DOI:10.7544/issn1000-1239.20210575
59(3):582-596,2022

RS类纠删码的译码方法

唐 聃[1,2] 蔡红亮[1] 耿 微[1]
[1](成都信息工程大学软件工程学院 成都 610225)
[2](四川省信息化应用支撑软件工程技术研究中心 成都 610225)
(tangdan@foxmail.com)

Decoding Method of Reed-Solomon Erasure Codes

Tang Dan[1,2], Cai Hongliang[1], and Geng Wei[1]
[1](*School of Software Engineering, Chengdu University of Information Technology, Chengdu* 610225)
[2](*The Software Engineering Technology Research Support Center of Informatization Application of Sichuan, Chengdu* 610225)

Abstract As known, RS (Reed-Solomon) codes can construct any fault-tolerant codewords according to the application environment, which has good flexibility, and the storage system using RS erasure code as the fault-tolerant method can achieve the optimal storage efficiency. However, compared with XOR(exclusive-OR)-based erasure codes, RS erasure codes require too much time to decode, which greatly hinders its use in the distributed storage system. In order to solve this problem, this paper proposes a decoding method for RS erasure codes. This new method completely discards the matrix inversion which is commonly used in all current RS erasure codes decoding methods, and only uses the addition and multiplication with less computational complexity, and the linear combination of the invalid symbols by the valid symbols can obtained by the simple matrix transformation on the constructed decoding transforming matrix, thereby reducing the complexity of decoding calculations. Finally, the correctness of the method is proved theoretically, and for each file of different sizes, three file blocks of different sizes are divided, and the data blocks obtained by the division are tested. The experimental results show that in the case of different file block sizes, the new decoding method has lower decoding time cost than other methods.

Key words RS code; erasure codes; decoding; data reconstruction; recovery cost

摘 要 RS(Reed-Solomon)码可以根据应用环境构造出任意容错能力的码字,有很好的灵活性,且使用RS纠删码作为容错方法的存储系统能达到理论最优的存储效率.但是,与异或(exclusive-OR, XOR)类纠删码相比,RS类纠删码译码计算的时间开销过大,这又很大程度上阻碍了它在分布式存储系统中的使用.针对这一问题,提出了一类RS纠删码的译码方法,该方法完全抛弃了当前大多RS类纠删码译码方法中普遍使用的矩阵求逆运算,仅使用计算复杂度更小的加法和乘法,通过构造译码变换矩阵并在此矩阵上执行相应的简单的矩阵变换,能够直接得出失效码元由有效码元组成的线性组合关系,从而降低译码计算复杂度.最后,通过理论证明了该方法的正确性,并且针对每种不同大小的文件,进行3种不同

收稿日期:2021-06-07;**修回日期**:2021-08-26
基金项目:四川省重点研发计划项目(2020YFG0150)
This work was supported by the Sichuan Provincial Key Research and Development Program (2020YFG0150).
通信作者:耿微(gengwei_work@163.com)

大小文件块的划分,将划分得到的数据块进行实验,实验结果表明:在不同的文件分块大小情况下,该新译码方法较其他方法的译码时间开销更低.

关键词　RS码;纠删码;译码;数据重构;修复成本

中图法分类号　TP302.8

数据的可靠性是存储系统需要考虑的首要问题,而随着信息技术的发展以及向各个行业领域的渗透,需要存储的数据量正持续地爆炸式增长.为了应对海量数据存储的可靠性问题,同时也为了提高数据访问的并发效率,通常有效的做法是使用多个存储节点共同构建1个存储系统,比如集中式的磁盘阵列(redundant arrays of independent disks, RAID)系统或基于网络的分布式存储系统等,而每个存储节点可以是1个磁盘、1台PC、1台服务器或1个移动终端等可用作数据存储的设备.与过去相比,当前单个存储设备的稳定性已经较高,但是对于由众多节点构成的大规模存储系统而言,节点故障事件仍然会频繁发生[1-2].关于如何在部分节点故障后能保证数据不丢失,并且在故障节点被替换后能迅速恢复失效数据是近年来存储领域研究的热点.早期的多节点存储系统多采用副本技术来为数据的可靠性提供保障,该技术把多个相同的数据副本存储到不同的节点上,相同数据的不同副本之间可互为备份,当有存储节点失效时,替换的节点可从任意1个或多个拥有相同数据副本的节点传入数据进行数据重构.但副本技术会导致系统存储效率过低的问题,如果需要在$k(\geqslant 1)$个节点同时失效后能有效重构数据,则需要把原始数据复制k份并分别存放在不同的节点上,整个系统的存储效率仅有$1/(k+1)$.因此,目前更多的研究集中在如何利用纠删码技术来保证存储系统中的数据可靠性.与副本技术相比,使用纠删码来构建存储系统的容错机制可在相同容错能力的条件下,极大地提高了存储效率.

从编译码运算方式的角度看,大体可以把存储系统中使用的纠删码分为2类:1)XOR类纠删码,这类编码的编译码运算过程可以仅使用二进制的异或运算,因此编译码的速度很高,包括了阵列码[2-8]和低密度奇偶校验码[9-11](low density parity check code, LDPC)等.阵列码具有2维编码结构,且通常有几何形式的构造方法,工程实现时简洁直观,但具备极大距离可分(maximum distance deparable, MDS)性质的阵列码通常容错能力不超过3个[5-7],导致其对当前实用的大规模存储系统适应性不足.近年来,也有更高容错能力的阵列码被提出[2,8],但是一般都会付出巨大的存储效率代价.LDPC码可以使得存储效率逼近最优,但该码更适合构造长码,且其成功译码具有概率性,因此目前在存储系统中的使用较少.2)RS(Reed-Solomon codes)类纠删码[12-14],此类编码能达到理论最优的存储效率,即具有MDS性质,且能根据具体环境构造出任意容错能力的码字,具有很强的应用灵活性.由于它独特的优势,目前的存储系统也更多是偏向以RS类纠删码为基础构建其容错机制.但是,RS类纠删码的运算需要在多元有限域上进行,这导致RS类纠删码的计算效率大大低于XOR(exclusive-OR, XOR)类纠删码.对于RS类纠删码而言,其译码运算复杂度又远高于其编码运算.

针对这一问题,多年来不断有学者积极寻求着解决方案.但是,相比于XOR类编码,RS类纠删码很难在数据元素和校验元素之间寻找到有几何规律的码链关系,从而在方法论的层面提升RS类纠删码计算的时间效率相对困难.现在对于RS类纠删码的译码方法研究更多集中在如何提高单节点失效后的重构效率[13]或者减少修复带宽[14],对能整体降低RS类纠删码删除错误重构计算时间开销的方法研究较少[15-16].目前RS类纠删码最为工程领域接受的计算效率提升途径为,使用一些专门针对有限域运算进行优化的指令集或底层方法来达到降低RS类纠删码计算时间成本的目的,其中比较常见的包括GF-Complete[17],Jerasure[18]以及Intel公司推出的ISA-L library[19],但是此类技术路线与本文研究内容关联度不大,故不再详述.

本文工作的主要贡献包括3个方面:

1) 提出了一种针对RS类纠删码码删除错误的译码方法,适用于存储系统中的失效数据重构,由于不再使用计算复杂度很高的矩阵求逆运算,能很大程度节省RS类纠删码的译码时间开销;

2) 新的译码方法也可以推广到所有的二进制XOR类纠删码;

3) 对该译码方法的研究表明:该方法还能在数据重构时规避某些节点的参与,可用于大规模存储

系统的数据重构时对重构参与节点的使用进行整体规划.

1 相关工作及研究动机

为了便于叙述,在此做约定:存储系统有 n 个存储节点,使用某一类(n,k)纠删码作为容错的核心方法,其要点是:数据存储前将原始数据分为 k 个大小相同的数据块,k 个数据块对应着编码的 k 个数据元素(长度为 k 的数据列向量 $\boldsymbol{D}=(d_0,d_1,\cdots,d_{k-1})^{\mathrm{T}}$),编码后形成 n 个数据块,分别存储于存储系统的 n 个不同节点上,而编码得到的 1 个码字 $\boldsymbol{C}$ 则分别对应着码字的 n 个码元(长度为 n 的码字列向量 $\boldsymbol{C}=(d_0,d_1,\cdots,d_{k-1},c_0,c_1,\cdots,c_{n-k-1})^{\mathrm{T}}=(c_0,c_1,\cdots,c_{k-1},\cdots,c_{n-1})^{\mathrm{T}}$,$c_i$ 为码字的码元),其中 k 和 n 均为正整数,k 为原始数据分组的长度,n 为码字的长度,且 $n>k$,$0\leqslant i\leqslant n-1$.文中若提到码元失效亦指该码元对应的存储节点失效,而存储节点失效可以理解为该存储节点上的全部数据丢失(亦即最坏的情况),反之亦然.而本文的所有方法叙述中,若无特别说明,编码的容错能力即指对删除错误的最大恢复能力,对于全部矩阵及存储阵列的行列的计数从 0 开始,且不妨假设所有运算有限域的特征为 2.下面,本节将首先以示例的形式简要描述当前纠删码最常用的 3 类译码方法.

1.1 基于码链的译码方法

大多 XOR 类纠删码的构造可以由几何形式表示,即码字中所有存在校验关系的数据位和校验位的集合可以看作 1 条码链[2],1 条码链即对应 1 个校验方程.特别是在阵列码中,如果用线连接码链上的所有码元则通常具有一定的几何规律.

以 RDP 码[6]为例,使用素数 5 构造出 RDP 码的码字结构如图 1 所示:

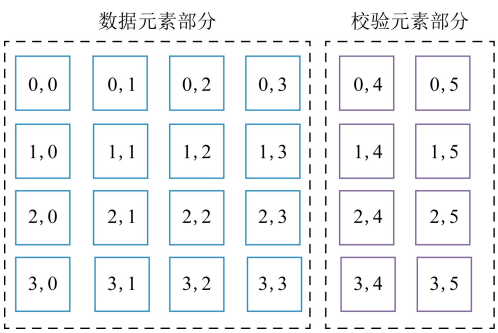


Fig. 1 Data layout of RDP code

图 1 RDP 码的布局结构

根据 RDP 码的编码规则,图 2 中用箭头连接起来的元素和所有图形相同的码元被称为码链,每条码链上所有码元的异或结果为 0.而简单来说,基于码链关系的译码方法,即是按照一定规则搜索仅有 1 个失效码元的码链,利用码链上其他有效码元的异或求得失效码元的值.

