异构对等分布式存储中的 DHitchhiker 码

胡金平,李贵洋,周 悦,李 慧,江小玉,韩鸿宇
(四川师范大学计算机科学学院,四川成都 610101)

摘 要: 具有高可用性和安全性的去中心化存储因其应用的相同参数低码率 RS(Reed-Solomon codes)码存在修复带宽较高的问题.对此,提出一种基于可信度的低修复带宽 DHitchhiker 码.首先将 Hitchhiker 码的第一子条带中的数据节点和部分校验节点捎带在余下的校验节点的第二个子条带上;接着将节点分类,让高可信节点存储余下的校验节点,低可信节点存储数据节点和部分校验节点,并让不同类型的节点采用不同的修复策略;最后理论结合实验证明,在修复低可信节点时,DHitchhiker 可降低约 25% 的修复带宽;在整体上,未分类存放的 DHitchhiker 码可降低约 0.5% 的修复带宽,基于可信度的 DHitchhiker 码可降低约 1% 的修复带宽和 2.5% ~3.3% 的修复时间.

关键词: MDS 码; RS 码; 可信度; 去中心化存储; Hitchhiker 码
中图分类号: TP333 **文献标识码:** A **文章编号:** 0372-2112 (2021)06-1151-08
电子学报 URL: http://www.ejournal.org.cn **DOI:** 10.12263/DZXB.20200288

DHitchhiker Codes in Heterogeneous Peer-to-Peer Distributed Storage

HU Jin-ping, LI Gui-yang, ZHOU Yue, LI Hui, JIANG Xiao-yu, HAN Hong-yu
(*College of Computer Science, Sichuan Normal University, Chengdu, Sichuan* 610101, *China*)

Abstract: Decentralized storage with high availability and security has the problem of high repair bandwidth due to the same parameter low rate Reed-Solomon codes. Thus, a low repair bandwidth DHitchhiker based on credibility is proposed. Data nodes and some parity nodes in the first sub-stripe of Hitchhiker are piggybacked on the second sub-stripe of the remaining parity nodes. We classify the nodes, and let thehigh-credit nodes store the remaining parity nodes, the low-credit nodes store the data nodes and some parity nodes, and different types of nodes adopt different repair strategies. It is proved that the repair bandwidth can be reduced by about 25% at repairing low-credit nodes. And the repair bandwidth can be reduced by about 0.5% without classifying the nodes. What's more, the repair bandwidth can also be reduced by about 1% and the repair time can be reduced by about 2.5% ~ 3.3% based on the credibility of the nodes.

Key words: MDS codes; Reed-Solomon codes; credibility; decentralized storage; hitchhiker

1 引言

随着云存储的迅猛发展与大数据时代的来临,存储系统利用纠删码技术[1,2]替代了传统的多副本技术.在传统的分布式存储中,通常采用主/从模式,如 GFS[3]、HDFS[4]、OpenEC[5],存在单节点故障[6]、恶意篡改等安全性问题.区块链技术[7]的成熟让具有高可用性和安全性的去中心化存储成为了数据备份的存储趋势,如 Storj[8]、Sia[9]等.

去中心化存储利用低码率的 RS 码[10]来保证数据的可靠性.RS 是参数取值任意的 MDS[11,12]码,是目前存储系统中常用的纠删码,但其高昂修复成本成为其应用的瓶颈.为降低 RS 码的修复带宽,人们提出了 PB (Piggybacking)架构[13,14]、再生码[15~17]、置换码[18,19]等编码.再生码能达到理论上最优的修复带宽[17],但因构造复杂和条代数庞大限制难以实现;而 PB 架构下的 Hitchhiker 码[20]因较小条带数和较低修复带宽而被应用在了中心化存储中.它用 piggyback 的方式将数据节点的修复转换为部分 MDS 性质和部分 piggyback 方程,降低了修复带宽;校验节点的修复仍然是整体的 MDS 性质.

然而 Hitchhiker 码不能直接运用到去中心化存储中.一是两者环境差异:中心化内的节点高效、性能优、稳定且可信,常为单节点失效;而去中心化内节点参差

收稿日期:2020-03-23;修回日期:2020-08-01;责任编辑:梅志强
基金项目:国家自然科学基金(No.61701331)

不齐,常为多节点失效. 二是多节点失效屏蔽了编码低修复带宽的优势,低码率让半数以上节点的修复带宽不变. 对此,提出了去中心化存储中基于可信度的DHitchhiker码:通过将第一个子条带中的数据节点和部分校验节点以捎带的形式存放在第二个子条带余下的校验上、并按分类模型采用不同数据存储的方式,达到降低修复带宽和时间的目的. 结果表明具有MDS性质的双条带DHitchhiker可降低捎带数据节点约25%、整体上约0.5%的修复带宽. 基于可信度的DHitchhiker可降低约1%的修复带宽和2.5%~3.3%的修复时间.

