Codes,IDR,IIDR 的更新开销

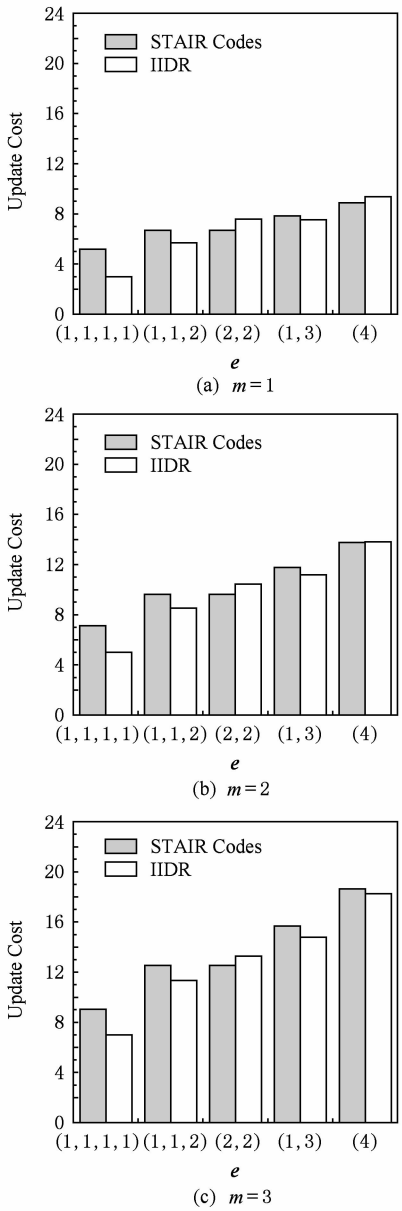


Fig. 10 Update cost of STAIR Codes, IIDR versus different e.

图 10 抵抗不同分布错误时 STAIR Codes,IIDR 方案更新开销

IDR 方案,2 个方案所需更新的行校验扇区基本一致,但随着 s,m 的增大,IDR 需要更新更多的本地校验扇区,则本方案与 IDR 方案更新开销差距随即缩小.

由于 STAIR Codes 方案不支持任意错误分布,只能在抵抗特定错误分布的情况下分析其更新性能.图 10 中设 $s=4$,针对其 5 种分布及 m 的不同取值,分析对比 STAIR Codes 与本方案的更新性能.如图 10 所示,2 个方案性能相差不大,在不同分布下各有优劣.其中分布为(2,2),(4)的情况下,STAIR Codes 更新性能较优;分布为(1,1,1,1),(1,1,2),(1,3)的情况下,本方案更新性能较优.这是由于 2 个方案所需更新的磁盘校验扇区数目基本一致,更新差距主要集中在全局校验扇区,由于本方案校验扇区由本地校验扇区和全局校验扇区组成,其中全局校验扇区数目比 STAIR Codes 少,更新时本方案只需更新单个本地校验扇区及全局校验扇区,所以更新开销比 STAIR Codes 小,特别是在单个扇区错误较多的分布,这种优势尤为明显.

综上所述,本方案更新性能总体优于 STAIR Codes 方案,特别是在抵抗存在单个扇区错误分布情况下优势较为明显.在抵抗任意扇区错误分布时,SD Codes 方案更新性能最优($s\leqslant 3$),本方案总体略差于 IDR 方案,但在部分参数下本方案优于 IDR 方案.

5 结 论

本文将磁盘间编码与磁盘内编码相结合,提出一种能抵抗磁盘错误和任意分布的扇区错误的混合编码方案.与传统的 IDR 方案相比,本方案有更高的磁盘利用率;相比 SD Codes 方案,本方案能够抵抗更多任意分布的扇区错误;相比 STAIR Codes 方案,本方案牺牲了一定的磁盘利用率,但能抵抗任意分布的扇区错误,更新性能也略优,特别是在处理单个扇区错误时有更小的 I/O 和计算开销.