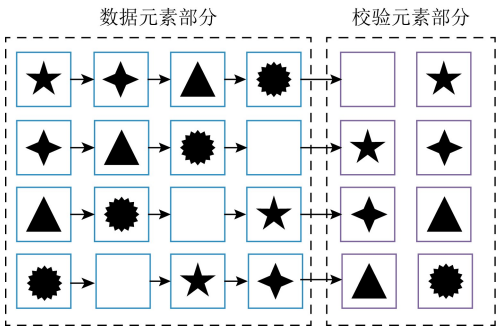


Fig. 2 Check relationship of RDP code

图 2 RDP 码的校验关系

假设在本示例中 RDP 码的第 0 列和第 2 列对应的存储节点失效,则基于码链的数据重构方法操作过程为:首先搜索到码链[(0,1),(1,0),(1,5),(2,4),(3,3)]中仅有 1 个失效码元(1,0),则将该码链上其他所有码元异或,从而得到码元(1,0)的值;码元(1,0)数据恢复后,其所在的另一条码链[(1,0),(1,1),(1,2),(1,3),(1,4)]上也就仅存在 1 个失效码元(1,2),异或该码链上所有有效码元则失效码元(1,2)得到恢复;码元(1,2)数据恢复后,其所在的另一条码链[(0,3),(1,2),(2,1),(3,0),(3,5)]上也就仅存在 1 个失效码元(3,0),异或该码链上所有有效码元则失效码元(3,0)得到恢复;以此类推,可以按照上述规律逐渐按序恢复出其他的失效码元(3,0),(3,2),(0,0),(0,2),(2,0),(2,2).

很多 XOR 类纠删码均可以用该译码方法进行数据重构,而基于码链的译码方法逻辑简单清晰,便于软硬件实现,因此在实际工程中被广泛使用.将运算的有限域 $GF(2^w)$ 上的所有数值替换为对应的 $w\times w$ 的 2 元域矩阵后,基于码链的译码方法也能勉强用于 RS 类纠删码的译码,其中 w 为正整数.但是,RS 类纠删码的构造方法与 XOR 类纠删码差异较大,一般没有清晰的几何编码结构,因此用该译码方法将有较大概率不能完全重构本应该可以恢复的失效码元.

1.2 基于生成矩阵的译码方法

RS 类纠删码的编码通常是采用生成矩阵乘以数据向量的做法,假设数据向量为 $\boldsymbol{D}$,生成矩阵为

$\boldsymbol{G}$,生成矩阵可以是由范德蒙矩阵或者柯西矩阵生成,但是本质上没有太大差异.

图3显示了1个(7,4)RS纠删码的编码过程,生成矩阵的上半部分为1个单位矩阵$\boldsymbol{I}$,下半部分为1个3×4的柯西矩阵$\boldsymbol{P}$,显然这是1个容错能力为3的系统柯西RS纠删码.

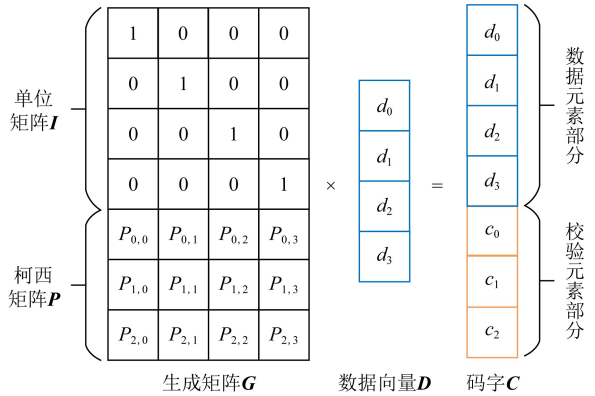


Fig. 3　Encoding process of RS code

图3　RS码的编码过程

其编码过程可表达为$\boldsymbol{G}\times\boldsymbol{D}=\boldsymbol{C}$,其中的参数$\boldsymbol{D}=(d_0,d_1,\cdots,d_{k-1})^{\mathrm{T}}$,$\boldsymbol{C}=(d_0,d_1,\cdots,d_{k-1},c_0,c_1,\cdots,c_{n-k-1})^{\mathrm{T}}$.

使用生成矩阵乘以由数据元素组成的列向量,从而得到由各码字元素构成的码字列向量.由矩阵的乘法定义可知,生成矩阵的第i行构成的行向量乘以数据元素构成的列向量得到码字向量的第i个码元,因此生成矩阵的第i行与码字向量的第i个码元存在对应关系.换句话说,抽取生成矩阵的任意行而组成的子矩阵,乘以数据元素构成的列向量,如果也选取码字向量中与生成矩阵抽取行相同位置的码元构成新的列向量,则图3中的等式仍然成立.

假设码字$\boldsymbol{C}$中的元素d_0和d_2这2个码元失效,我们可以去除码字这2个码元以及生成矩阵中对应的2行,等式仍然成立,如图4所示.

此时,从仍然有效的码元中任意选取4个(比如最后4个码元),构成列向量$\boldsymbol{C}'=(d_3,c_1,c_2,c_3)$,并使用生成矩阵中与列向量$\boldsymbol{C}'$中码元相同位置的第4~7行,构成1个矩阵$\boldsymbol{G}'$,若将数据元素列向量记作$\boldsymbol{D}$,则显然$\boldsymbol{G}'\times\boldsymbol{D}=\boldsymbol{C}'$成立,此时在该等式两边同时乘以矩阵$\boldsymbol{B}$的逆,则能恢复出数据元素$\boldsymbol{D}=\boldsymbol{G}'^{-1}\times\boldsymbol{C}'$.而矩阵$\boldsymbol{B}$的可逆性则由柯西矩阵或者范德蒙矩阵的性质来保证,此处不再赘述.这就是基于生成矩阵译码的基本原理,也可用于任意纠删码的译码.

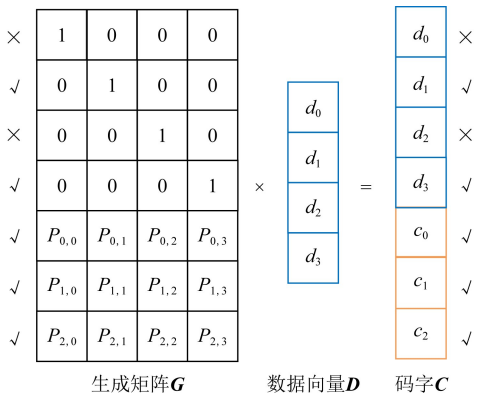


Fig. 4　Decoding process based on the generating matrix

图4　基于生成矩阵的译码过程

基于生成矩阵的译码方法,也有不少学者提出过改进方案.其中最典型的是Blomer等人[20]提出的降低有限域维度方案,该方案将运算在有限域$GF(2^w)$上生成矩阵中的每个元素用1个w阶的0-1方阵表示,源文件分为k个数据块,每个数据块由相同数量的运算单位构成,每个运算单位是1个w位的0-1比特串(看作是1个w维列向量),这样,编译码运算时均只需进行2元域的运算,从而提高计算速度.而此后Plank等人[21]又在此基础之上提出如何减少生成矩阵中数值1的方法,从而进一步减少运算时间.不过,文献[20-21]改进方案本质上是通过降低运算有限域的维度达到的,方法本身和本节叙述的原理没有实质的差异.此外,随着对容错能力要求的提高,此类改进方案可能会导致生成矩阵的尺寸膨胀过快,从而译码运算时将面临高复杂度的大矩阵求逆运算的问题.

1.3　基于校验矩阵的译码方法

基于生成矩阵译码方法的问题为,对1个容错能力为$k(k>1)$的RS纠删码,1个码元失效和k个码元失效,在译码过程中使用到的矩阵求逆运算的次数和所需求逆矩阵的尺寸是相同的.如果码字的数据元素和校验元素同时出现了失效,则使用该方法译码则需要先恢复数据元素,再重新进行编码以恢复校验元素,比较繁琐,也增加了本来不必要的计算工作量.而基于校验矩阵的译码方法可以在很大程度解决上述问题.基于校验矩阵的译码方法几经改进[15-16,22],但译码原理总体一致,本节根据文献[16]的方法,并以1.2节的(7,4)柯西RS纠删码为例进行简要叙述.

1个(7,4)柯西RS纠删码的校验矩阵的尺寸为3×7,其中校验矩阵的前半部分为1个3×4的柯西

矩阵 $\boldsymbol{P}$(和 1.2 节示例中的矩阵 $\boldsymbol{P}$ 相同),后半部分为 1 个单位矩阵 $\boldsymbol{I}$,如图 5 所示:

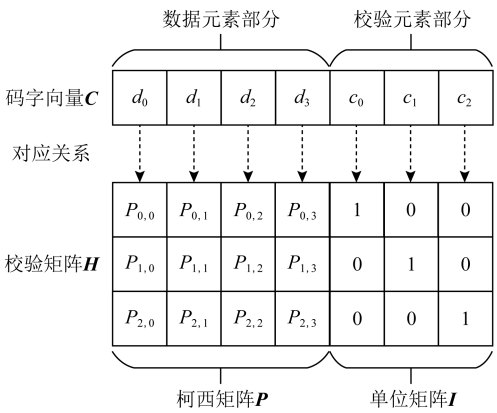


Fig. 5 The check matrix and correspondence between its column vectors and code elements

图 5 校验矩阵及列向量与码字元素的对应关系

由校验矩阵的定义可知,校验矩阵的每一列与码字中的各个码元一一对应:

$$\begin{cases} d_0\times P_{0,0}+d_1\times P_{0,1}+d_2\times P_{0,2}+d_3\times P_{0,3}+c_0=0,\\ d_0\times P_{1,0}+d_1\times P_{1,1}+d_2\times P_{1,2}+d_3\times P_{1,3}+c_1=0,\\ d_0\times P_{2,0}+d_1\times P_{2,1}+d_2\times P_{2,2}+d_3\times P_{2,3}+c_2=0. \end{cases} \tag{1}$$

将校验矩阵记作 $\boldsymbol{H}$,码字向量记作 $\boldsymbol{C}$,则在所有码元没有错误发生时恒有等式 $\boldsymbol{H}\times\boldsymbol{C}=\boldsymbol{0}$ 成立.