2 相关理论基础

2.1 相关术语

为了方便叙述,将文中的参数和相关术语描述分别成罗列表1和表2.

表1 参数列表

参数	描述	参数	描述
n	节点总数	k	原始数据块数
r	校验节点个数	τ	启动修复的阈值
p	$k+\tau$ 与 $r-\tau$ 的商	q	$k+\tau$ 与 $r-\tau$ 的余数
δ	总修复下载量	γ	修复带宽率

表2 相关术语描述

术语	描述
MDS性质	将原始 k 块数据编码后,得到 r 个校验,其中任意 k 个节点都能恢复全部数据
条带	即stripe,同一个编码区域的 n 个units集合,由 k 个原始数据和 r 个校验数据组成
子条带	即sub-stripe,将一个条带进行纵向划分后而得到的多个分列,其仍然是一个编码区域
捎带数据	第一个子条带捎带到校验节点的第二个子条带的数据对应的节点,共 $k+\tau$ 个
捎带校验	第一个子条带未参与捎带的校验节点,共 $r-\tau$ 个

2.2 RS码

一个(k,r)的RS码,在有限域$GF(2^m)$中,能纠正r个已知错误. 若编码数据向量$\boldsymbol{M}^{\mathrm{T}}=(m_1,m_2,\cdots,m_k)$、生成矩阵向量$\boldsymbol{G}^{\mathrm{T}}=[g_{j,i}]_{(0<i\leqslant n,0<j\leqslant k)}$和编码后的码字向量$\boldsymbol{W}^{\mathrm{T}}=(w_1,w_2,\cdots,w_n)$中的元素均在$GF(2^m)$中,$\boldsymbol{G}$的任意$n$行线性无关,编码:$\boldsymbol{W}^{\mathrm{T}}=\boldsymbol{GM}$.

解码时,因$\boldsymbol{G}$非奇异,所以$\boldsymbol{W}$中任意的k行都能恢复全部数据. 校验节点丢失后重新计算校验;数据节点丢失后就选择$\boldsymbol{W}$中任意k行$\boldsymbol{W}_{\mathrm{K}}^{\mathrm{T}}=\boldsymbol{G}_{\mathrm{K}}\boldsymbol{M}$,$\boldsymbol{W}_{\mathrm{K}}^{\mathrm{T}}$中大小为$k\times k$的矩阵$\boldsymbol{G}_{\mathrm{K}}$的逆元$\boldsymbol{G}_{\mathrm{K}}^{-1}$存在,左乘上逆元$\boldsymbol{G}_{\mathrm{K}}^{-1}$便可恢复失效节点数据.

如图1所示. 分布式系统通常会选择合适的(k,r)作为编码的参数. 将待存储的文件切分为k个相同大小的数据块,并存储到k个节点中;然后用RS码编码,得到r个校验数据,并将其存储到其他r个节点中. 在解码时,系统将选择一个数据收集器(DC)去下载k个未丢失的节点,解码出失效节点的数据. DC的瓶颈,可参考Lee P P等人的文献[21,22].

