GRC：一种适用于多节点失效的高容错低修复成本纠删码

林 轩 王意洁 裴晓强 许方亮 符永铨
(国防科学技术大学 计算机学院 并行与分布处理国家重点实验室 长沙 410073)
(xuanlin90@foxmail.com)

GRC: A High Fault-Tolerance and Low Recovery-Overhead Erasure Code for Multiple Losses

Lin Xuan, Wang Yijie, Pei Xiaoqiang, Xu Fangliang and Fu Yongquan
(*National Key Laboratory for Parallel and Distributed Processing*, *College of Computer*, *National University of Defense Technology*, *Changsha* 410073)

Abstract As the infrastructure of cloud computing, large-scale distributed fault-tolerance storage system are employing erasure codes instead of replication as the data redundant scheme, since the former consume much less storage space than the latter with the same data reliability. However, the high recovery-overhead limits the practical applications of erasure codes. Although a few improved erasure codes have been proposed, they show poor performance at the situation of recovery from multiple node failures. This paper proposes a high fault-tolerance and low recovery-overhead erasure code called Group Repairable Codes(GRC), which is efficient in multiple node failures. GRC reduces the data volume needed to be transmitted when constructing failed blocks by adding group parities for the data and parity blocks, which saves the resources of network and disk I/O. Further, GRC reduces the recovery-overhead for repairing the multiple node failures by adding multiple group parity blocks. Based on the characters of GRC, the paper proposed GSBD(Greedy Strategy Based Decode Algorithm), which reduces the data volume in the whole recovery by ensuring the cost of recovery to be minimum each time. The results of thorough experiments in a real cluster show that, compared with RS code, GRC can reduces network resource consumption by 50%-55% and improve repair speed by 75%-90% with 21% extra storage overhead, compared with LRC code, GRC can reduces network resource consumption by 35%-45% and improve repair speed by 40%-50% with 13% extra storage overhead, compared with Basic Pyramid codes, GRC can reduces network resource consumption by 15%−25% and improve repair speed by 20%−25% with 6% extra storage overhead.

Key words distributed storage system; multiple losses recovery; erasure codes; data reconstruction

摘 要 作为云计算重要基础的大规模分布式容错存储系统，采用纠删码作为数据冗余技术能比多副本技术以更低的存储开销获得相同的数据可靠性.然而，过高的修复成本使纠删码技术在实际中的应用受到限制.已有的改进工作虽然可以降低成本，但在多节点失效修复的成本过高.本文提出一种适用于多节点失效的高容错低修复成本纠删码——分组修复码(Group Repairable Codes，简称 GRC).GRC 码通过将条带分组并增加组编码块，显著减少了修复所要传输的数据量，从而节省了宝贵的网络带宽和磁盘 I/O 资源；GRC 码通过多个组编码块在多节点失效时降低修复成本，且维持较好容错能力.根据 GRC 码的特征，本文提出基于贪心策略的解码算法 GSBD(Greedy Strategy Based Decode Algorithm),GSBD 通过保证每个失效块的修复成本最小以优化修复过程.实验结果显示，与 RS 码相比，GRC 码将修复网络带宽和磁盘 I/O 降低 50%−55%，修复速度提高 75%−90%，仅需增加 21%存储空间；与 LRC 码相比，GRC 码将修复网络带宽和磁盘 I/O 降低 35%−45%，修复速度提高 40%−50%，仅需增加 13%存储空间；与 Basic Pyramid Code(简称 BPC)相比，GRC 码将修复网络

收稿日期:
基金项目:国家重点基础研究发展规划(973)项目(2011CB302601);国家自然科学基金项目(61379052);国家高技术研究发展计划(863计划)项目(2013AA01A213);湖南省自然科学杰出青年基金项目(14JJ1026); 高等学校博士学科点专项科研基金资助课题(20124307110015);国家自然科学基金项目(61402509)
通信作者: 王意洁 Email: wangyijie@nudt.edu.cn

带宽和磁盘 I/O 降低 15%-25%，修复速度提高 20%-25%，仅需增加 6%存储空间.

关键词 分布式存储系统;多节点失效;纠删码;数据修复
中图法分类号 TP393

1 引言

在云计算[1]与数据密集型计算等领域，规模庞大和快速增长的数据需要低价格和易扩展的存储[2].集中式存储由于容易产生性能瓶颈而不能满足需求，构建于大量廉价商用硬件的分布式存储系统将数据分散到不同的存储节点，突破了性能瓶颈.但由于庞大的系统规模，导致节点失效成为常态现象，需要采取数据冗余技术保证数据的可靠性.

目前的数据冗余技术包括多副本技术[3]和纠删码技术[4].多副本技术通过复制数据到多个节点提高数据可靠性，得到了广泛应用，谷歌的 GFS[5]、HDFS[6]、Facebook[7]的图片存储系统等都采用副本技术.但是，多副本技术的空间利用率低，为了获得较高的可靠性，不得不消耗几倍于原数据的存储空间.例如，现在通常采用的 3 副本机制，需要消耗 3 倍于原数据的存储空间.这在数据量较小时还可以承受，然而随着大数据时代的到来，海量的数据使得多副本技术存储成本急剧增大，迫切需要降低数据存储成本[8].

纠删码技术能以极低的数据存储开销实现与多副本技术相同甚至更高的可靠性[9][10]. 系统型 MDS 纠删码[11]是最常见的纠删码，它将k个数据块经过数学计算产生n个块($n > k$)，这n个块中包含原有的k个数据块.应用 MDS 纠删码的存储系统只需消耗 1.2 倍于原数据的存储开销就可以获得和 3 副本存储系统相同的容错性，但却大大减少了额外存储空间.因此，纠删码技术在分布式存储领域越来越受到重视，Facebook[12]、Google[13][14]和 Microsoft[15]等公司都尝试将纠删码应用到自己的分布式存储系统中.

然而，过高的纠删码修复成本阻碍了纠删码技术的实用性.当一个数据块失效时，纠删码存储系统需要传输数倍于数据块大小的数据量来修复，不仅占用较高的网络资源，而且严重降低数据读取速度[16].纠删码修复对网络资源的占用会影响到系统中其他正在执行的任务，比如 Hadoop 中的 MapReduce 任务，而[17]发现，MapReduce 任务的最大瓶颈就是网络带宽.