此时仍然假设码字中的第 0 个和第 2 个码元,即 d_0 和 d_2 失效,则做操作:如图 6 所示,移动码字向量 $\boldsymbol{C}$ 中的失效码元到最右边,并将其作为 1 个子向量记作 $\boldsymbol{y}$,剩余的有效码元也记作 1 个新的子向量 $\boldsymbol{d}$,$\boldsymbol{d}$ 和 $\boldsymbol{y}$ 组成的向量记作向量 $\boldsymbol{C}=(\boldsymbol{d}\mid\boldsymbol{y})$;然后按照当前向量 $\boldsymbol{C}$ 中各码元和校验矩阵中各列向量

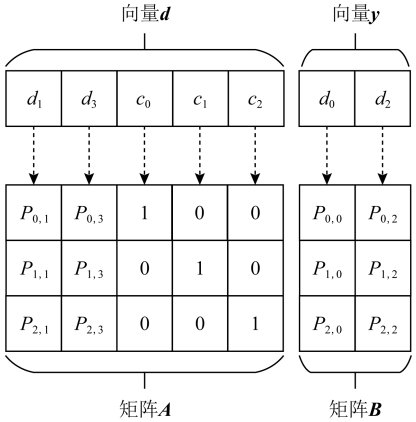


Fig. 6 Adjustment of position of column vectors and code elements in the check matrix

图 6 调整校验矩阵中列向量与码字元素的位置

的对应关系,重新排列校验矩阵中的各列,将向量 $\boldsymbol{d}$ 中码元对应校验矩阵中的各列组成的子矩阵记作 $\boldsymbol{A}$,向量 $\boldsymbol{d}$ 中码元对应校验矩阵中的各列组成的子矩阵记作 $\boldsymbol{B}$,因此校验矩阵可以表示为 $\boldsymbol{H}=(\boldsymbol{A}\mid\boldsymbol{B})$,显然等式 $\boldsymbol{H}\times\boldsymbol{C}=\boldsymbol{A}\times\boldsymbol{d}+\boldsymbol{B}\times\boldsymbol{y}=\boldsymbol{0}$ 成立,因为运算有限域的特征为 2,则等式 $\boldsymbol{A}\times\boldsymbol{d}=\boldsymbol{B}\times\boldsymbol{y}$ 成立.

当码元 d_0 和 d_2 失效,则向量 $\boldsymbol{y}$ 变为未知,此时方程式 $\boldsymbol{A}\times\boldsymbol{d}=\boldsymbol{B}\times\boldsymbol{y}$ 中的未知向量 $\boldsymbol{y}$ 有解,当且仅当矩阵 $\boldsymbol{B}$ 中存在 2(失效码元数量)行线性无关,即 $\boldsymbol{B}$ 中存在 1 个可逆的 2×2 的子矩阵.取出矩阵 $\boldsymbol{B}$ 中的 2 行,构成新的矩阵记作 $\bar{\boldsymbol{B}}$,再从矩阵 $\boldsymbol{A}$ 中取出和作 $\bar{\boldsymbol{B}}$ 中相同位置的 2 行,构成新的矩阵,记作 $\bar{\boldsymbol{A}}$,则等式 $\bar{\boldsymbol{A}}\times\boldsymbol{d}=\bar{\boldsymbol{B}}\times\boldsymbol{y}$ 显然成立.此时,未知向量可由方程 $\boldsymbol{y}=\bar{\boldsymbol{B}}^{-1}\times\bar{\boldsymbol{A}}\times\boldsymbol{d}$ 得出.而矩阵 $\bar{\boldsymbol{B}}$ 的可逆性由柯西矩阵或者范德蒙矩阵的性质保证.

与基于生成矩阵的译码方法相比,本节方法在译码时能减小需要进行求逆运算的矩阵尺寸,对于 1 个 r 容错的 RS 类纠删码而言,如果失效码元数量为 $f(f\leqslant r)$,则需要进行求逆运算的矩阵尺寸为 $f\times f$,当仅有 1 个失效码元需要重构时,则求 1 个特定数值在运算有限域上的乘法逆元即可.因此,此类译码方法的计算效率与基于生成矩阵的译码方法相比有了较为明显的提高.

1.4 本文的研究动机

由于 RS 类纠删码使用了有限域上的计算操作,从而导致了很高的计算时间成本.不过,我们的实验表明,与有限域上的加法及乘法相比,RS 类纠删码译码操作中需要频繁使用到的矩阵求逆运算才是导致 RS 码计算效率低的最重要因素.表 1 显示了在同一台电脑上进行 5 000 次有限域 $GF(2^8)$ 上的加法、乘法,以及不同阶次方阵求逆运算的时间消耗对比.不难看出,有限域上加法和乘法操作的时间开销基本相当,而矩阵求逆运算的时间开销则是要高出加法和乘法运算好几个数量级的,且随着求逆矩阵尺寸的增加而成倍增长,从复杂度上来讲,对于规模为 $n\times n$ 的矩阵来讲,矩阵求逆的运算复杂度为 $O(n^3)$,这也是为什么基于校验矩阵的译码方法时间效率优于基于生成矩阵译码方法的原因.但是,基于校验矩阵的译码方法并没有消除矩阵求逆运算,只是减小了需要进行求逆运算的矩阵尺寸.如果能在译码过程中完全消除矩阵的求逆运算,则能更大程度减小 RS 类纠删码的译码时间开销.这也是本文研究的出发点和动机.

Table 1　Time Cost Comparison of 5 000 Operations over the Finite Field $GF(2^8)$

表 1　有限域 $GF(2^8)$上 5000 次运算操作的时间消耗对比

运算操作	时间消耗/s
加法	0.100
乘法	1.059
3 阶方阵求逆	307.570
4 阶方阵求逆	545.810
5 阶方阵求逆	741.340
6 阶方阵求逆	976.261
7 阶方阵求逆	1682.268
8 阶方阵求逆	2285.434

2　一类新的 RS 类纠删码的译码方法

本节将提出一类新的 RS 类纠删码译码方法,该方法能完全消除译码过程中的矩阵求逆运算操作,从而降低译码操作的译码时间复杂度.在介绍新的方法之前,首先引入码元关系矩阵的概念,并对重要性质进行说明.

2.1　码元关系矩阵

有限域 $GF(q)$上的所有(n,k)线性分组码,码字 $\boldsymbol{C}=(c_0,c_1,c_2\cdots,c_{n-1})^{\mathrm{T}}$ 中任意 1 个码元 c_i,i 为整数且 $0\leqslant i\leqslant n-1$,均可由所有码元的线性组合进行表示,则所有码元可表示为

$$\begin{cases} a_{0,0}c_0+a_{0,1}c_1+\cdots+a_{0,n-1}c_{n-1}=c_0,\\ a_{1,0}c_0+a_{1,1}c_1+\cdots+a_{1,n-1}c_{n-1}=c_1,\\ \vdots\\ a_{n-1,0}c_0+a_{n-1,1}c_1+\cdots+a_{n-1,n-1}c_{n-1}=c_{n-1}. \end{cases} \tag{2}$$

本文将该非齐次线性方程组的系数矩阵定义为码元关系矩阵 $\boldsymbol{W}$.

显然,对于任意 1 个(n,k)线性分组码,其码元关系矩阵一定是 1 个 $n\times n$ 的方阵:

$$\boldsymbol{W}=\begin{pmatrix} a_{0,0} & a_{0,1} & \cdots & a_{0,n-1}\\ a_{1,0} & a_{1,1} & \cdots & a_{1,n-1}\\ \vdots & \vdots & & \vdots\\ a_{n-1,0} & a_{n-1,1} & \cdots & a_{n-1,n-1} \end{pmatrix}. \tag{3}$$

由码元关系矩阵的定义可知,$\boldsymbol{W}$ 的第 i 行 $\boldsymbol{W}_i=(a_{i,0},a_{i,1},\cdots,a_{i,n-1})$代表了码字第 i 个码元 c_i 的一类线性表示方法,即 $c_i=a_{i,0}c_0+a_{i,1}c_1+\cdots+a_{i,n-1}c_{n-1}$,$i$ 为整数且 $0\leqslant i\leqslant n-1$,矩阵 $\boldsymbol{W}$ 的第 j 列对应着参与计算的码字中的第 j 个码元,j 为整数且 $0\leqslant j\leqslant n-1$.

对于任意线性分组码,其码元关系矩阵 $\boldsymbol{W}$ 通常不会是唯一的,但是单位矩阵显然是码元关系矩阵,它表示每一个码元和自身相等.下面给出码元关系矩阵的 2 个性质:

性质 1. 对码元关系矩阵进行任何初等行变换后的结果不再是码元关系矩阵.

证明. 在有限域 $GF(q)$中,矩阵的初等行变换包含 3 类变换[23]:

1) 以 1 个非 0 的数 t 乘以矩阵的某一行,其中 $t\in GF(q)$;

2) 把矩阵的某一行的 t 倍加到另一行,其中 t 为 $GF(q)$中非 0 元素;

3) 互换矩阵中任意 2 行的位置.

下面就这 3 类初等行变换进行分别说明:

1) 根据定义,线性关系矩阵中的第 i 行代表了码字 $\boldsymbol{C}=(c_0,c_1,\cdots,c_{n-1})$中第 i 个码元由所有码元的 1 种线性表示方式:$c_i=a_{i,0}c_0+a_{i,1}c_1+\cdots+a_{i,n-1}c_{n-1}$,其中 i 为整数且 $0\leqslant i\leqslant n-1$.对其中 1 行乘以 1 个非 0 数 t 则等价于把该码元也乘以 t:$t\times c_i=t\times(a_{i,0}c_0+a_{i,1}c_1+\cdots+a_{i,n-1}c_{n-1})$.显然,由于 t 为非 0 数,则 $c_i\neq t\times(a_{i,0}c_0+a_{i,1}c_1+\cdots+a_{i,n-1}c_{n-1})$.因此,在线性关系矩阵中,第 i 行乘以 t 后得到的新的第 i 行$(t\times a_{i,0},t\times a_{i,1},\cdots+t\times a_{i,n-1})$不再是第 i 个元素 c_i 的线性表达,所以 1 个非 0 数 t 乘以原始码元关系矩阵的某一行后的结果不再是线性关系矩阵.

2) 不妨假设将第 j 行的 t 倍加到 i 行上,可以表示为:$t\times c_j+c_i=(t\times a_{j,0}+a_{i,0})c_0+(t\times a_{j,1}+a_{i,1})c_1+\cdots+(t\times a_{j,n-1}+a_{i,n-1})c_{n-1}$,其中 i 和 j 为整数且 $0\leqslant i\leqslant n-1$.除非 $t=0$,否则 $t\times c_j+c_i\neq c_i$.因此,第 j 行的 t 倍加到第 i 行的新的第 i 行$(t\times a_{j,0}+a_{i,0},t\times a_{j,1}+a_{i,1},\cdots,t\times a_{j,n-1}+a_{i,n-1})$已经不再是第 i 个元素 c_i 的线性表达,因此,在线性关系矩阵中,把矩阵的某一行的 t 倍加到另一行之后的结果不再是线性关系矩阵.

3) 根据码元关系矩阵的定义可知,线性关系矩阵的每一行代表了不同的元素,显然任意 2 行相互交换后的结果不再是线性关系矩阵.　　证毕.

性质 2. 对于任意长度为 n 的行向量 $\boldsymbol{v}=(v_0,v_1,\cdots,v_{n-1})$,将 $\boldsymbol{v}$ 的 t 倍加到码字 $\boldsymbol{C}=(c_0,c_1,\cdots,c_{n-1})$对应的码元关系矩阵的任意行,其结果仍然为码元关系矩阵,其中 $v_0c_0+v_1c_1+\cdots+v_{n-1}c_{n-1}=0$,且 $c_i,v_i,t\in GF(q)$.