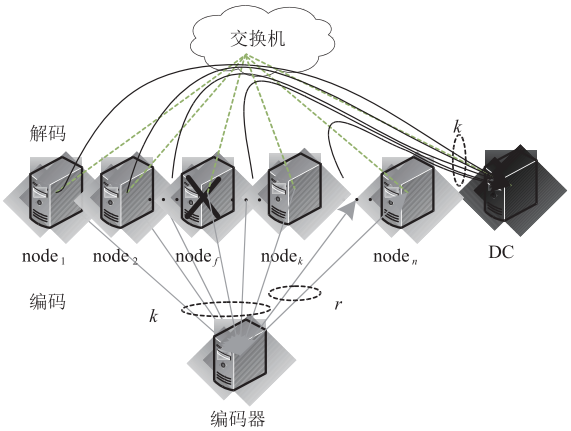

图1 分布式存储中RS码的应用

2.3 Hitchhiker码

Hitchhiker码有XOR、XOR+和nonXOR三种构造方式. XOR与XOR+用异或提高了降阶读和失效修复的速度. nonXOR能构建在任何RS码之上. 如图2所示,四种编码修复单节点失效的情况如下:校验节点unit_{11}丢失,它们均需要下载$\{a_i\mid_{i=1}^{10}\}$和$\{b_i\mid_{i=1}^{10}\}$. 数据节点unit_8丢失,RS码下载任意10个节点的数据,用MDS性质恢复unit_8. 而Hitchhiker码下载$\{b_1,\cdots,b_7,b_9,b_{10},f_1(\boldsymbol{b})\}$用部分MDS性质修复$b_8$. XOR下载$\{f_4(\boldsymbol{a})\oplus a_7\oplus a_8\oplus a_9\oplus a_{10},a_7,a_9,a_{10}\}$;而XOR+和nonXOR则下载$\{f_4(\boldsymbol{a})\oplus a_7\oplus a_8\oplus a_9,a_7,a_9\}$,均用piggyback方程修复$a_8$. XOR下载量是14,而后两者是13. 原因在于后两者用了可逆变换(横向减法)存储a_{10}.

2.4 修复策略

在去中心化存储中,系统的可用性会随着瞬态故障不断的变化,因此必须给定动态的冗余机制,如Storj中的(K,M,O,N). 纠删码虽能降低存储成本和容忍潜在的瞬时故障,但其代价是解码时间和修复带宽. 减少解码时间的方式有改进编码(异或)、修复流水线等方式. 降低修复带宽的方式有改进编码、改变修复策略等方式. 在去中心化存储中未降低解码时间;但采用了著名的TotalRecall分布式存储系统[23]来降低修复带宽,用集中式的延迟修复方式同时修复多失效的节点. 如Storj和Sia分别采用RS(20,20),RS(10,20),丢失5个节点后集中修复.

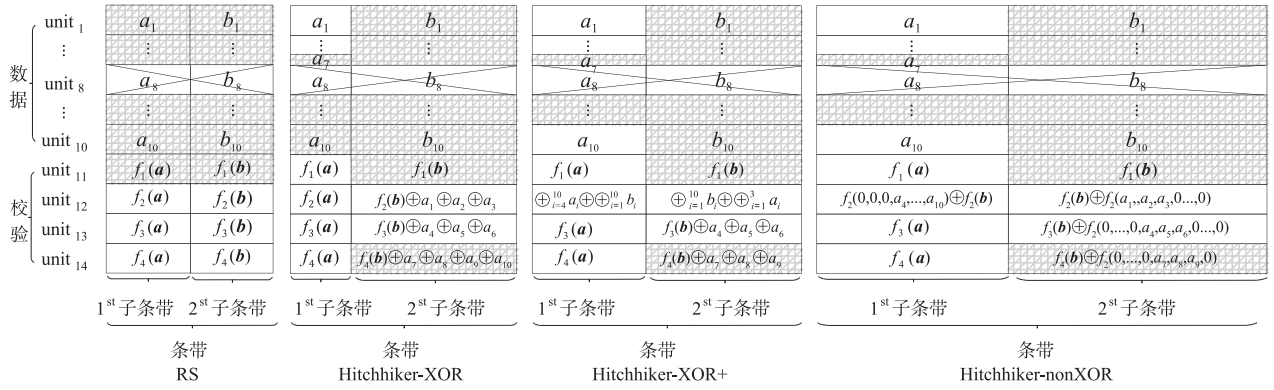

图2　Hitchhiker(10,4)与RS(10,4)编码的结构

3　DHitchhiker 码

3.1　节点类型

3.1.1　节点分类

节点分类旨在让每个节点处于一个合理的可信度下. 分类标准有历史数据、网络带宽和提供方三方面. 历史数据可知节点在线时长和上下线的频率、受奖励和惩罚的次数及轻重、是否发起过拜占庭攻击等,用符号 $H=f(h_1,h_2,\cdots,h_n)$ 表示. 网络带宽包括最大能够供使用的带宽、已被占用的带宽、当前空闲的带宽等,用符号 $B=f(b_1,b_2,\cdots,b_n)$ 表示. 节点的提供方包括提供方的可信度,在提供方的等级等,用符号 $P=f(p_1,p_2,\cdots,p_n)$ 表示. 称 $C=F(H,B,P)$ 为分类模型. 三个标准同时作用,以量化和权重的方式计算每一个节点的得分、并归一,据此将节点分为高可信节点和低可信节点. 如图 3 所示.