研究者们针对数据修复问题提出了很多改进型纠删码.Dimakis 等提出了再生码[18][19]，Huang C 等提出了 Pyramid 码[20]和 LRC 码[9]，Sathiamoorthy M 提出了 XORing elephants[12].已有工作存在两方面不足：（1）关注单点失效的低成本快速修复.然而，多点失效的修复在分布式系统中并不少见.比如，一些系统像 Total Recall[20]应用的是延迟修复策略，即在失效节点达到一定的数目才开始修复.（2）关注数据块失效的低成本快速修复，而在编码块失效时修复成本过高. Pyramid 码和 LRC 码通过将数据条带分组生成局部冗余(local parities)[9]降低数据修复带宽，但忽略为编码条带生成局部冗余，所以在编码块失效时需要整个条带的块参与修复，修复成本高.

针对现有纠删码在多点失效修复时效率不高问题，本文提出一种适用于多点失效的高容错低修复成本纠删码——分组修复码(Group Repairable Codes，简称 GRC).GRC 码通过将条带分组并生成多个组编码块，在多节点失效时降低修复成本；通过为编码条带增加局部冗余，进一步降低成本.实验结果显示，与 RS 码相比，GRC 码将修复网络带宽和磁盘 I/O 降低 50%-55%，修复速度提高 75%-90%，仅需增加 21%存储空间；与 LRC 码相比，GRC 码将修复网络带宽和磁盘 I/O 降低 35%-45%，修复速度提高 40%-50%，仅需增加 13%存储空间；与 Basic Pyramid Code(BPC)[20]相比，GRC 码将修复网络带宽和磁盘 I/O 降低 15%-25%，修复速度提高 20%-25%，仅需增加 6%存储空间.

本文第 2 节介绍和分析以降低修复成本为目标的纠删码；第 3 节提出了一种适用于多点失效的高容错低修复成本纠删码 GRC；第 4 节介绍 GRC 码实现；第 5 节通过实验全面测试 GRC 码性能，并与 RS 码、LRC 码和 BPC 码进行对比；第 6 节总结全文.

2 相关工作

近年来，纠删码技术因其能以极低的额外存储空间开销获得极高的数据可靠性而在分布式系统中越来越受重视，但过高的修复成本阻碍了纠删码在实际系统中的应用，改进纠删码以降低修复成本的研究受到了广泛的关注.

Dimakis 等人提出了降低修复成本的再生码[17][19]. 再生码分为 $k/n \le 1/2$ 的低比率码和 $k/n > 1/2$ 的高比率码.对于低比率码(即存储开销是原数据的 2 倍多)，已经有比较完美的构造方法[22][23].但相对于常见的 3 副本系统，将纠删码系统的存储开销降低到原数据的 1.4-1.8 倍是目标[12]，所以低比率纠删码的实际意义并不大.高比率再生码目前只在理论上被证明了其存在性，但还没有实际的构造，而且此类纠删码需要较高的磁盘 I/O 资源.

另一类降低纠删码修复成本的方法是为纠删码增加局部冗余[20][24][25]，此方法虽然打破 MDS 码的最优存储利用率，但是有效降低了网络流量.Huang C 等提出了 Pyramid 码[20]和 LRC 码[9].

Pyramid 码和 LRC 码都通过为数据条带增加局部冗余降低数据块修复网络带宽和磁盘 I/O，但它们的编码块修复成本依然很高.Sathiamoorthy M 提出了纠删码 XORing elephants[12]，XORing elephant 码将数据块分成若干组，每组产生一个局部编码块，但由于每组编码块数目为 1，所以此纠删码修复多节点失效开销高.周松等提出了阵列结构的纠删码 EXPyramid[26]，EXPyramid 码为数据阵列同时取行冗余和列冗余来降低修复成本.但由于其空间利用率低，所以并不适用于大规模存储系统.

为了便于理解，基于文献[27][28]，给出一些基本概念的说明和定义：

1) 数据块——原始文件经过简单的分割得到的一定大小的数据串.
2) 编码块——利用纠删码算法，通过计算数据块而得到的存储冗余信息的数据串.
3) 条带——独立地与同一纠删码算法相关的所有信息的集合.
4) 数据条带——条带中所有数据块的集合.
5) 编码条带——条带中所有编码块的集合.
6) 条带长度——条带中包含的块数.
7) 容错度——纠删码可以容任意块失效数目.
8) MDS 码——满足 Singleton 边界条件[29]的编码方式，是达到理论上最优存储利用率的一类纠删码的统称.

3 分组修复码 GRC

3.1 纠删码的数据修复问题定义

用(n,k)表示一个纠删码，其中k个数据块$D_1,D_2,...,D_k$，计算产生n个块$C_1,C_2,...,C_n$，这个过程称为编码. 具有 MDS 性质的纠删码统称为 MDS 码，MDS 码保证在n个块中，只要有k个块是可用的则数据可用. 当$C_1,C_2,...,C_n$中不大于$n-k$个块失效时，则可选其中任意k个块来计算产生失效块，这个过程称为解码(或修复). 为了便于理解，基于文献[27][28]，给出纠删码数据修复问题的定义.

定义 1. 修复成本. 纠删码修复过程需要读取的数据量称为修复成本.

分布式存储系统中，修复过程对别的任务的影响主要表现在占用网络带宽和磁盘 I/O 资源，所以修复过程读取的数据量越小越好，即纠删码的修复成本越小，则存储系统的性能越好. (n,k)MDS 码中每个块的修复成本都是k个块，GRC 码的目标是减少修复成本，即用少于k个块修复一个失效块.

定义 2. 修复率. 当失效数r给定时，(n,k)纠删码共有$\binom{n}{r}$种失效情况，所有可修复情况数和所有失效情况数的比值称为修复率.

定义 3. 全局编码块. 条带中的所有数据块计算得到的存储冗余信息的数据串.

定义 4. 组编码块（或称局部编码块）. 条带中部分数据块计算得到的存储冗余信息的数据串.

本文提出的 GRC 码将条带分组，每组生成的编码块即是组编码块.组编码块计算方法和全局编码块相同(只要将其他组的数据块置零)，所以可看成全局编码块在各组的投影.组编码块可以合成(通过有限域上的加，即异或运算)对应的全局编码块. 本文频繁用到的符号如表 1 所示.