证明. 根据码元关系矩阵的定义可知,码元关系矩阵的第 i 行 $\boldsymbol{W}_i=(a_{i,0},a_{i,1},\cdots,a_{i,n-1})$为码字

$\boldsymbol{C}$ 中第 i 个码元 c_i 由所有码元的其中 1 种线性表示:$c_i=a_{i,0}c_0+a_{i,1}c_1+\cdots+a_{i,n-1}c_{n-1}$.则等式 $c_i=a_{i,0}c_0+a_{i,1}c_1+\cdots+a_{i,n-1}c_{n-1}+0$ 显然成立.而根据前提条件,对于向量 $\boldsymbol{v}$,有等式 $v_0c_0+v_1c_1+\cdots+v_{n-1}c_{n-1}=0$ 成立.则等式 $tv_0c_0+tv_1c_1+\cdots+tv_{n-1}c_{n-1}=t\times 0=0$ 也成立.那么,等式 $c_i=(a_{i,0}c_0+a_{i,1}c_1+\cdots+a_{i,n-1}c_{n-1})+(tv_0c_0+tv_1c_1+\cdots+tv_{n-1}c_{n-1})$也成立,对该等式归并同类项后得到等式:$c_i=(a_{i,0}+tv_0)c_0+(a_{i,1}+tv_1)c_1+\cdots+(a_{i,n-1}+tv_{n-1})c_{n-1}$.

显然,这也是对第 i 个码元 c_i 的一类线性表示.因此,将向量 $\boldsymbol{v}$ 的 t 倍加到线性关系矩阵任意行后的结果仍然是同一码字的码元关系矩阵. 证毕.

2.2 失效数据恢复步骤

不妨假设在有限域 $GF(q)$ 上构建 1 个 (n,k) RS 纠删码,其中 $q=p^w$,p 为素数,w,n,k 均为正整数,且 $k<n$.假设为原始信息分成 k 组后,经过编码生成码字 $\boldsymbol{C}=(c_0,c_1,\cdots,c_{k-1},c_k,\cdots,c_{n-1})^{\mathrm{T}}$,将其校验矩阵记作 $\boldsymbol{H}$,显然 $\boldsymbol{H}$ 的尺寸为 $(n-k)\times n$,并且 $\boldsymbol{H}\times\boldsymbol{C}=\boldsymbol{0}$,该编码最大可以容忍 $n-k$ 个码元丢失.

假设其中有 f 个码元失效,$f\leqslant n-k$,将失效码元组成的集合标记为 F,$F=\{F_1,F_2,\cdots,F_f\}$,即 $\{F_1,F_2,\cdots,F_f\}=\{c_{s_1},c_{s_2},\cdots,c_{s_f}\}$,其中下标 $\{s_1,s_2,\cdots,s_f\}\in\{0,1,2,\cdots,n-1\}$,也就是下标为失效码元位置.

步骤 1. 初始化码元关系矩阵.生成 1 个尺寸为 $n\times n$ 的单位矩阵 $\boldsymbol{I}_{n\times n}$,根据码元关系矩阵的定义,因为 $c_i=c_i$,i 为整数且 $0\leqslant i\leqslant n-1$,因此 $\boldsymbol{I}_{n\times n}$可以作为初始的码元关系矩阵 $\boldsymbol{W}_{n\times n}$,记作:

$$\boldsymbol{W}_{n\times n}=\boldsymbol{I}_{n\times n}=\begin{pmatrix}1&0&\cdots&0\\0&1&\cdots&0\\\vdots&\vdots&&\vdots\\0&0&\cdots&1\end{pmatrix}. \tag{4}$$

步骤 2. 定义译码变换矩阵并初始化.将码元关系矩阵 $\boldsymbol{W}_{n\times n}$和校验矩阵 $\boldsymbol{H}_{(n-k)\times n}$拼接成 1 个矩阵,称为初始译码变换矩阵,记作 $\boldsymbol{S}^0$:

$$\boldsymbol{S}^0=\begin{pmatrix}\boldsymbol{W}_{n\times n}\\\hline\boldsymbol{H}_{(n-k)\times n}\end{pmatrix}=\begin{pmatrix}\boldsymbol{S\_up}^0\\\hline\boldsymbol{S\_down}^0\end{pmatrix}. \tag{5}$$

显然,矩阵 $\boldsymbol{S}^0$ 的尺寸为 $(2n-k)\times n$,这里的矩阵的下标从 0 开始,将 $\boldsymbol{S}^0$ 中前 n 行构成的子矩阵记作 $\boldsymbol{S\_up}^0$,而将其最后 $n-k$ 行构成的子矩阵记作 $\boldsymbol{S\_down}^0$.

在该译码过程中,对于 f 个丢失码元,需要对初始译码变换矩阵 $\boldsymbol{S}^0$ 进行 f 次变换操作.将经过第 i 次矩阵变换之后更新的译码变换矩阵 $\boldsymbol{S}^i$ 中前 n 行构成的子矩阵记作 $\boldsymbol{S\_up}^i$,而将其最后 $n-k$ 行构成的子矩阵记作作 $\boldsymbol{S\_down}^i$,i 为整数且 $1\leqslant i\leqslant f$.

在每次进行译码变换矩阵的变换过程中,由于只进行了行变换,那么对于 $\boldsymbol{S\_up}^i$ 和 $\boldsymbol{S\_down}^i$,第 i 列均对应着码字中的第 i 个码元,因此译码变换矩阵 $\boldsymbol{S}^i$ 的每一列与码元也有相同的对应关系.

步骤 3. 对于丢失码元集合 F,$\{F_1,F_2,\cdots,F_f\}=\{c_{s_1},c_{s_2},\cdots,c_{s_f}\}$,按序取出失效码元集合 F 中的 1 个元素 c_{s_i},其中$\{s_1,s_2,\cdots,s_f\}\in\{0,1,\cdots,n-1\}$.在矩阵 $\boldsymbol{S}^{i-1}$的丢失码元 c_{s_i} 对应的第 s_i 列上寻找非 0 元素,所有满足条件的非 0 元素的行编号形成集合,标记为 Rs^i,$Rs^i=\{s_1^i,s_2^i,\cdots,s_q^i\}$,其中的元素实际为 q 个非 0 元素所在行的位置,并且$\{s_1^i,s_2^i,\cdots,s_q^i\}\in\{0,1,\cdots,2n-k-1\}$.

步骤 4. 在集合 Rs^i 中搜索一个大于等于 n 的元素,若存在则将该元素标记为 $id\_row=s_v^i$,$v\in\{1,2,\cdots,q\}$,并从 Rs^i 中删除,然后转到步骤 5;否则表明该列对应的码元 c_{s_i} 不可恢复,终止.

步骤 5. 按序取出 $Rs^i=\{s_1^i,s_2^i,\cdots,s_q^i\}$中的一个不等于 id_row 的元素 s_j^i,$j\in\{1,2,\cdots,q\}\cap j\neq v$,并做计算:$\boldsymbol{S}^i(s_j^i,:)=\boldsymbol{S}^{i-1}(s_j^i,:)+\boldsymbol{S}^{i-1}(s_v^i,:)\times(\boldsymbol{S}^{i-1}(s_j^i,s_i)/\boldsymbol{S}^{i-1}(s_v^i,s_i))$计算完成从集合 Rs^i 中删除元素 s_j^i.

步骤 6. 反复执行第 5 步,直到集合 Rs^i 为空.将矩阵 $\boldsymbol{S}^i$ 的第 id_row 行上所有元素置为 0,然后将集合 F 中的元素 c_{s_i} 删除.

步骤 7. 判断矩阵 $\boldsymbol{S}^i$ 的子矩阵 $\boldsymbol{S\_down}^i$ 是否所有元素均为 0,若是则表明不能再恢复任何失效码元,若此时集合 F 中还存在元素则表明这些失效码元不可恢复,终止.

步骤 8. 重复步骤 3~7,直到失效码元集合 F 为空,即所有失效码元可成功恢复,转到步骤 9.当存在失效码元不可恢复时,终止.

步骤 9. 此时 $\boldsymbol{S}^i$ 的上半部分子矩阵 $\boldsymbol{S\_up}^i$ 仍然为码元关系矩阵,其中包含着有效码元对失效码元的线性关系,也可简称为译码矩阵,记 $\boldsymbol{R}$,$\boldsymbol{R}=\boldsymbol{S\_up}^i$,显然,矩阵 $\boldsymbol{R}$ 的尺寸为 $n\times n$,而矩阵 $\boldsymbol{R}$ 的第 s_i 行(列)对应码字的第 s_i 个码元.那么失效码元 c_{s_i} 可以进行恢复:

$$c_{s_i}=\sum_{u=0}^{n-1}\boldsymbol{R}(s_i,u)\times s_u, \tag{6}$$

其中,$1\leqslant i\leqslant f$,$0\leqslant u\leqslant n-1$.

使用步骤1～9,可以对有限域$GF(q)$上的RS类纠删码的失效码元进行恢复,或者是所有失效码元能被恢复成功,若不能将所有码元成功恢复也能清楚地知道哪些码元可恢复而哪些不能恢复.

2.3　方法的正确性及分析

本节首先对和新译码方法相关的一些重要定理进行阐述,并在此基础上说明本文方法的正确性.

定理1. 对于有限域$GF(q)$,$\boldsymbol{S}$为(n,k)线性分组码的译码变换矩阵,则将矩阵$\boldsymbol{S}$第i行乘以t后,矩阵$\boldsymbol{S}$的子矩阵$\boldsymbol{S_down}$仍然为校验矩阵.其中$q=p^m$,p为素数,$t\neq0$,$t\in GF(q)$,i为整数且$n\leqslant i\leqslant n+k-1$.

证明. 由译码变换矩阵$\boldsymbol{S}$的构造方法可知,该矩阵最上面n行为码元关系矩阵,而下面的$n-k$行为该编码的校验矩阵.因此,在译码变换矩阵$\boldsymbol{S}$中最下面的$n-k$行均代表码字的校验方程:$a_{i,0}c_0+a_{i,1}c_1+\cdots+a_{i,k-1}c_k$,其中$i$为整数且$n\leqslant i\leqslant n+k-1$.对译码变换矩阵$\boldsymbol{S}$第$i$行乘以$t$,则等价于$t\times a_{i,0}\times c_0+t\times a_{i,1}\times c_1+\cdots+t\times a_{i,k-1}\times c_{k-1}=t\times(a_{i,0}c_0+a_{i,1}c_1+\cdots+a_{i,k-1}c_{k-1})=0$.因此,对于译码变换矩阵$\boldsymbol{S}$而言,在其最下面的$n-k$行的任意1行上乘以1个非0数$t$后,组成的子矩阵$\boldsymbol{S_down}$仍然为校验矩阵. 证毕.