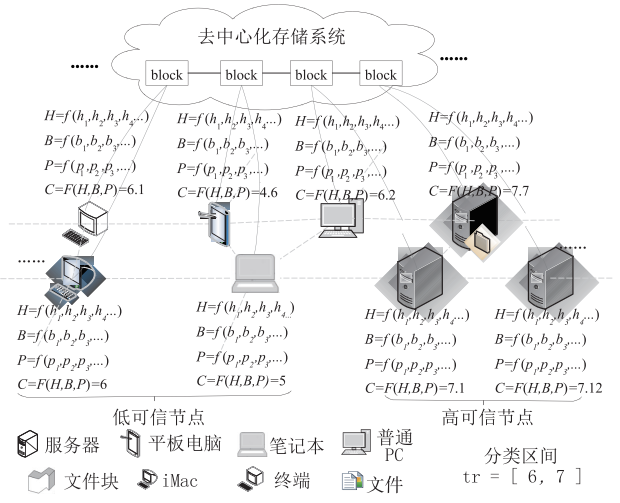

图3　节点分类的基本准则图

3.1.2　可信节点数量

RS 最多容忍 $n-k$ 个节点失效,只要失效节点的数量不少于 $n-k+1$,数据将会永久失效. 根据节点失效的随机性可知,丢失的 $n-k+1$ 个节点随机分布在 n 个节点中. 那么穷举 $n-k+1$ 个节点的失效率,其和与编码的失效率相等. 若高可信节点和低可信节点的失效概率分别为 λ_h、λ_l,个数分别有 $\tilde{n}$、$n-\tilde{n}$ 个. 那么 $i(i\in[0,n-k+1])$ 个高可信节点丢失的失效率为 $R_h^L=C_n^i\lambda_h^i(1-\lambda_h)^{\tilde{n}-i}$. $n-k+1-i$ 个低可信节点丢失的失效率为 $R_l^L=C_{n-\tilde{n}}^{n-k+1-i}\lambda_l^{n-k+1-i}(1-\lambda_l^{k+i-\tilde{n}-1})$. 由于高可信节点和低可信节点同时丢失,总的失效率 $R_i^L=R_h^L\times R_l^L$. 所有失效的情况之和与失效率相等,即 $\sum_{i=0}^{n-k+1}R_i^L=R^L$,便可知系统中高可信节点的数量.

3.2　编码与解码

3.2.1　编码过程

将 Hitchhiker 应用到去中心化存储应根据实际场景保留 τ 个校验的第二个子条带不捎带数据,以保证编码的 MDS 性质. 得到新的 DHitchhiker 的编码模型 DHitchhiker(k,r,τ). 编码有 RS 编码、保留校验、捎带 3 步. 均以$(k=10,r=20,\tau=5)$为例.

第 1 步,两个子条带利用相同的编码矩阵按照 RS 的编码方式进行编码,存放在 n 个不同的节点中. 此例子将 10 个原始数据编码得到 20 个的校验,存放在 30 个节点中. 如图 4 的 step1 所示.

第 2 步,保留 τ 个校验节点不捎带数据. 将这 τ 个校验标记,并将它们的第一个子条带的数据看成原始数据参与到第三步的捎带,称这些参与捎带的 $k+\tau$ 个节点为捎带数据节点. 此例子将选出的 5 个校验节点不捎带数据,并将它们和 10 个原始数据节点当作捎带数据节点,参与第三步的捎带. 如图 4 的 step2 所示,A 中的数据参与到第三步的捎带,并将 $\text{unit}_1\sim\text{unit}_{15}$ 称为捎带数据节点.

第 3 步,将捎带数据节点均匀得存放在剩下的校验节点中,称剩下的校验节点为捎带校验节点. 均分法则为两步:先按式(1)计算每个捎带校验最少需捎带的数据的量 p 和余下的捎带数据的量 q.

$$\begin{cases}p=\lfloor(k+\tau)/(r-\tau)\rfloor\\q=(k+\tau)\bmod(r-\tau)\end{cases}\tag{1}$$

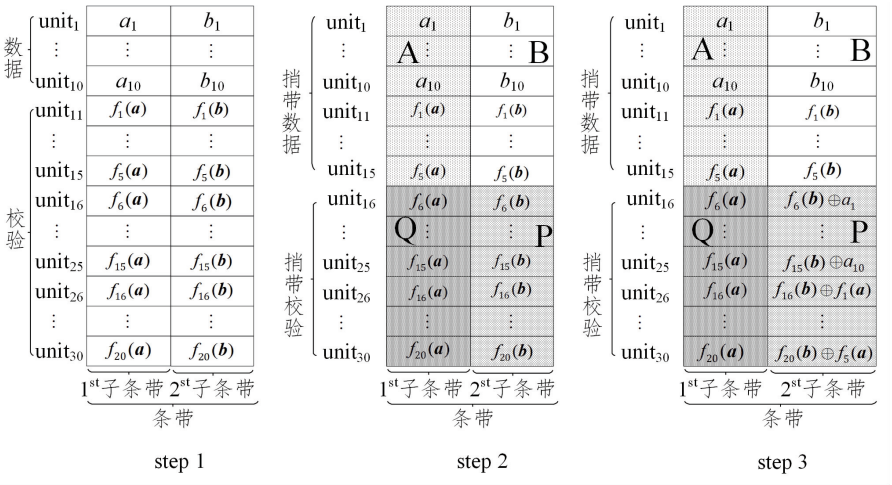

图4 去中心化存储中的双条带DHitchhiker编码示意图

让每个捎带校验先捎带 p 个捎带数据. 由于横向减法只能对数据节点有效,在选用捎带数据时要优先选用保留的 τ 个校验. 接着根据式(2)(推导见 4.2.2)判断余下的 q 个捎带数据的存放.