表 1. 符号表

符号	含义
n	条带长度
k	数据条带长度
m	编码条带长度
m_0	全局编码块数
m_1	一个组的组编码块数
r	失效块数
D_i	条带中第i个数据块
P_i	条带中第i个全局编码块
P_{il}	P_i在第l组的投影生成的组编码块

3.2 基本思想

纠删码技术中，应用最为广泛的 MDS 码在理论上被证明具有相同存储开销下的最优容错能力，但 MDS 码修复时需要传输k个块，占用了大量网络带宽，而网络带宽资源是分布式存储系统的性能瓶颈.为了降低纠删码的修复成本，研究者们提出了一些由 MDS 码改进得到的纠删码，比如应用分组思想的 LRC 码. LRC 码将数据条带分组并在每个组内计算产生一个局部编码块，以保证单块失效时能够在组内修复，降低修复成本.但由于 LRC 码在每组只产生一个编码块，无法在组内修复多节点失效情况，在多节点失效时修复成本高.

为了解决多节点失效修复成本高的问题，本文提出了 GRC 码.GRC 码是一种基于 MDS 码的改进型纠删码，为了描述方便，记原 MDS 码为(n,k)MDS 码，其中n表示整个条带的长度，k表示数据条带的长度，$n>k$恒成立.GRC 码将数据条带分成L个组，每个组中的数据块数目为k/L.GRC 码在每个组内计算产生m_1个组编码块.组编码块保证了 GRC 码在多节点失效修复时只需传输k/L个块，传输的数据量是 MDS 码和 LRC 码在多节点修复时的$1/L$倍.为了保持一定的容错度，GRC 码基于整个数据条带计算产生m_0个全局编码块，通常情况下m_0远小于数据条带长度k.通过控制生成矩阵，GRC 码各组中序号相同的组编码块通过有限域上的加法运算可合成m_1个全局编码块，合成的m_1个全局编码块和已有的m_0个全局编码块共同维持 GRC 码m_0+m_1的容错度.此外，GRC 码将m_0个全局编码块集合作为一组并计算产生组内编码块，使得这m_0个全局编码块失效修复时也可在组内完成，故只需传输m_0个块，相比于原本传输的k个块，修复成本大幅降低.

相比于 MDS 码和 LRC 码，GRC 码大幅降低了

多节点失效修复时的数据传输量.综上所述，GRC 码通过增加少量冗余显著降低修复成本，体现了 GRC 码用少量存储空间换网络带宽资源的思想.在数据传输量大的云计算存储中心，网络带宽是瓶颈，通过增加少量的存储开销显著减少修复网络带宽具有重要的实用价值.

3.3 编码算法

GRC 码是基于 MDS 码的一种改进型编码，所以本节首先介绍 MDS 码，然后给出 GRC 码的构造流程和编码公式，最后举例演示 GRC 编码过程.

如公式(1)所示，(n,k)MDS 码中由k个数据块$D_1,D_2,...,D_k$产生n个块$C_1,C_2,...,C_n$.若产生的n个块集合中包含全部的k个原数据块，这样的纠删码称为系统型纠删码，简称系统码[31].系统码因其良好的访问性能，成为大部分系统的首选，本文提出的 GRC 码是一种基于系统型 MDS 码的改进型纠删码.(n,k)MDS 码的编码公式可表示为：

$$G\times D=\begin{pmatrix} g_{11} & \cdots & g_{1k} \\ g_{21} & \cdots & g_{2k} \\ \vdots & \ddots & \vdots \\ g_{n1} & \cdots & g_{nk} \end{pmatrix}\times\begin{pmatrix} D_1 \\ D_2 \\ \vdots \\ D_k \end{pmatrix}=\begin{pmatrix} C_1 \\ C_2 \\ \vdots \\ C_n \end{pmatrix} \quad (1)$$

若 (n,k) MDS 码是系统码，则有$C_i=D_i(1\le i\le k)$，假设m个编码块表示成$P_1,P_2,...,P_m$，则有$C_{i+k}=P_i(1\le i\le m)$.系统码在编码过程中保持数据块不变，主要工作是生成编码块P_i，为了表示简洁，可以用P_i的产生公式表示系统码的编码算法.

一般的，GRC 码按如下方式构造.从(n,k)MDS 码出发，将数据条带分成L个组(记为$S_l,l=1,2,...,L$)，组S_l包含k_l个块.保持原纠删码前m_0个编码块不变，为每个组计算$m_1=m-m_0$个组编码块，组S_l的编码块$P_{il}(m_0<i\le m)$计算方法和P_i的一样，只要将除S_l外其他组的数据块置零.将前m_0个不变的编码块作为第$L+1$组，记为S_{L+1}，将S_{L+1}中m_0个编码块进行异或运算生成组编码块P_{m+1}.GRC 码的编码公式如下：

$$P_i=\sum_{j=1}^{k}g_{ij}D_j(0<i\le m_0) \quad (2)$$

$$P_{il}=\sum_{j=(l-1)\times(k/L)+1}^{l\times(k/L)}g_{ij}D_j(0<l\le L) \quad (3)$$

$$P_{m+1}=\sum_{j=1}^{m_0}P_j \quad (4)$$

定理 1. 一个由(n,k)MDS 码出发构造的 GRC 码可容任意$m=n-k$个块失效；且前L个组都是(k_l+m_1,k_l)MDS 码.

证明：假设任意m个块失效，其中含r个组编码块，则有$m-r$个数据块或全局编码块失效.分为两种情况：1)$r\ge m_1$；或 2)$r<m_1$，当$r\ge m_1$时，将所有的组编码块看成失效，从原 MDS 码角度看m_1个块失效，加上$m-r$个其他数据块或编码块失效，共有$m_1+m-r\le m$个块失效，这种情况可修复.当$r<m_1$时，最坏情况是所有失效组编码块P_{il}有不同的i，即在m_1中有r个失效块，从原 MDS 码角度看共有$r+(m-r)=m$个失效块，这种情况可修复.

第二部分用反证法，假设一个组S_l是(k_l+m_1,k_l)非 MDS 码，即S_l不能容任意m_1个块失效.考虑一种特殊情况，除了S_l其他组的数据块全为 0，则S_l加上m_0个全局编码块，就变为原 MDS 码.假设所有m_0个全局编码块全部失效，S_l中有任意m_1个失效块，则共有$m_0+m_1=m$个失效块，从原 MDS 码的角度看，可修复.由于m_0个全局编码块在修复中并没有被用到，说明S_l可以修复任意m_1个失效块，与假设矛盾.