定理2. 对于任意有限域$GF(q)$,$\boldsymbol{S}$为$n-k$线性分组码的译码变换矩阵,则将矩阵$\boldsymbol{S}$第j行乘以t并加到第i行后,矩阵$\boldsymbol{S}$的子矩阵$\boldsymbol{S_up}$仍然为码元关系矩阵.其中$q=p^m$,p为素数,$t\neq0$,$t\in GF(q)$,i和j均为整数且$n\leqslant j\leqslant n+k-1$,$0\leqslant i\leqslant n+k-1$.

证明. 分2种情况讨论:

1) 当$n\leqslant i\leqslant 2n-k-1$时.此时对译码变换矩阵$\boldsymbol{S}$的变换集中在该矩阵的最下面$n-k$行,该部分由编码的校验矩阵初始化.此时第$i$行对应的校验方程为$a_{i,0}c_0+a_{i,1}c_1+\cdots+a_{i,k-1}c_{k-1}=0$,第$j$行对应的校验方程为$a_{j,0}c_0+a_{j,1}c_1+\cdots+a_{j,k-1}c_{k-1}=0$,则将矩阵$\boldsymbol{S}$第$j$行乘以$t$后加到第$i$行则等价于$(ta_{j,0}+a_{i,0})c_0+(ta_{j,1}+a_{i,1})c_1+\cdots+(ta_{j,k-1}+a_{i,k-1})c_{k-1}=0$,显然成立.

2) 当$0\leqslant i\leqslant n-1$时.此时,译码变换矩阵$\boldsymbol{S}$的第$i$行对应的是码元线性关系方程$c_i=a_{i,0}c_0+a_{i,1}c_1+\cdots+a_{i,n-1}c_{n-1}$,而译码变换矩阵$\boldsymbol{S}$的第$j$行对应的为该编码的校验方程$a_{j,0}c_0+a_{j,1}c_1+\cdots+a_{j,k-1}c_{k-1}=0$.将矩阵$\boldsymbol{S}$第$j$行乘以$t$加到第$i$行则等价于:$c_i=a_{i,0}c_0+a_{i,1}c_1+\cdots+a_{i,n-1}c_{n-1}+0$,显然也成立.这与线性关系矩阵的性质2也刚好吻合.因此,定理2成立. 证毕.

根据2.2节的描述,新的译码方法总是在译码变换矩阵$\boldsymbol{S}$的最下面$n-k$行中选取符合条件的行,标记为第i行,然后将第i行乘以1个数后加到矩阵$\boldsymbol{S}$的另外1行或多行上.换句话说,即对译码变换矩阵$\boldsymbol{S}$进行有一定限制条件的初等变换.由于方法每次选取的第i行均满足条件$n\leqslant i\leqslant n+k-1$.根据定理1和定理2,虽然译码变换矩阵$\boldsymbol{S}$在不断地进行变换和运算,其子矩阵$\boldsymbol{S_up}$一直是码元关系矩阵,并能最终作为失效码元恢复的译码矩阵.综上所述,新方法可以对RS码字中的删除错误进行恢复.

该方法与文献[24]中的基于生成矩阵译码方法的译码方法,及文献[16]的基于校验矩阵的译码方法对比,主要存在2个方面的创新及优势:

1) 算法复杂度较低

对于1个(n,k)RS编码而言,$k<n$.当有f个丢失码元时,采用本文所述方法恢复f个丢失码元的时间复杂度为$O(f\times n)$.

证明. 本文所提的译码恢复方法的主要运算除了最后步骤9中如式(6)恢复计算f个失效码元固定次数$E_0=f\times n$的译码恢复运算,其余主要的运算集中在当$i=1:f$循环执行的步骤3～7对f个丢失码元的译码变换矩阵$\boldsymbol{S}^i$的更新操作,$1\leqslant i\leqslant f$.

在$i=1:f$的每次循环中,其主要的有限域计算主要是步骤5中的对于$j=1:p$且$j\neq v$循环执行的公式所述的运算,这里需要循环的次数为$p-1$,p为对应列非0元素的个数.

在$j=1:p$且$j\neq v$的每次循环中,根据步骤5中的执行公式,可以得出需要执行n次乘法运算,故译码变换运算需执行的运算次数$E_1\leqslant f\times p\times n$,因此可得总体的运算次数$E=E_0+E_1\leqslant(p+1)\times f\times n$,可以得出该方法的时间复杂度为$O(f\times n)$.

而文献[24]所采用的生成矩阵求逆方法运算复杂度约为$O(n^3)$,1个(n,k)线性分组码作为纠删码时,当丢失的码元数量小于其分组码的最小码距的时候,这个纠删码的译码是可解的.文献[16]中采用基于校验矩阵的译码方法,根据1.3节中的方法描述,当其中有f个码元丢失时,其所需要进行的是对校验矩阵提取的对应的$f\times f$满秩的子矩阵进行矩阵求逆,以进行后续的f个丢失码元的恢复,故其复杂度为$O(f^3)$.而文献[25]中的方法根据文中的描述,其复杂度为$O(f\times n^2)$. 证毕.

2) 可同时恢复原始数据码元和校验码元

本文所述方法中丢失的校验码元可在译码过程中直接求得,无需额外计算.该方法可通过变换 1 次性的同时求得原始数据码元以及校验码元,这也有益于减少译码运算,降低所有失效元素恢复的时间.

而文献[24]和文献[25]所述方法中需要先求得原始数据,再通过式(2)所示方程组进行运算得到丢失的校验码元,增加了运算次数.

2.4 新方法示例

我们以在第 1 节中提到的 RS 纠删码为例来说明新方法的执行过程,如 1.2 节所述,这是 1 个(7,4)系统码,首先在有限域 $GF(2^8)$ 上构造出该码的生成矩阵 $\boldsymbol{G}$(其中的子矩阵 $\boldsymbol{P}$ 为柯西矩阵):

$$\boldsymbol{G}=\left(\frac{\boldsymbol{I}}{\boldsymbol{P}}\right)=\begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\\254&253&252&251\\45&104&89&32\\22&44&103&88\end{pmatrix}. \tag{7}$$

不妨假设数据元素向量为 $\boldsymbol{D}=(1,2,3,4)^{\mathrm{T}}$,则码字向量 $\boldsymbol{C}=(c_0,c_1,\cdots,c_6)^{\mathrm{T}}$ 能计算得出:$\boldsymbol{C}=\boldsymbol{G}\times\boldsymbol{D}=(1,2,3,4,0,241,160)^{\mathrm{T}}$.其中有限域的构建方法可参考文献[26],由于篇幅原因本文不再叙述.

仍然假设码字中的第 0 和第 2 个元素失效,则失效码元集合为 $F=\{c_0,c_2\}$.首先初始化译码变换矩阵 $\boldsymbol{S}^0$(子矩阵 $\boldsymbol{W}$ 为单位矩阵、而子矩阵 $\boldsymbol{H}$ 为该编码的校验矩阵):

$$\boldsymbol{S}^0=\left(\frac{\boldsymbol{W}}{\boldsymbol{H}}\right)=\left(\frac{\boldsymbol{S\_up}^0}{\boldsymbol{S\_down}^0}\right)=$$
$$\begin{pmatrix}1&0&0&0&0&0&0\\0&1&0&0&0&0&0\\0&0&1&0&0&0&0\\0&0&0&1&0&0&0\\0&0&0&0&1&0&0\\0&0&0&0&0&1&0\\0&0&0&0&0&0&1\\254&253&252&251&1&0&0\\45&104&89&32&0&1&0\\22&44&103&88&0&0&1\end{pmatrix}. \tag{8}$$

首先,从失效码元序号集合 F 中取出第 1 个元素 $c_{s_1}=c_0$,搜索该元素对应的矩阵 $\boldsymbol{S}^0$ 中的第 $s_1=0$ 列,发现该列中的第 0,7,8,9 行上元素非 0,形成集合 $Rs^1=\{0,7,8,9\}$.从集合 Rs^1 中找出不小于 7 的元素 $id_row=s_v^1=7,v\in\{1,2,\cdots,q\}$并从集合 Rs^1 中删除 ,然后从集合 Rs^1 中按序取出各元素 $s_j^1,j\in\{1,2,\cdots,q\}\bigcap j\neq v$,对译码变换矩阵做操作:

$$\boldsymbol{S}^1(s_j^1,:)=\boldsymbol{S}^0(s_j^1,:)+\boldsymbol{S}^0(7,:)\times(\boldsymbol{S}^0(s_j^1,0)/\boldsymbol{S}^0(7,0)).$$

操作完成后将元素 s_j^1 从集合 Rs^1 中删除.反复从集合 Rs^1 中取出元素 s_j^1 并执行上述操作,直到集合 Rs^1 为空,则从 F 中删除元素 c_0.此时,译码变换矩阵 $\boldsymbol{S}^1$:

$$\boldsymbol{S}^1=\begin{pmatrix}1&255&254&253&3&0&0\\0&1&0&0&0&0&0\\0&0&1&0&0&0&0\\0&0&0&1&0&0&0\\0&0&0&0&1&0&0\\0&0&0&0&0&1&0\\0&0&0&0&0&0&1\\0&0&0&0&0&0&0\\0&125&87&225&47&1&0\\0&43&79&4&24&0&1\end{pmatrix}. \tag{9}$$

当 $i=2$ 时,从失效码元序号集合 F 中取出第 2 个元素 $c_{s_2}=c_2$,搜索该元素对应的矩阵 $\boldsymbol{S}^1$ 中的第 $s_2=2$ 的列,发现该列中的第 0,2,8,9 行上元素非 0,形成集合 $Rs^2=\{0,2,8,9\}$.从集合 Rs^2 中找出数值不小于码长 7 的元素 $id\_row=s_v^2=8,v\in\{1,2,\cdots,q\}$,并从 Rs^2 中删除 id_row,然后从集合 Rs^2 中按序取出各元素 $s_j^2,j\in\{1,2,\cdots,q\}\bigcap j\neq v$,对译码变换矩阵做操作:

$$\boldsymbol{S}^2(s_j^2,:)=\boldsymbol{S}^1(s_j^2,:)+\boldsymbol{S}^1(8,:)\times(\boldsymbol{S}^1(s_j^2,2)/\boldsymbol{S}^1(8,2)).$$

反复从集合 Rs^2 中取出元素并执行 2.2 节中详细步骤操作,直到集合 Rs^2 为空.并删除元素 c_2 后,集合 F 为空,循环退出.所有变换完成后,矩阵 $\boldsymbol{S}^2$:

$$\boldsymbol{S}^2=\begin{pmatrix}0&82&0&9&5&168&0\\0&1&0&0&0&0&0\\0&39&0&139&216&170&0\\0&0&0&1&0&0&0\\0&0&0&0&1&0&0\\0&0&0&0&0&1&0\\0&0&0&0&0&0&1\\0&0&0&0&0&0&0\\0&0&0&0&0&0&0\\0&207&0&97&108&248&1\end{pmatrix}. \tag{10}$$

其中的前 7 行构成的子矩阵 $\boldsymbol{S\_up}^2$ 即为可用作失效元素恢复的码元关系矩阵,即译码矩阵 $\boldsymbol{R}$.