$$q(p+\tau+3-r)>0 \tag{2}$$

若满足式(2),就用横向减法将 q 个捎带数据捎带在第一个子条带上. 否则捎带在后 q 个捎带校验上,让每个捎带校验再多捎带一个数据. 用符号 U_i 记作第 i 个捎带校验捎带的捎带数据,即 $U_i=\{a_{x_1},a_{x_2},\cdots,a_{x_\alpha},f_{x_1}(\boldsymbol{a})\cdots,f_{x_\alpha}(\boldsymbol{a})\}$. 当 $i\in\{1,2,\cdots,r-\tau-q\}$ 时,$|U_i|=p$;当 $i\in\{r-\tau-q+1,r-\tau\}$ 时,若 $q\neq0$,$|U_i|=p+1$;若 $q=0$,$|U_i|=p$. 按公式计算,例子中的 $p=1,q=0$,且不满足式(2),所以 A 中的 15 个数据刚好以一对一的形式捎带在 P 中,即 $|U_i|=1,i\in[1,15]$,如图 4 的 step3 所示.

3.2.2 解码过程

解码时需根据节点失效的位置来确定帮助数据. 如算法 1 所示. 用符号 L 记作丢失节点的集合.

算法 1 DHitchhiker 的解码算法(多节点失效)

Input:失效节点集合 $L=\{\text{unit}_{x_1},\text{unit}_{x_2},\cdots,\text{unit}_{x_\tau}\}$,捎带校验捎带的数据的集合 $U_{i=k+\tau+1}^{k+r}$.
Output:失效节点的数据 $L'=\{(a_{x_1},b_{x_1}),\cdots,(f_{x_1}(\boldsymbol{a}),f_{x_1}(\boldsymbol{b})),\cdots\}$.
Step1:判断丢失节点的类型,isAllData = true.
  For $i=0$; $i<\tau$; i++ then
    If $(k+\tau)<L_i.x_i\leqslant(k+r)$ then
      isAllData = false;break;
Step2:全是捎带数据即 isAllData = true 时,S,isOne = true.
  For $i=0$; $i<k+r$; i++ then
    If U_i.size()! =1 then isOne = false;break;
  If isOne = =false then
    If $\exists(S\subseteq L,S\neq\varnothing,|S|>1)$ 使 $S\not\subset U_{i=k+\tau+1}^{k+r}$ then
      HitchhikerMultDecoder();
    Else HitchhikerMultMDSDecoder();
  Else HitchhikerMultDecoder();
Step3:不全是捎带数据即 isAllData = false 时.
  If $\forall(x_i\in[r+\tau+1,n])$ then MDSDecoder();
  Else HitchhikerMultMDSDecoder();

算法 1 将丢失节点分为全是捎带数据和不全是捎带数据两类. 对于全是捎带数据:当 $|U_i|=1$ 时,用算法 2 来解码. 当 $|U_i|>1$ 时,若 $\exists(S\subseteq L,S\neq\varnothing,|S|>1)$,满足 $S\subseteq U_i$ 时,即存在 L 的非空子集 $S(|S|>1)$ 为校验捎带数据集合 U_i 子集(存在两个及以上的失效节点将第一个子条带的数据存放在同一个捎带校验上)时,用算法 3 解码;反之,用算法 2 解码. 不全为捎带数据的解码方式有两类:一是丢失的全是捎带校验,重新计算. 二是丢失的不全是捎带校验. 先判断捎带数据对应的捎带校验是否丢失,若丢失,按算法 3 解码. 反之,按算法 2 解码. 捎带校验用算法 3 解码. 当然,数据的总下载量始终为 $2k$. 实际上,DHitchhiker 码的解码可分为捎带解码和非捎带解码,后者退化为 RS 码.