证毕.

相比于 MDS 码，GRC 码通过分组并在组内计算产生多个组编码块以降低多节点失效修复时的数据传输量.另外，GRC 码将全局编码块集合作为一组并为其生成局部冗余，从而降低全局编码块在修复时的数据传输量.

下面从(14,10)MDS 码出发构造(17,10)GRC 码以演示编码过程.图 1 显示了(14,10)MDS 码的编码结构，$D_1\sim D_{10}$表示数据对象分割后得到的数据块，$P_1\sim P_4$表示数据块经过计算后产生的编码块.

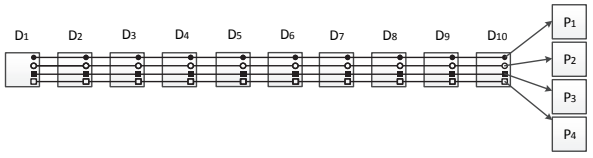

图 1 (14,10)MDS 码示意图

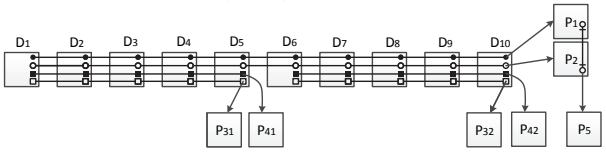

图 2 (17,10)GRC 码示意图

如图 2 所示，GRC 码将 MDS 码中 10 个数据块分成数目相等的两组$S_1=\{D_1,D_2,D_3,D_4,D_5\}$，$S_2=\{D_6,D_7,D_8,D_9,D_{10}\}$，并保持 MDS 码的两个全局编码块$P_1,P_2$不变.计算组编码块$P_{31},P_{41}$，计算方法和计算$P_3,P_4$一样，只是让$S_2$中所有数据块置零，同理计算组编码块$P_{32},P_{42}$. 将保持不变的全局编码块集合记为$S_3=\{P_1,P_2\}$，计算$S_3$的组编码块$P_5$.编码块$P_3$的计算公式为$P_3=\sum_{j=1}^{10}g_{3j}D_j$，组编码块$P_{31},P_{41}$的计算公式分别为$P_{31}=\sum_{j=1}^{5}g_{3j}D_j$，$P_{32}=\sum_{j=6}^{10}g_{3j}D_j$，因此$P_3$=$P_{31}$+$P_{32}$(所有的$\sum$和+在有限域上都是异或运算)，同理有$P_4$=$P_{41}$+$P_{42}$.相比于 MDS 码，GRC 码将数据条带分组并在组内取多个编码块，这种设计可以保证 GRC 码在多节点失

效时也能够在组内完成修复，传输更少数目的块，降低多节点失效的修复成本.GRC 码不但为数据条带分组取局部冗余，也将全局编码块分成一组.如图 2 所示，将P_1, P_2作为一组，并在组内计算产生局部冗余块P_5，以保证全局编码块P_1, P_2的失效修复只需传输两个块，相比于传输k个块，显著降低了修复成本.

3.4 解码算法

(n,k)MDS 码在数据块失效时，可以根据失效块数目直接判断是否可修复.如果失效块数目大于$n-k$则不可修复，否则可以修复.然而，在 GRC 码中，知道失效块数目并不能直接判断是否可修复.在失效数为 6 时，GRC 码可修复图 3 所示情况，却无法修复图 4 所示情况，图中深色块表示失效块.

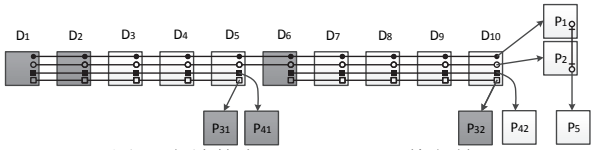

图 3 失效数为 6，GRC 码可修复情况

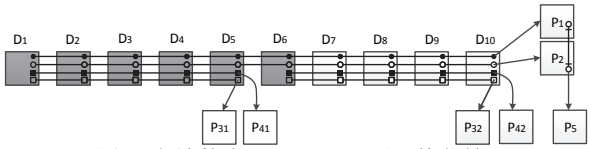

图 4 失效数为 6，GRC 码不可修复情况

相比于(n,k)MDS 码修复一个块所要读取的块数目确定为k，GRC 码中修复一个块有多种选择.以(14,10)MDS 码和(17,10)GRC 码为例，若 D_1 失效，(14,10)MDS 码需要 10 个块修复，而 GRC 码既可以用 $(D_2 \sim D_9, P_1)$ 10 个块修复，也可以用 $(D_2 \sim D_5, P_{31})$5 个块修复.后者的解码公式为 $D_1 = g_{31}^{-1}(P_{31} - g_{32}D_2 - g_{33}D_3 - g_{34}D_4 - g_{35}D_5)$，使修复成本降低 50%.若$P_1$失效， (14,10)MDS 码需要 10 个块修复，而 GRC 码用(P_2, P_5)2 个块修复，解码公式$P_1 = P_2 + P_5$，使修复成本降低 80%.

由上面的例子可见，GRC 码中对于同一个块的修复通常有多种选择，而且每修复好一个块，修复条件就会发生变化，所以修复过程不是一步完成的，而是一个多阶段选择的过程.当一个块的解码操作有多种选择时，研究如何解码才能够使得整个修复过程传输的数据量最小且能最大限度地修复失效块具有重要实际意义.

根据 GRC 码的特征，本文提出了基于贪心策略的解码算法 GSBD(**G**reedy **S**trategy **B**ased **D**ecode Algorithm).GSBD 执行两类修复：组修复（或称局部修复）和全局修复.组修复是指利用和失效块在同一组内的块即可完成的修复，由于组内的块数较少，所以组修复需要传输的块数较少，修复成本较低；全局修复是指需传输数目为k个块才能完成的修复，修复成本高.为了降低修复成本，数据修复应遵循以下原则：修复过程应尽量选择组内修复，只有在组内无法完成修复时再全局修复.全局修复一个块后再次判断是否可以进行组内修复，若可以则立即转到组内修复.