参 考 文 献

[1] Li M, Lee P P C. STAIR codes: A general family of erasure codes for tolerating device and sector failures [J]. ACM Trans on Storage, 2014, 10(4): 1-30

[2] Bairavasundaram L N, Goodson G R, Pasupathy S, et al. An analysis of latent sector errors in disk drives [C] //Proc of the 2007 ACM SIGMETRICS Int Conf on Measurement and Modeling of Computer Systems. New York: ACM, 2007: 289-300

[3] Micron Technology, Inc. N-29-17: NAND flash design and use considerations introduction Micron [EB/OL]. 2006 [2015-06-01]. https://www.micron.com/~/media/documents/products/technical-note/nand-flash/tn2917.pdf

[4] Schroeder B, Damouras S, Gill P. Understanding latent sector errors and how to protect against them [J]. ACM Trans on storage, 2010, 6(3): 523-530

[5] Oprea A, Juels A. A clean-slate look at disk scrubbing [C] //Proc of the 8th USENIX Conf on File and Storage Technologies. Berkeley, CA: USENIX Association, 2010: 57-70

[6] McKusick M K, Joy W N, Leffler S J, et al. A fast file system for UNIX [J]. ACM Trans on Computer Systems, 1984, 2(3): 181-197

[7] Vijayan P, Lakshmi N B, Nitin A, et al. IRON file systems [C] //Proc of the 20th ACM Symp on Operating Systems Principles. New York: ACM, 2005: 206-220

[8] Dholakia A, Eleftheriou E, Hu X Y, et al. A new intra-disk redundancy scheme for high-reliability RAID storage systems in the presence of unrecoverable errors [J]. ACM Trans on Storage, 2008, 4(1): 133-169

[9] Iliadis I, Haas R, Hu X Y, et al. Disk scrubbing versus intra-disk redundancy for high-reliability raid storage systems [C] //Proc of the 2008 ACM SIGMETRICS Int Conf on Measurement and Modeling of Computer Systems. New York: ACM, 2008: 241-252

[10] Plank J S, Blaum M. Sector-disk (SD) erasure codes for mixed failure modes in RAID systems [J]. ACM Trans on Storage, 2014, 10(1): 112-128

[11] Plank J S, Blaum M, Hafner J L. SD codes: Erasure codes designed for how storage systems really fail [C] //Proc of the 11th USENIX Conf on File and Storage Technologies. Berkeley, CA: USENIX Association, 2013: 95-104

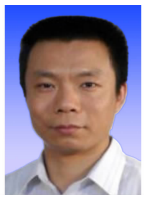
[12] Li M, Lee P P C. STAIR codes: A general family of erasure codes for tolerating device and sector failures in practical storage systems [C] //Proc of the 12th USENIX Conf on File and Storage Technologies. Berkeley, CA: USENIX Association, 2014: 147-162

[13] Luo Xianghong, Shu Jiwu. Summary of research for erasure code in storage system [J]. Journal of Computer Research and Development, 2012, 49(1):1-11(in Chinese)
(罗象宏, 舒继武. 存储系统中的纠删码研究综述[J]. 计算机研究与发展, 2012, 49(1): 1-11)

[14] Plank J S, Greenan K M, Miller E L. Screaming fast Galois field arithmetic using Intel SIMD instructions [C] //Proc of the 11th USENIX Conf on File and Storage Technologies. Berkeley, CA: USENIX Association, 2013: 299-306

Bian Jianchao, born in 1989. PhD candidate. His main research interests include information security, coding theory, storage security and reliability, disaster backup and recovery.

Zha Yaxing, born in 1985. PhD candidate. His main research interests include information security, cryptography, storage security and reliability, disaster backup and recovery (zhayaxing@safe-code.com).



Luo Shoushan, born in 1962. Professor and PhD supervisor in the School of Computer Science at Beijing University of Posts and Telecommunications. His main research interests include information security, cryptography, secure multi-party computation (luoshoushan@safe-code.com).

Li Wei, born in 1986. PhD candidate. His main research interests include information security, cryptography, network security (liwei@safe-code.com).