根据码元关系矩阵的定义可以,2个失效码元可以计算得出:

$$\begin{cases} c_0=82\times c_1+9\times c_3+5\times c_4+168\times c_5=0, \\ c_2=39\times c_1+139\times c_3+216\times c_4+170\times c_5=2. \end{cases} \tag{11}$$

3　新方法的性能分析及应用扩展

首先说明实验运行的平台信息(以下简称为实验平台),实验平台使用了Ceph v13.2.3进行分布式存储系统的搭建,所有纠删码代码由Python2.7实现,但是为了编译码方法代码构建与分析的方便,没有直接使用ceph的逻辑存储关系,系统中使用完全相同的计算机作为存储节点.其主要配置信息为:CPU为Intel i5-4570U,6 MB缓存,内存容量为8 GB,而每台计算机上仅搭建1个OSD(object storage daemon).在实验时,每个存储节点对应码字中相同位置的1个码元,码元失效即对应的节点失效.

3.1　RS类纠删码译码的时间开销

本节实验除了使用到本文的译码方法,一方面我们还选用了文献[24]中的译码方法作为基于生成矩阵译码方法的代表,选用了文献[16]的译码方法作为基于校验矩阵的译码方法,以及文献[25]中的译码方法,在同样的环境下进行运行测试.另一方面,选用了硬件加速方法[18]及降低有限域规模的方法[20]进行对比.在这些实验中,若无特别说明,用于编译码测试的原始数据均为10 MB.

实验1. 在实验平台上分别构建(4,2),(5,3),(7,4),(7,3)柯西RS纠删码作为容错方法,计算有限域为$GF(2^8)$.将10 MB原始数据编码后按序存储到各个存储节点中,分别使其中的1个和2个存储节点失效,在用新的节点替换后,分别采用基于生成矩阵的译码方法[24]、基于校验矩阵的译码方法[16]、文献[25]中的译码方法及本文方法重构数据,其数据重构的时间开销分别如图7和图8所示.

总体来看,单个失效节点失效时采用基于校验矩阵的译码方法进行数据重构的时间开销最低,新方法略高于此,但是从2个失效节点开始,采用新译码方法进行数据重构的时间开销就低于另外3种方法.

实验2. 在实验平台上同样构建(4,2),(5,3),(7,4),(7,3)柯西RS纠删码作为容错方法,计算有限域为$GF(2^8)$.将原始数据编码后按序存储到各个存储节点中,采用的原始数据文件大小分别为10 MB,50 MB,100 MB,文件分块大小分别为1 KB,4 KB,1 MB.分别使其中的1个和2个存储节点失效,分别采用基于生成矩阵的译码方法[24]、基于校验矩阵的译码方法[16]、文献[25]中的译码方法及本文方法重构数据,其数据重构的时间开销分别如表2和表3所示.从实验结果来看,在不同的文件分块大小情况下,本文所提的方法总体上较其他方法的译码时间开销更低.

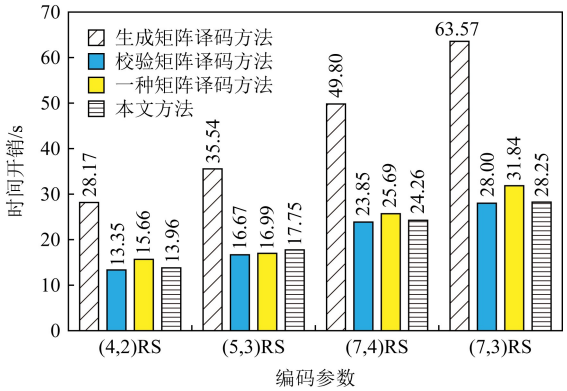


Fig. 7　Comparison of the time cost of four different methods to reconstruct a single failed node

图7　4种不同方法重构单个失效节点的时间开销对比

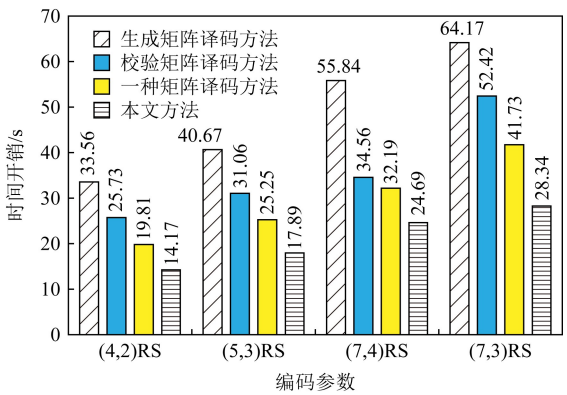


Fig. 8　Comparison of the time cost of four different methods to reconstruct two failed nodes

图8　4种不同方法重构2个失效节点的时间开销对比

实验3. 在实验平台上构建(9,4)柯西RS纠删码作为容错方法,将原始数据编码后按序存储到各个存储节点中,计算有限域为$GF(2^8)$.可知该存储系统的节点容错能力为5,即最多5个节点失效后,其数据可以得到有效恢复.分别使其中的1～5个存储节点失效,并用新的节点替换后,采用文中的方法与本文方法进行数据重构,数据重构的时间开销变化如图9所示.

Table 2 Comparison of the Time Cost of Reconstructing a Single Failed Node

表 2 重构单个失效节点时间开销对比

参数	方法	文件分块大小/KB								
		文件大小=10 MB			文件大小=50 MB			文件大小=100 MB		
		1	4	1 024	1	4	1 024	1	4	1 024
(4,2)	文献[24]	28.17	27.79	28.46	128.29	122.50	121.64	267.94	258.65	269.50
	文献[16]	**13.35**	**13.17**	**13.49**	**54.53**	**51.86**	**56.08**	**114.83**	**113.21**	**115.81**
	文献[25]	15.66	14.48	14.80	56.49	53.73	58.09	116.08	115.44	117.90
	本文方法	13.96	13.77	14.10	56.27	53.52	57.87	115.89	116.92	116.57
(5,3)	文献[24]	35.54	35.14	39.17	169.06	161.35	168.74	350.23	342.67	348.34
	文献[16]	**16.67**	**16.48**	**18.37**	**65.22**	**61.60**	**62.38**	**132.05**	**133.19**	**135.85**
	文献[25]	16.99	17.79	18.72	66.46	62.77	64.53	134.37	135.53	138.24
	本文方法	17.75	16.56	18.46	67.32	63.59	64.33	132.63	133.78	136.45
(7,4)	文献[24]	49.80	46.26	59.04	238.73	235.66	237.46	491.51	482.90	495.38
	文献[16]	**23.85**	**22.16**	**28.28**	**95.19**	**95.64**	**103.21**	**193.22**	**190.62**	**193.39**
	文献[25]	25.69	22.94	29.27	98.52	98.98	106.82	199.64	198.87	196.01
	本文方法	24.26	23.46	24.94	100.79	95.82	109.28	196.00	194.12	200.52
(7,3)	文献[24]	63.57	62.85	69.96	308.47	297.17	312.55	622.90	618.76	627.87
	文献[16]	**28.00**	**29.66**	**33.02**	**145.60**	**140.26**	**150.36**	**284.57**	**282.61**	**291.64**
	文献[25]	31.84	30.49	33.95	147.93	142.51	152.76	292.53	290.53	294.80
	本文方法	28.25	28.92	33.30	146.89	144.36	151.69	290.49	288.49	293.70

注:黑体数字表示译码修复时间最短的.

Table 3 Comparison of the Time Cost of Reconstructing Two Failed Nodes

表 3 重构 2 个失效节点时间开销对比

参数	方法	文件分块大小/KB								
		文件大小=10 MB			文件大小=50 MB			文件大小=100 MB		
		1	4	1 024	1	4	1 024	1	4	1 024
(4,2)	文献[24]	33.56	32.86	33.32	144.72	139.63	143.21	302.16	299.83	304.25
	文献[16]	25.73	24.72	25.08	104.33	109.64	103.21	189.91	184.69	192.25
	文献[25]	19.81	18.38	19.51	79.94	76.59	79.24	148.52	147.36	149.48
	本文方法	**14.17**	**14.41**	**14.12**	**67.16**	**65.70**	**66.72**	**122.23**	**120.13**	**122.83**
(5,3)	文献[24]	40.67	39.59	41.91	188.59	185.15	198.87	379.23	378.32	388.47
	文献[16]	31.06	29.87	30.62	148.87	146.96	149.67	287.06	291.29	299.53
	文献[25]	25.25	24.13	23.47	102.33	102.74	103.99	205.61	199.19	206.86
	本文方法	**17.89**	**17.5**	**18.57**	**71.13**	**70.14**	**74.64**	**142.84**	**139.58**	**145.49**
(7,4)	文献[24]	55.84	52.81	61.24	247.31	242.21	244.26	505.61	501.92	520.29
	文献[16]	34.56	34.45	37.03	178.85	172.97	181.65	306.38	293.9	302.09
	文献[25]	32.19	29.02	32.91	133.5	131.14	136.71	263.59	256.89	270.37
	本文方法	**24.69**	**24.05**	**23.47**	**106.82**	**105.35**	**109.81**	**205.29**	**204.23**	**206.51**
(7,3)	文献[24]	64.17	62.35	73.26	327.34	314.5	338.84	648.65	641.08	653.54
	文献[16]	52.42	51.03	52.97	231.14	225.29	246.37	454.82	443.58	460.95
	文献[25]	41.73	34.97	37.22	191.14	189.84	196.41	361.2	362.31	372.61
	本文方法	**28.69**	**28.79**	**29.19**	**149.08**	**147.27**	**152.36**	**295.03**	**294.72**	**302.13**

注:黑体数字表示译码修复时间最短的.