根据以上原则，解码算法 GSBD 的解码过程如下：第一阶段，组修复.对于每个组，如果可用块数不少于失效块数，则修复所有失效块并标记为可用.若失效块全部能够组内修复，则解码成功，程序结束.第二阶段，全局修复.若存在失效块不能在组内修复，则转入全局修复，固定$i(m_0 < i \le m)$，如果所有的组编码块P_{il}被标记为可用，则标记P_i为可用.在全局层面，如果可用块数不少于失效块数，则修复一个失效块，并将修复好的块标记为可用，转入第一阶段.否则，解码失败，程序结束.GSBD 的具体过程如算法 1 所示，第一阶段对应 4~9 行，第二阶段对应 10~15 行.

解码算法 GMCD 紧密契合了 GRC 码的编码特性，使得 GRC 码在修复过程中能够首先选择组内修复，以保证数据传输量最少.对于无法在组内完成修复的失效块，先进行全局修复，一旦满足组内修复的条件，就转到组内修复，从而使得总的修复成本最低.

算法 1. 基于贪心策略的解码算法 GSBD

输入：

erasedLocations[]: 失效块位置数组；
locationsToRead[]: 要读取的块在条带中的位置数组；
readBufs[]: 读数据缓存数组；

输出：

writeBufs[]: 写数据缓存数组；

过程：

```
01. Initialize erasedLocations[] , locationsToRead[] ,
    readBufs[] based on system input;
02. unrec_num = erasedLocation[].length;
03. FOR (i = 1 to unrec_num)
04.   IF canBeRecInGp(erasedLocations[]) == TRUE
05.     decodeInGroup();
06.     Store the results in writeBufs[];
07.     Update erasedLocation[];
08.     IF erasedLocation[].length == 0
09.       RETURN TRUE;
10.   IF(decodeInGlo()==FALSE)
11.     RETURN FALSE;
12.   ELSE
13.     Store the results in writeBufs[];
14.     Update erasedLocation[];
15.  BREAK;
16. END FOR
```

4 GRC 码实现

本文在 HDFS-RAID[30]中实现了 GRC 码，GRC 码以插件方式加入到 HDFS-RAID.为了区分应用 RS 码的 HDFS-RAID，将实现了 GRC 码的分布容错存储系统称为 HDFS-GRC.

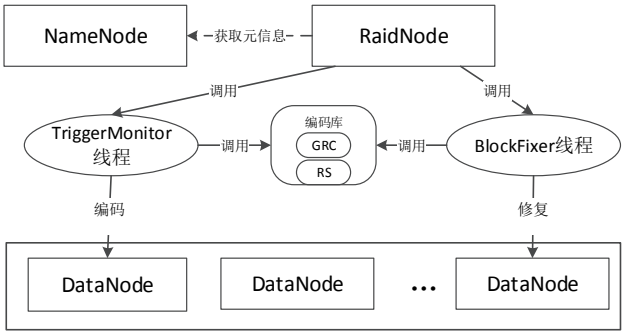

图 5. HDFS-GRC 体系结构示意图

如图 5 所示，运行于 RaidNode 上的后台线程 TriggerMonitor 一旦检测到文件满足编码条件(根据配置文件设置的参数而定)就启动纠删码编码过程.(n,k)GRC 码为文件的每k个数据块生成$n-k$个编码块.由于文件尺寸原因，最后一个条带可能不足k个块，不完整条带用 0 补成完整条带.编码完成后将块根据 Hadoop 配置的块放置策略散布到集群上，为避免同条带中的块放到同个节点，所有块随机放置到 DataNode.

当有失效文件被检测到，RaidNode 的后台线程 BlockFixer 启动修复. 当 BlockFixer 检测到失效块，首先根据 GRC 码的结构确定需要读取的块，然后进行数据的读取，进入修复过程.

5 实验结果与分析

本文在实际分布式存储系统中对 GRC 码进行了多方面的测试，并将其与现有纠删码进行了比较.

5.1 实验环境和设计

实验采用的分布式存储系统集群由 30 个节点组成，其中一个作为 NameNode 节点，其余 29 个节点作为 DataNode 节点，NameNode 节点同时运行 RaidNode 进程. 每个节点配置 2 个 8 核 Intel Xeon E5-2640 2.5GHz 处理器，48G RAM，2T 硬盘和 1GB/s 以太网卡.所有节点运行 64 位 CentOS6.3 系统和 JDK1.6.0_37.

实验所用数据为 200 个 640M 文件，数据块大小保持 HDFS 默认块的 64M，每个文件在(17,10)GRC 码、(16,10)BPC 码、(15,10)LRC 码和(14,10)RS 码中分别产生 17、16、15 和 14 个完全块组成的条带.

实验对比容错度均为 4 的(17,10)GRC 码、(16,10)BPC 码、(15,10)LRC 码和(14,10)RS 码. 其中(17,10)GRC 码为本文提出的分组修复码的一种具体形式，其结构如图 2 所示. BPC 码的全称是 Basic Pyramid Codes，是 Pyramid[20]码一种主要形式. BPC 码通过增加局部冗余来降低修复成本.(16,10)BPC 码将 10 个数据块分成两组，在每组计算产生 2 个局部编码块，为整个数据条带计算产生 2 个全局编码块. LRC 码也是一种通过增加局部冗余来降低修复成本的纠删码.(15,10)LRC 码将数据条带分成两组，每组产生一个组编码块，为整个数据条带计算产生 3 个全局编码块. RS 码是典型的 MDS 码，(14,10)RS 码是 Facebook 在其存储集群中所用的纠删码[12]，其结构如图 1 所示.

5.2 实验对比指标和方法

实验对比指标有容错能力、修复成本和修复时间. 容错能力是衡量纠删码性能的一个重要指标. 修复成本反应了纠删码的数据修复操作对网络带宽的占用情况. 修复时间衡量修复算法的实时能力.