算法 2 HitchhikerMultDecoder()

Input:失效节点集合 $L=\{\text{unit}_{x_1},\text{unit}_{x_2},\cdots,\text{unit}_{x_\tau}\}$
Output:失效节点的数据 $L'=\{(a_{x_1},b_{x_1}),\cdots,(f_{x_1}(\boldsymbol{a}),f_{x_1}(\boldsymbol{b})),\cdots\}$.
Step1:利用 MDS 性质恢复第二个字条带的数据.
  下载 $\{\underbrace{\{b_1,\cdots,b_x,f_1(\boldsymbol{b}),\cdots,f_\tau(\boldsymbol{b})\}}_{\tau}\setminus\{\text{unit}_{x_1}:b,\cdots,\text{unit}_{x_\tau}:b\}\}$;
  $\underbrace{\{\text{unit}_{x_1}:b,\cdots,\text{unit}_{x_\tau}:b\}}_{\tau}$ = MDSDecoder();
Step2:利用捎带数据解码第一个子条带的数据.
  下载 $U_{i=k+\tau+1}^{k+r}$ 中除了 $\text{unit}_{x_1}:a,\cdots,\text{unit}_{x_\tau}:a$ 的部分
    For $i=0$; $i<\tau$; i++ then
      $a_{x_i}=f_i(\boldsymbol{b})\oplus f_o(U_{x_i})-U_{x_i}\setminus a_{x_i}$
Step3:返回 $L'=\{(a_{x_1},b_{x_1}),\cdots,(f_{x_1}(\boldsymbol{a}),f_{x_1}(\boldsymbol{b})),\cdots\}$.

算法 2 首先利用 MDS 性质得第二子条带数据,其

中 $\text{unit}_{x_1}:b$ 为 x_1 节点的第二个字条带的数据. 然后利用捎带得到第一子条带的数据.

算法 3　HitchhikerMultMDSDecoder( )

Input:失效节点集合 $L=\{\text{unit}_{x_1},\text{unit}_{x_2},\cdots,\text{unit}_{x_r}\}$

Output:失效节点的数据 $L'=\{(a_{x_1},b_{x_1}),\cdots,(f_{x_1}(\boldsymbol{a}),f_{x_1}(\boldsymbol{b})),\cdots\}$.

Step1:利用 MDS 性质恢复第一个子条带的数据.

下载 $\{\underbrace{\{a_1,\cdots,a_k,f_1(\boldsymbol{a}),\cdots,f_r(\boldsymbol{a})\}\backslash\{\text{unit}_{x_1}:a,\cdots,\text{unit}_{x_r}:a\}}_{k}\}$;

$\underbrace{\{\text{unit}_{x_1}:a,\cdots,\text{unit}_{x_r}:a\}}_{r}=\text{MDSDecoder}(\,)$

Step2:消去捎带校验节点的第二个子条带捎带的数据.

下载 $\{\underbrace{\{b_1,\cdots,b_k,f_1(\boldsymbol{b}),\cdots,f_r(\boldsymbol{b})\oplus f_r(\boldsymbol{a})\}\backslash\{\text{unit}_{x_1}:b,\cdots,\text{unit}_{x_r}:b\}}_{k}\}$;

利用第 step1 已得到的捎带数据消去捎带校验中的捎带数据 $\{\underbrace{f_{r+1}(\boldsymbol{b})\oplus a_1,\cdots,f_r(\boldsymbol{b})\oplus f_r(\boldsymbol{a})}_{x}\}$;

Step3:利用 MDS 性质恢复第二个子条带的数据.

$\underbrace{\{\text{unit}_{x_1}:b,\cdots,\text{unit}_{x_r}:b\}}_{r}=\text{MDSDecoder}(\,)$

Step4:返回 $L'=\{(a_{x_1},b_{x_1}),\cdots,(f_{x_1}(\boldsymbol{a}),f_{x_1}(\boldsymbol{b})),\cdots\}$.

算法 3 利用第一个子条带的 MDS 性质得到 L 中的 $\boldsymbol{a}$;消去第二个子条带中捎带的数据;利用第二个子条带的 MDS 性质解码. 图 5 为解码示意图.