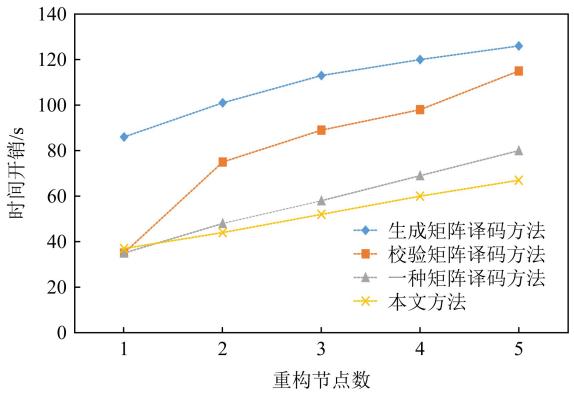


Fig. 9　Comparison of the time cost of four different methods to reconstruct different number of failed nodes

图 9　4 种不同方法重构不同数量失效节点的时间开销对比

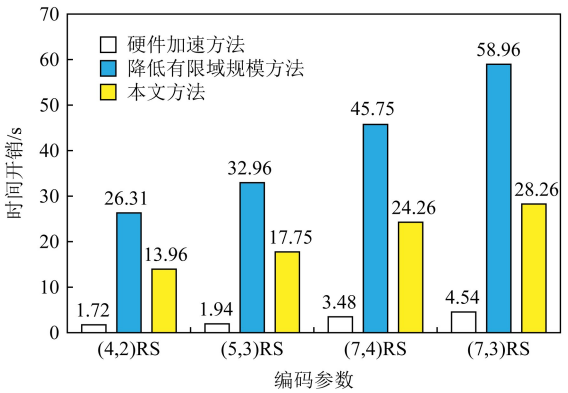


Fig. 10　Comparison of the time cost of three different methods to reconstruct a single failed node

图 10　3 种不同方法重构单个失效节点的时间开销对比

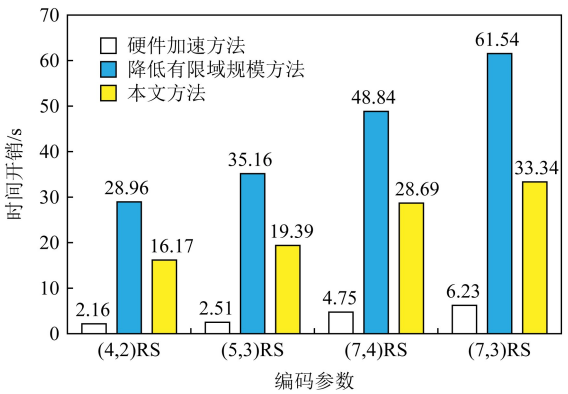


Fig. 11　Comparison of the time cost of three different methods to reconstruct two failed nodes

图 11　3 种不同方法重构 2 个失效节点的时间开销对比

不难看出,采用基于校验矩阵的译码方法进行数据重构的时间开销会随着所需重构节点数的增加而逼近基于生成矩阵的译码方法,文中的方法整体时间开销比本文要高一些,而采用新方法进行数据重构的时间开销则总体随着所需重构数据量的增加而线性增加.

实验 4 和实验 5 主要是将本文所提方法与硬件加速方法 Jerasure[18]、降低有限域规模的方法[20]进行比较.Jerasure[18]采用了硬件加速方法,属于硬件类方法,其通过 SSE 指令集来加速编解码的速度,并且支持不同的有限域上的编码运算,对 RS 编码和柯西 RS 编码的支持较好.Jerasure 已经发展到 2.0 版本,该方法在仅使用 SSE 的情况下,依然能够有效地提升 RS 类纠删码的编译码表现.降低有限域规模的方法主要是将运算进行分解以便在较小的有限域上执行以降低有限域运算的开销.

实验 4. 在分布式存储实验平台上以柯西 RS 纠删码作为其存储编码方法,柯西 RS 编码参数分别为(4,2),(5,3),(7,4),(7,3),计算有限域同样为 $GF(2^8)$.对原始文件采用相应的编码方法进行编码后将编码分片存储到各个存储节点中,当 1 个和 2 个存储节点失效,分别采用硬件加速方法 Jerasure[18]、降低有限域规模的方法[20]和本文方法进行重构,数据重构的时间开销分别如图 10 和图 11 所示.

可以看出,该方法与文献[20]中基于降低有限域规模的方法相比其译码效率显著提高,而稍差于硬件加速方法 Jerasure[18].

实验 5. 在实验平台上以柯西 RS 纠删码作为存储容错方法,编码参数为(8,4),计算有限域为 $GF(2^8)$.经过编码后将数据分存在不同节点,节点失效数目从 1 开始逐渐增大到 4,即最大容错能力.分别采用硬件加速方法[18]、降低有限域规模的方法[20]与本文方法进行数据重构,其数据重构时间开销如图 12 所示.

从图 12 中可以看出,本文所述方法的译码时间较降低有限域规模方法大幅降低,较硬件加速方法偏高,并随着失效节点数目的增加译码时间增长速度较缓.

从实验 4 和实验 5 的结果来看,本文所提的方法的译码表现比同属于软件类的非理论改进的降低有限域规模的方法[20]的表现要好,比基于硬件加速方法的 Jerasure[18]表现要差一些.因本文的研究内容及改进方法都是属于软件方法领域,故从软件方法的角度来讲,本文所提的方法比其他软件理论改进方法要优,而本文所提的理论方法改进与硬件方

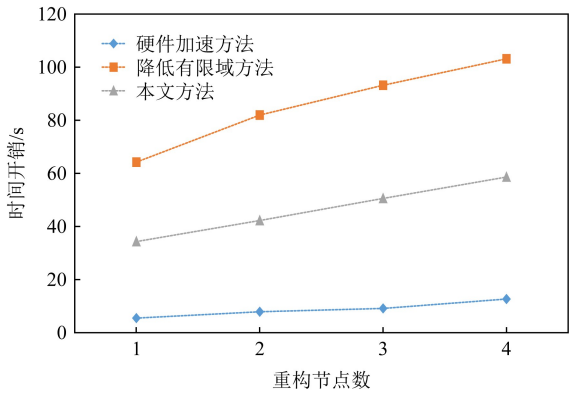


Fig. 12 Comparison of the time cost of three different methods to reconstruct different number of failed nodes

图 12 3 种不同方法重构不同数量失效节点的时间开销对比

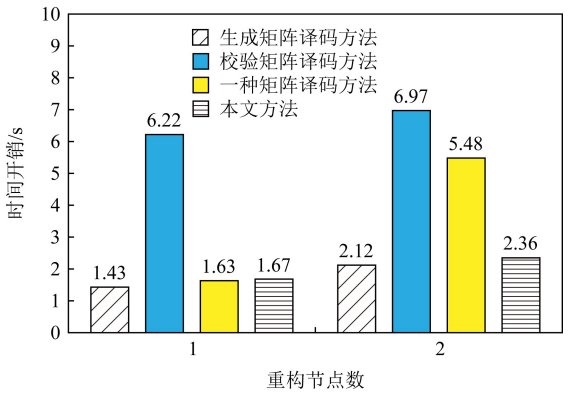


Fig. 13 Time cost of reconstructing in the RDP codes storage system

图 13 基于 RDP 码存储系统中的数据重构时间开销

法的改进并不是对立的,可以将理论方法改进与硬件加速结合来进一步提升编译码效率,这可以作为以后的一个研究方向.

3.2 适用于 XOR 类纠删码译码

一般而言,XOR 类纠删码的构造均有一定几何形式的规律,在编译码操作中通常会基于码链关系进行,这种做法直观形象且更加容易理解,因此最频繁使用的是二进制的异或运算.和 RS 类纠删码一样,XOR 类纠删码也是线性码的 1 个子类,它继承了线性码的所有性质,而对所有线性码的译码方法也能适用.比如,在本文第 1 节中构造的 RDP 码,它的生成矩阵即为 1 个 24×16 的矩阵 $\boldsymbol{G}$,而原始数据元素也可以表示为 1 个 16×1 的列向量 $\boldsymbol{D}$,RDP 码的码字仍然可以计算得出:$\boldsymbol{C}=\boldsymbol{G}\times\boldsymbol{D}$,其中矩阵 $\boldsymbol{G}$、数据向量 $\boldsymbol{D}$ 和码字向量 $\boldsymbol{C}$ 中的元素均为 0 或者 1.所有操作涉及的乘法即为二进制与,而加法则为二进制异或,换句话说 XOR 类纠删码的运算在有限域 $GF(2)$ 上进行.而在 $GF(2)$ 中,$1+1=0$,显然其特征为 2,所以,本文中讲到的基于生成矩阵译码的方法、基于校验矩阵译码的方法和本文提出的新方法均能适用于 XOR 类纠删码.

实验 6. 在实验平台上,使用素数 5 构建 1 个 RDP 码作为容错方法,其存储阵列尺寸为 4×6,计算有限域为 $GF(2)$.将原始数据编码后按序存储到各个存储节点中,分别使其中的第 0 个存储节点失效,以及第 0 个和第 2 个存储节点同时失效,并用新的节点替换后,数据重构的时间开销如图 13 所示.

不难看出,对于基于二进制运算的纠删码译码,采用基于码链关系的译码方法的计算时间开销最小,本文的新方法在计算时间开销上相比略高.

但是,如第 1 节所述,成功使用基于码链的译码方法进行数据重构的关键为能找到仅有 1 个失效元素的码链,而如果不存在这样的码链,则显然无法使用该方法.容易验证,使用素数 5 构造的 EVENODD 码,假设其第 0 列和第 2 列发生错误失效(并没有超过 EVENODD 码的容错能力),则无法使用基于码链的译码方法恢复失效数据.而使用本文提出的译码方法则能顺利恢复所有失效数据.

3.3 数据重构的参与节点规划

将纠删码作为容错方法的存储系统中,失效节点的数据重构过程,是从有效节点中传入与失效数据有相关性的数据并通过计算达到恢复丢失数据的目的.通常而言,恢复丢失数据可以有不止 1 种重构方案,换句话说,码元的线性表达式一般不止 1 个.如果在实际存储系统的运行过程中,存在某些负载过重的节点,则从系统整体运行效率的角度而言是不希望这些节点再参与到重构过程中的.本节将通过 1 个典型示例说明如何使用本文的译码方法规划参与数据重构的有效节点.

继续沿用 2.2 节中的示例,在实验平台上构建 (7,4)柯西 RS 纠删码作为容错方法,将原始数据编码后生成的各个码元按序存储到各对应存储节点中,计算有限域为 $GF(2^8)$.该柯西 RS 纠删码的生成矩阵如式(7)所示,若数据元素向量为 $\boldsymbol{D}=(1,2,3,4)^{\mathrm{T}}$,则计算得到的码元向量为 $\boldsymbol{C}=(1,2,3,4,0,241,160)^{\mathrm{T}}$.仍然假设码元 C_0 和 C_2 失效,即存储系统中第 0 个和第 2 个节点失效.将 2.4 节计算出的译码矩阵记作 $\boldsymbol{R}$:

$$\boldsymbol{R}=\begin{pmatrix}0&82&0&9&5&168&0\\0&1&0&0&0&0&0\\0&39&0&139&216&170&0\\0&0&0&1&0&0&0\\0&0&0&0&1&0&0\\0&0&0&0&0&1&0\\0&0&0&0&0&0&1\end{pmatrix}. \quad (12)$$

因此根据译码矩阵 $\boldsymbol{R}$ 的第 0 行和第 2 行可知,这 2 个失效节点上的数据全部可由式(11)计算得出.恢复失效码元用到的有效码元为 c_1,c_3,c_4,c_5,即存储系统中对应的第 1,3,4,5 个节点上的数据参与了数据重构的过程.