为了测量纠删码的容错能力，实验应用蒙特卡洛法编写模拟程序，随机生成失效节点，将模拟失效次数设为 *n*，当 *n* 不断增大时，修复率无限逼近于某个值，该值为纠删码在 *r* 个节点失效时的修复率. 修复率表示纠删码在固定数目节点失效时的修复能力，不同失效节点数时的修复率综合反映了纠删码的容错能力. 为了测量纠删码的修复成本和修复时间，实验采用随机关闭节点的方式模拟实际应用中的节点故障规律.随机关闭若干 DataNode 节点，将修复读取的数据量测量结果输出到日志文件，查看日志文件以计算修复成本；记录第一个块开始修复的时间至最后一个块修复完毕的时间差作为修复时间.

5.3 结果比较与分析

5.3.1 容错能力

图 6 给出了(17,10)GRC 码 、(16,10)BPC 码、(15,10)LRC 码和(14,10)RS 码的容错能力对比，横轴表示失效节点个数，纵轴表示相应修复率(采用百分比表示法). 在失效节点数为 1，2，3，4 时，四种纠删码均能完全修复，当大于 4 个节点失效，(14,10)RS 码的数据丢失，而(17,10)GRC 码、(16,10)BPC 码和(15,10)LRC 码在很高概率上容 5 个节点失效情况，(17,10)GRC 码、(16,10)BPC 码在较高概率上容 6 个节点失效，只有(17,10)GRC 码在概率上容 7 个节点失效. 为了降低修复成本，GRC 码相对于 RS 码、LRC 码和 BPC 码增加了少量额外冗余数据，这些额外冗余数据同时也提高了 GRC 码的容错能力. 因此，(17,10)GRC 码的容错能力强于(16,10)BPC 码、(15,10)LRC 码和(14,10)RS 码.

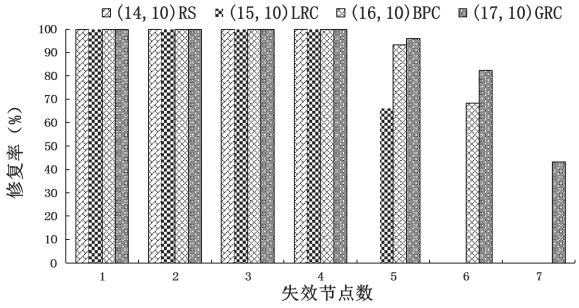

图 6. 容错能力对比

5.3.2 修复成本

图 7 为(17,10)GRC 码、(16,10)BPC 码、(15,10)LRC 码和(14,10)RS 码的修复成本对比，图中的横坐标表示失效节点数和失效块数，纵坐标表示修复成本. (17,10)GRC 码相比于(14,10)RS 码修复成本降低了 50%-55%，相比于(15,10)LRC 码修复成本降低了 35%-45%，相比于(16,10)BPC 码修复成本降低了 15%-25%.从图 8 看出 GRC 码、BPC 码、LRC 码和 RS 码的修复成本都随失效块增多呈线性增长，斜率表示平均修复成本，(17,10)GRC 码、(16,10)BPC 码、(15,10)LRC 码和(14,10)RS 码的平均修复成本分别为 0.29GB(4.7 个块) 、0.37GB(5.8 个块)、0.45GB(7 个块)和 0.62GB(9.8 个块).

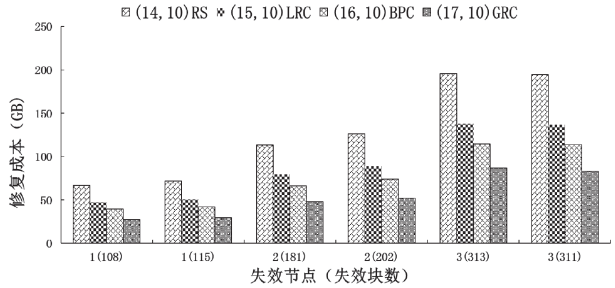

图 7. 修复成本对比图

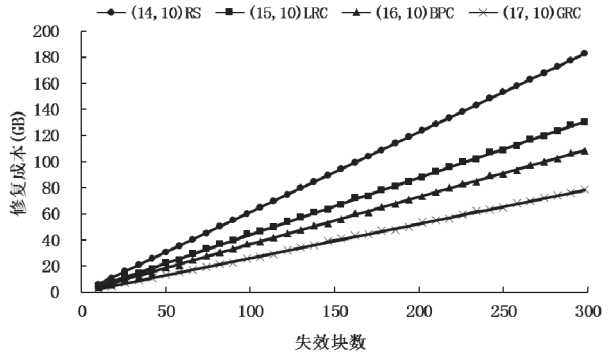

图 8. 修复成本随失效块数增长趋势

5.3.3 修复时间

图 9 为(17,10)GRC 码、(16,10)BPC 码、(15,10)LRC 码和(14,10)RS 码的修复时间对比.横轴表示失效节点数和失效块数，纵轴表示修复时长. (17,10)GRC 码相比于(14,10)RS 码修复速度快 75%-90%，相比于(15,10)LRC 码修复速度快 40%-50%，相比(16,10)BPC 码修复速度快 20%-25%. 图 10 为修复时间随失效块数增长趋势图，随着失效块数的增长，GRC 码、BPC 码、LRC 码和 RS 码的修复时间均呈线性增长. 斜率表示修复单块时间，(17,10)GRC 码、(16,10)BPC 码、(15,10)LRC 码和(14,10)RS 码修复单块时间分别为 7.2 秒、9 秒、10.8 秒和 13.6 秒.

修复时间会影响数据的可靠性和可用性.在大规模存储系统中，如果不能快速地修复失效的数据块，那么在修复过程中容易发生数据块再次失效，导致数据丢失.修复时间的另一个重要影响是在数据可用性上，修复速度越快则可用性越好.在 3 副本系统中，若一个数据块失效，那么此块的副本可以立即被访问到，但是对于 GRC 码、BPC 码、LRC 码和 RS 码，则需要修复工作.因为 GRC 码完成修复工作更快，所以相对于 BPC 码、LRC 码和 RS 码，GRC 码使得纠删码存储系统的数据可用性更高.