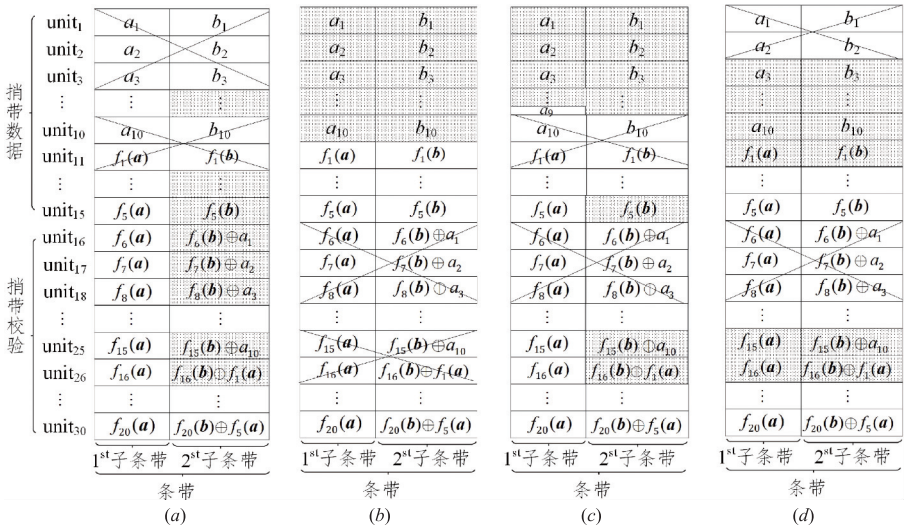

图5　去中心化存储中的Hitchhiker解码算法

3.3　数据存放策略

数据存放策略旨在在使用 DHitchhiker 码的基础上通过数据存放的位置进一步降低修复带宽。方法有二:降低修复下载量较大节点的失效频率;减少让失效频率较大节点的修复下载量。高可信节点失效率较低,低可信节点的失效率较高。让高可信节点存储修复下载量较大的捎带校验,满足一;让低可信节点存储修复下载量较小的捎带数据,满足二。同时满足,以减少总修复带宽。如图 6 所示.

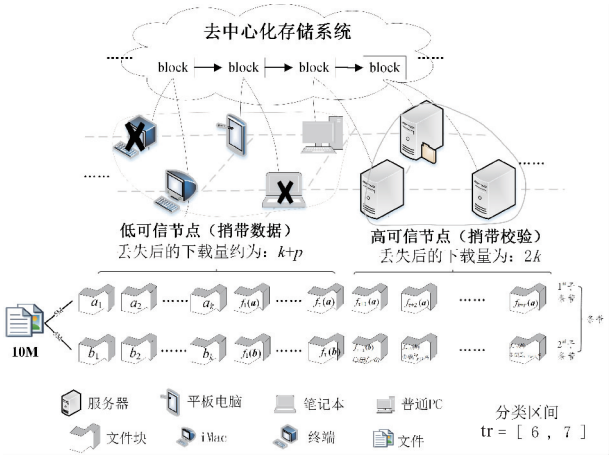

图6　DHitchhiker码的数据存放的结构

4　理论分析

4.1　MDS 性质

DHitchhiker 码把一个条带均分两个子条带的编码,只要一个子条带的 MDS 性质存在,那么整个编码的 MDS 性质都存在. 如图 4 所示,观察 DHitchhiker 码发现,该编码的第一个子条带的数据未做改变. 所以 DHitchhiker 码仍然具有 MDS 性质.

4.2　修复带宽率

修复带宽率是修复失效节点的平均下载量占原数据总量的百分比,如表 3 所示.

表 3　修复带宽率

	不区分节点	区分节点
修复带宽率 γ	$\gamma=(\bar{\delta}/2k)\times 100\%$	
平均修复下载量	$\bar{\delta}=\delta/(k+r)$	$\bar{\delta}=\bar{\delta}^{\text{sys}}p^{\text{low}}+\bar{\delta}^{\text{par}}p^{\text{high}}$
总修复下载量 δ 失效率权重 p	$\delta=\delta^{\text{sys}}+\delta^{\text{par}}$	$p^{\text{low}}=R_{\text{l}}^{\text{L}}/(R_{\text{l}}^{\text{L}}+R_{\text{h}}^{\text{L}})$ $p^{\text{high}}=R_{\text{h}}^{\text{L}}/(R_{\text{l}}^{\text{L}}+R_{\text{h}}^{\text{L}})$ $\bar{\delta}^{\text{sys}}=\delta^{\text{sys}}/k$, $\bar{\delta}^{\text{par}}=\delta^{\text{par}}/r$

RS 码可下载任意的 k 个节点上的数据修复. 分为两个子条带后 RS 码的每个节点的下载量为 $2k$. 又因为 $p^{\text{low}}+p^{\text{high}}=1$, 所以 RS 码的修复带宽率始终为 1. DHitchhiker 码用 MDS 性质修复的捎带校验, 总修复下载量为 $2k(r-\tau)$. 修复捎带数据节点需判断其捎带的位置. 如表 4 所示, 式(2)为横向减法与非横向减法之差. 整体的修复带宽率如表 5 所示.