若在存储系统的运行过程中,某个有效节点的负载较重,不希望其参与数据重构的过程,则可以检查此时的译码变换矩阵的子矩阵 $\boldsymbol{S\_down}$,在矩阵 $\boldsymbol{S\_down}$ 中,若该节点对应的列上存在非 0 元素则可以规划该节点不参加重构运算,若该节点对应的列上元素全部为 0,则该节点必需参加到失效节点的重构中.

具体操作时,首先检查某有效节点对应矩阵 $\boldsymbol{S\_down}$ 中的列是否为全 0,若不是则可以在译码方法执行时将该列对应的码元看作失效.比如 2.2 节的示例中,假设我们不希望节点 1 参与重构运算,则首先检查矩阵 $\boldsymbol{S\_down}$ 的第 1 列,发现该列为$(0,0,207)^{\mathrm{T}}$,不是全 0 列,因此节点 1 可以不参与重构运算.在译码时将节点 1 也作为失效节点对待,即设置 $F=\{c_0,c_1,c_2\}$,然后再执行译码方法进行数据重构.这样最终得到的译码矩阵 $\boldsymbol{R}$:

$$\boldsymbol{R}=\begin{pmatrix}0&0&0&54&6&86&131\\0&0&0&146&157&42&50\\0&0&0&102&135&6&88\\0&0&0&1&0&0&0\\0&0&0&0&1&0&0\\0&0&0&0&0&1&0\\0&0&0&0&0&0&1\end{pmatrix}. \quad (13)$$

根据译码矩阵 $\boldsymbol{R}$,2 个失效节点上的数据可计算得出:

$$\begin{cases}c_0=54\times c_3+6\times c_4+86\times c_5+131\times c_6=0,\\ c_2=102\times c_3+135\times c_4+6\times c_5+88\times c_6=2.\end{cases} \quad (14)$$

显然节点 1 此时不再参与失效数据的重构.

当然,计算完成后,译码变换矩阵的子矩阵 $\boldsymbol{S\_down}$ 也变为了零矩阵,换句话说,3 个删除错误已经达到了该编码的最大容错能力,任意的第 4 个节点如果失效将导致所有数据丢失不可恢复.将本来有效的节点作为失效节点,我们是使用这一策略来规避该有效节点参与重构运算.所以,使用本文方法来进行重构运算的节点规划时有如下结论成立:

对于 r 容错能力的纠删码,当有 f 个节点失效时,我们最多能避免 $r-f$ 个有效节点参与重构运算,其中 r,f 为正整数,且 $f\leqslant r$.

4　总　　结

本文提出了一类新的 RS 类纠删码的译码方法,该方法完全避免了在有限域上计算复杂度很高的矩阵求逆运算,仅通过简单的矩阵初等变换,使用加法和乘法即可对失效数据进行重构.经实验表明,该方法与目前 RS 类纠删码译码主要使用的基于生成矩阵和校验矩阵的方法相比,整体译码的计算时间开销有较大幅度的降低.虽然是为了降低 RS 类纠删码译码计算代价提出,但该方法同样适用于 XOR 类纠删码等其他类型的纠删码.此外,新的方法经过少量改动,即可对参与重构运算的有效节点做出动态规划,有利于平衡存储系统的整体性能.

作者贡献声明:唐聃提出了算法思路和实验方案;蔡红亮提出指导意见并修改论文;耿微负责完成实验并撰写论文.

参　考　文　献

[1] Rashmi K V, Shah N B, GuDikang, et al. A solution to the network challenges of data recovery in erasure-coded distributed storage systems: A study on the Facebook warehouse cluster [C/OL] //Proc of the 5th USENIX Workshop on Hot Topics in Storage and File Systems. Berkeley, CA: USENIX Association, 2013 [2020-03-07]. https://www.usenix.org/conference/hotstorage13/workshop-program/presentation/rashmi

[2] Tang Dan, Shu Hongping. A class of array erasure codes with high fault tolerance [J]. Scientia Sinica Informationis, 2016, 46(4): 523-538 (in Chinese)
(唐聃, 舒红平. 一类多容错的阵列纠删码[J]. 中国科学: 信息科学, 2016, 46(4): 523-538)

[3] Wang Yijie, Xu Fangliang, Pei Xiaoqiang. Research on erasure code-based fault-tolerant technology for distributed storage [J]. Chinese Journal of Computers, 2017, 40(1): 236-255 (in Chinese)
(王意洁, 许方亮, 裴晓强. 分布式存储中的纠删码容错技术研究[J]. 计算机学报, 2017, 40(1): 236-255)

[4] Fu Yingxun, Shu Jiwu. D-Code: An efficient RAID-6 code to optimize I/O loads and read performance [C] //Proc of the 29th IEEE Int Parallel and Distributed Processing Symp. Piscataway, NJ: IEEE, 2015: 603-612

[5] Plank J S, Blaum M. Sector-disk (SD) erasure codes for mixed failure modes in RAID systems [J]. ACM Transactions on Storage, 2014, 10(1): 1-17

[6] Corbett P, English B, Goel A, et al. Row-diagonal parity for double disk failure correction [C/OL] //Proc of the 3rd USENIX Conf on File and Storage Technologies. Berkeley, CA: USENIX Association, 2004 [2019-10-18]. https://www.usenix.org/legacy/events/fast04/tech/corbett.html

[7] Cheng Huang, Xu Lihao. STAR: An efficient coding scheme for correcting triple storage node failures [J]. IEEE Transactions on Computers, 2008, 57(7): 889-901

[8] Hafner J L. Weavercodes: Highly fault tolerant erasure codes for storage systems [C] //Proc of the 4th USENIX Conf on File and Storage Technologies. Berkeley, CA: USENIX Association, 2005: 211-224

[9] Gallager R. Low-density parity-check codes [J]. IRE Transactions on Information Theory, 1962, 8(1): 21-28

[10] Elkelesh A, Ebada M, Cammerer S, et al. Decoder-in-the-loop: Genetic optimization-based LDPC code design [J]. IEEE Access, 2019, 7: 141161-141170

[11] Karimi B, Banihashemi A H. Construction of QC LDPC codes with low error floor by efficient systematic search andelimination of trapping sets [J]. IEEE Transactions on Communications, 2019, 68(2): 697-712

[12] Vajha M, Ramkumar V, Puranik B, et al. Clay codes: Moulding MDS codes to yield an MSR code [C] //Proc of the 16th USENIX Conf on File and Storage Technologies. Berkeley, CA: USENIX Association, 2018: 139-154

[13] Li Weiqi, Wang Zhiying, Jafarkhani H. Repairing Reed-Solomon codes over $GF(2l)$ [J]. IEEE Communications Letters, 2020, 24(1): 34-37

[14] Dau S H, Duursma I M, Kiah H M, et al. Repairing Reed-Solomon codes with multiple erasures [J]. IEEE Transactions on Information Theory, 2018, 64(10): 6567-6582

[15] Hellerstein L, Gibson G A, Karp R M, et al. Coding techniques for handling failures in large disk arrays [J]. Algorithmica, 1994, 12(2/3): 182-208

[16] Zhang Yongzhe, Wu Chentao, Li Jie, et al. PCM: A parity-check matrix based approach to improve decoding performance of xor-based erasure codes [C] //Proc of the 34th IEEE Symp on Reliable Distributed Systems. Piscataway, NJ: IEEE, 2015: 182-191

[17] Plank J S, Miller E L, Houston W B. GF-complete: A comprehensive open source library for galois field arithmetic version 1.0 [J/OL]. Electrical Engineering and Computer Sciences, 2013 [2020-03-21]. https://xueshu.baidu.com/usercenter/paper/show?paperid=9c12bc31188f86982e7939de1e09c65a&site=xueshu_se&hitarticle=1

[18] Plank J S, Greenan K M. Jerasure: A library in C facilitating erasure coding for storage applications version 2.0 [R/OL]. Knoxville: University of Tennessee, 2014 [2019-09-20]. http://web.eecs.utk.edu/~jplank/plank/papers/Jerasure-2-0-Archive.pdf

[19] 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

[20] Blomer J, Kalfane M, Karp R, et al. An XOR-based erasure-resilient coding scheme [R/OL]. Berkeley, CA: International Computer Science Institute (ICSI), 1995 [2020-00-00]. https://www.cs.utexas.edu/users/diz/pubs/erasure.pdf

[21] Plank J S, Xu Lihao. Optimizing cauchy Reed-Solomon codes for fault-tolerant network storage applications [C] //Proc of the 5th IEEE Int Symp on Network Computing and Applications. Piscataway, NJ: IEEE, 2006: 173-180

[22] Tang Dan. Research of methods for lost data reconstruction in erasure codes over binary fields [J]. Journal of Electronic Science and Technology, 2010, 14(1): 43-48

[23] Huffman W C, Brualdi R A, Pless V S. Handbook of Coding Theory [M]. Amsterdam: Elsevier, 1998

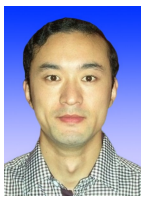
[24] Plank J S. A tutorial on Reed-Solomon coding for fault-tolerance in RAID-like systems [J]. Software Practice and Experience, 1997, 27(9): 995-1012

[25] Hafner J L, Deenadhayalan V, Rao K K, et al. Matrix methods for lost data reconstruction in erasure codes [C] //Proc of the 4th USENIX Conf on File and Storage Technologies. Berkeley, CA: USENIX Association, 2005: 183-196

[26] Lidl R, Niederreiter H. Introduction to Finite Fields and Their Applications [M]. Cambridge, UK: Cambridge University Press, 1994

Tang Dan, born in 1982. PhD, professor. Member of CCF. His main research interests include coding theory, distributed storage.

唐 聃,1982 年生.博士,教授,CCF 会员.主要研究方向为编码理论、分布式存储.



Cai Hongliang, born in 1983. PhD, lecturer. Member of CCF. His main research interests include coding theory, visual cryptography.

蔡红亮,1983 年生.博士,讲师,CCF 会员.主要研究方向为编码理论、视觉密码.

Geng Wei, born in 1997. Master candidate. Student member of CCF. Her main research interests include coding theory, distributed storage.

耿 微,1997 年生.硕士研究生,CCF 学生会员.主要研究方向为编码理论、分布式存储.