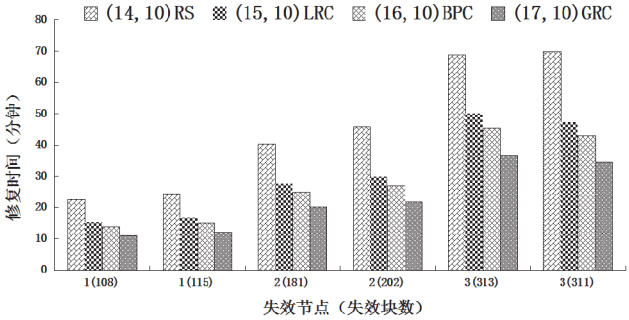

图 9. 修复时间对比图

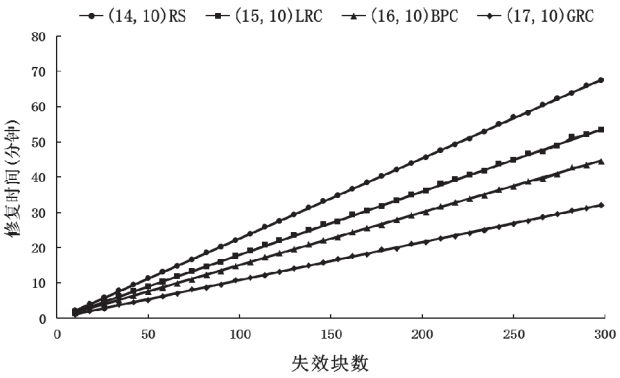

图 10. 修复时间随失效块数增长趋势

5.3.4 不同参数 GRC 码的性能比较

下面比较不同参数 GRC 码的性能. 基于 GRC 码的编码公式，将$n=16$，$k=10$，$m_0=3$，$m_1=1$的 GRC 码称为 (16,10)GRC 码;将$n=18$，$k=10$，$m_0=1$，$m_1=3$的 GRC 码称做(18,10)GRC 码，根据定理 1，(17,10)GRC 码和(16,10)GRC 码、(18,10)GRC 码具有相同的容错度. 下面给出它们的性能对比图. 图 11 显示(17,10)GRC 码相比于(16,10)GRC 码修复成本有较明显的下降，这是因为(17,10)GRC 码能保证在组内修复两块失效. 而(18,10)GRC 码相比于(17,10)GRC 码修复成本并无明显地下降，这是因为三块失效的概率相对于两块失效要小的多. 图 12 的显示了(17,10)GRC 码相比于(16,10)GRC 码修复速度有较明显的提升，而(18,10)GRC 码相比于(17,10)GRC 码修复速度并没有较明显的下降，与修复成本指标的对比结果相似. 原因是修复速度和修复成本成正比，修复成本越低意味着修复数据传输量越少，从而修复速度就越快. 实验结果表明，(17,10)GRC 码用较少的额外存储空间显著降低了纠删码的修复成本.

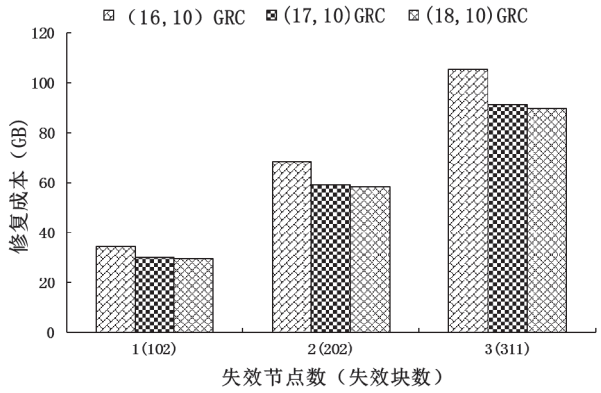

图 11. 不同参数 GRC 码的修复成本对比

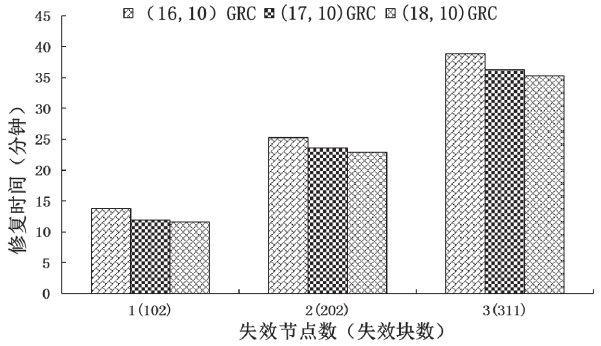

图 12. 不同参数 GRC 码的修复时间对比

6 总结

纠删码技术因其极低的额外存储空间消耗而在现代存储系统中变得越来越重要, 然而过高的修复成本阻碍了纠删码在实际系统中的应用和推广, 因此，低成本快速的失效块修复对提高存储系统的性能有着重要的意义.

本文提出了分组修复码(Group Repairable Codes，简称 GRC)，GRC 码大幅减少了修复对网络带宽和磁盘 I/O 资源的占用，且显著提高了修复速度.本文实验结果表明，与 RS 码相比，GRC 码将修复网络带宽和磁盘 I/O 降低 50%-55%，修复速度提高 75%-90%，仅需增加 21%存储空间；与 LRC 码相比，GRC 码将修复网络带宽和磁盘 I/O 降低 35%-45%，修复速度提高 40%-50%，仅需增加 13%存储空间；与 BPC 码相比，GRC 码将修复网络带宽和磁盘 I/O 降低 15%-25%，修复速度提高 20%-25%，仅需增加 6%存储空间.

GRC 码的一个重要应用是存档系统，当条带很大时(50 到 100 个块)，GRC 码以极低的存储开销大幅降低修复成本和提供高容错能力.而 RS 码修复占用带宽量是随着条带长度呈线性增长的.

(17,10)GRC 码和(18,10)GRC 码适用于延迟修复的情形，即修复的间隔时间较长，或达到一定数目的失效块数才开始修复.如果修复间隔较短，可选择额外存储空间较低的(16,10)GRC 码，所以 GRC 码是一种灵活的纠删码，具体的参数，可根据实际系统的需求选择.

参 考 文 献

[1] Armbrust M, Fox A, Griffith R, et al. A view of cloud computing[J]. Communications of the ACM, 2010, 53(4):50–58.

[2] WANG YiJie, SUN WeiDong, ZHOU Song, PEI XiaoQiang, LI XiaoYong. Key technologies of distributed storage for cloud computing. Journal of Software , 2012,23(4):962 − 986.
(王意洁，孙伟东，周松，等. 云计算环境下的分布存储关键技术[J]. 软件学报，2012，23(4)：962−986.)

[3] Wang Yijie, Li Sijun. Research and performance evaluation of data replication technology in distributed storage systems[J]. Int Journal of Computers and Mathematics with Applications, 2006, 51(11):1625–1632.

[4] 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.)

[5] Ghemawat S, Gobioff H, Leung S T. The Google file system[C]//ACM SIGOPS Operating Systems Review. ACM, 2003, 37(5): 29-43.

[6] SHVACHKO K, KUANG H, RADIA S, et al. The Hadoop Distributed File System[C]//Proceedings of 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST). Los Alamitos, CA, USA: IEEE Computer Society, 2010:1–10.

[7] THUSOO A, SHAO, Z et al. Data Warehousing and Analytics Infrastructure at Facebook[C]//Proceedings of the 2010 ACM SIGMOD International Conference on Management of data, ACM, 2010, 1013-1020.

[8] Weidong S, Yijie W, Xiaoqiang P. Tree-structured parallel regeneration for multiple data losses in distributed storage systems based on erasure codes[J]. Communications, China, 2013, 10(4): 113-125.

[9] LIN W K, CHIU D M, LEE Y B. Erasure Code Replication Revisited[C]//Proceedings of the Fourth International Conference on Peer-to-Peer Computing. 2004. Washington, DC, USA: IEEE Computer Society, P2P '04.

[10] WEATHERSPOON H, KUBIATOWICZ J. Erasure Coding Vs. Replication: A Quantitative Comparison[C]//Revised Papers from the First International Workshop on Peer-to-Peer Systems. 2002. London, UK: Springer-Verlag, IPTPS '01.

[11] Wicker, Stephen B., and Vijay K. Bhargava, eds. Reed-Solomon codes and their applications[M]. John Wiley & Sons, 1999.

[12] Sathiamoorthy M, Asteris M, Papailiopoulos D, et al. XORing elephants: Novel erasure codes for big data[C]//Proceedings of the VLDB Endowment. VLDB Endowment, 2013, 6(5): 325-336.

[13] Fikes A. Storage architecture and challenges[J]. Talk at the Google Faculty Summit, 2010.

[14] Mckusick K, Quinlan S. GFS: evolution on fast-forward[J]. Communications of the ACM, 2010, 53(3):42–49.

[15] Huang Cheng, Simitci H, Xu Yikang, et al. Erasure coding in windows azure storage[C]//Proc of the 2012 USENIX Conf on Annual Technical Conf. Berkeley: USENIX Association, 2012.

[16] Sun W, Wang Y, Fu Y, et al. A Discrete Data Dividing Approach for Erasure-Code-Based Storage Applications[C]//Service Oriented System Engineering (SOSE), 2014 IEEE 8th International Symposium on. IEEE, 2014: 308-313.

[17] Chowdhury M, Zaharia M, Ma J, et al. Managing data transfers in computer clusters with orchestra[C]//ACM SIGCOMM Computer Communication Review. ACM, 2011, 41(4): 98-109.

[18] Dimakis A G, Ramchandran K, Wu Y, et al. A survey on network codes for distributed storage[J]. Proceedings of the IEEE, 2011, 99(3): 476-489.

[19] JIEKAK S, KERMARREC A M, LE SCOUARNEC N, et al. Regenerating Codes: A System Perspective[C]//Proceedings of the 2012 IEEE 31st Symposium on Reliable Distributed Systems(SRDS). 2012:436–441.

[20] Huang C, Chen M, Li J. Pyramid codes: Flexible schemes to trade space for access efficiency in reliable data storage systems[J]. ACM Transactions on Storage (TOS), 2013, 9(1): 3.

[21] Bhagwan R, Tati K, Cheng Y, et al. Total Recall: System Support for Automated Availability Management[C]//NSDI. 2004, 4: 25-25.

[22] Rashmi K V, Shah N B, Kumar P V. Optimal exact-regenerating codes for distributed storage at the MSR and MBR points via a product-matrix construction[J]. IEEE Transactions on Information Theory, 2011, 57(8): 5227-5239.

[23] Shah N B, Rashmi K V, Kumar P V, et al. Interference alignment in regenerating codes for distributed storage: necessity and code constructions[J]. IEEE Transactions on Information Theory, 2012,

58(4): 2134-2158.
[24] Gopalan P, Huang C, Simitci H, et al. On the locality of codeword symbols[J]. IEEE Transactions on Information Theory, 2012, 58(11): 6925-6934.
[25] Khan O, Burns R, Plank J, et al. In search of I/O-optimal recovery from disk failures[C]//Proceedings of the 3rd USENIX conference on Hot topics in storage and file systems. USENIX Association, 2011: 6-6.
[26] Zhou Song, Wang Yijie. EXPyramid: an array-based flexible coding scheme with high fault-tolerance and low recovery-overhead[J]. Journal of Computer Research and Development, 2011, 48:30–36(in Chinese)
(周松, 王意洁. EXPyramid:一种灵活的基于阵列结构的高容错低修复成本编码方案[J]. 计算机研究与发展, 2011, 48:30-36.)
[27] Hafner J L, Deenadhayalan V, Rao K K, et al. Matrix Methods for Lost Data Reconstruction in Erasure Codes[C]//FAST. 2005, 5: 15-30.
[28] Hafner J L, Deenadhayalan V, Kanungo T, et al. Performance metrics for erasure codes in storage systems[J]. IBM Res. Rep. RJ, 2004, 10321.
[29] MacWilliams F J, Sloane N J A. The theory of error-correcting codes[M]. Elsevier, 1977.
[30] Borthakur D, Schmidt R, Vadali R, et al. Hdfs raid[C]//Hadoop User Group Meeting. 2010.
[31] Plank J S. The RAID-6 Liberation codes [C]. In Proceedings of the 6th Usenix Conference on File and Storage Technologies(FAST). 2008: 97–110.

林　轩，男，1990 年生，硕士研究生，主要研究方向为分布式容错存储系统，纠删码.

王意洁，女，1971 年生，教授，博士生导师，主要研究方向为网络计算，海量数据处理，并行与分布式处理.

裴晓强，男，1986 年生，博士研究生，主要研究方向为分布式存储系统，纠删码，数据放置.

许方亮，男，1989 年生，博士研究生，主要研究方向为分布式容错存储系统，纠删码.

符永铨，男，1983 年生，助理研究员，主要研究方向为云计算体系结构，云存储，网络测量.