表 4 数据节点的总修复下载量

捎带校验节点位置	捎带节点数目		捎带节点数		总修复下载量	
	横向减法	非横向减法	横向减法	非横向减法	横向减法	非横向减法
$[k+\tau-1, r-\tau-q]$	$k+\tau-q(p+1)$		p	p	$pq(k+p\tau)+(k+(r+q-\tau-2)\tau)q$	$(k+(p+1)\tau)(pq+q)$
$[r-\tau-q+1, r-\tau]$	pq	$pq+q$	p	$p+1$		

表 5 DHitchhiker 码的修复带宽率

	不区分节点		区分节点	
	横向减法	非横向减法	横向减法	非横向减法
总修复下载量	$pq(k+p\tau)+(k+(r+q-\tau-2)\tau)q+2k(r-\tau)$	$(k+\tau)(k+p\tau)+q(p+1)\tau+2k(r-\tau)$	$\bar{\delta}^{\text{sys}}=(pq(k+p\tau)+(k+(r+q-\tau-2)\tau)q)/(k+\tau)$ $\bar{\delta}^{\text{par}}=2k(r-\tau)/(r-\tau)$	$\bar{\delta}^{\text{sys}}=((k+\tau)(k+p\tau)+q(p+1)\tau)/(k+\tau)$ $\bar{\delta}^{\text{par}}=2k(r-\tau)/(r-\tau)$
修复带宽率	$\dfrac{pq(k+p\tau)+(k+(r+q-\tau-2)\tau)q+2k(r-\tau)}{2k(k+r)}$	$\dfrac{(k+\tau)(k+p\tau)+q(p+1)\tau+2k(r-\tau)}{2k(k+r)}$	$\dfrac{p^{\text{low}}(pq(k+p\tau)+(k+(r+q-\tau-2)\tau)q)/(k+\tau)+2kp^{\text{high}}}{2k}$	$\dfrac{p^{\text{low}}((k+\tau)(k+p\tau)+q(p+1)\tau)/(k+\tau)+2kp^{\text{high}}}{2k}$

4.3 时间复杂度

让参与编码的码字都在有限域 GF(2^m) 内, $O(m)$ 为一次加法或减法的时间复杂度, $O(m^2)$ 为一次乘法的时间复杂度为[24]. 实际上, 一个 (k,r) 的 MDS 码, 生成一个校验或修复一个数据节点需要 k 次乘法和 $k-1$ 次加法. 两种编码的编解码时间复杂度如表 6 所示, x_1 和 x_2 分别为该类型子条带的数量.

表 6 时间复杂度

		RS 码	DHitchhiker 码	
			横向减法	非横向减法
编码时间复杂度	加法	$2r(k-1)$	$2rk+(r-\tau)p$	$2rk-2r+rp-\tau p+q$
	乘法	$2rk$	$2r(k-1)+(r-\tau)p+1$	$2rk+rp-\tau p+q$
	复杂度	$O(2r(km^2+(k-1)m))$	$O((2rk+(r-\tau)p)m^2+(2r(k-1)+(r-\tau)p+1)m)$	$O((2rk+rp-\tau p+q)m^2+(2rk-2r+rp-\tau p+q)m)$
解码时间复杂度	加法	$2\tau(k-1)$	$O_{\text{dhh}}^{\text{dl}}=O(\tau(km^2+(k-1)m)+\tau(pm^2+pm))$ $O_{\text{dhh}}^{\text{dr}}=O((r-\tau-1)(km^2+(k-1)m)+\tau(pm^2+pm))$	$O_{\text{dhh}}^{\text{dl}}=O(\tau(km^2+(k-1)m)+\tau(pm^2+pm))$ $O_{\text{dhh}}^{\text{dr}}=O(\tau(km^2+(k-1)m)+\tau((p+1)m^2+(p+1)m))$
	乘法	$2\tau k$		
	复杂度	$O(2\tau(km^2+(k-1)m))$	$O_{\text{dhh}}^{\text{par}}=O(2\tau(km^2+(k-1)m)+\tau m)$, $O_{\text{dhh}}=(2(r-\tau)O_{\text{dhh}}^{\text{par}}+x_1O_{\text{dhh}}^{\text{dl}}+x_2O_{\text{dhh}}^{\text{dr}}+\tau(km^2+(k-1)m))/2(k+r)$	

5 实验分析

5.1 实验环境

通过 NS3[25] 的事件驱动模拟仿真工具, 构造出由 70 个不同网络带宽、不同稳定性的节点组成的去中心化环境. 这些节点按照 $C=F(H,B,P)$ 的分类模型分类后得到 40 个高可信节点和 30 个低可信节点. 基于该环境, 来仿真 DHitchhiker 码与 RS 码的修复带宽率、编码时间和修复时间.

5.2 实验结果

修复带宽率: 实验仿真了 ($k=10/20/30, \tau=5$) 在不同情况下的修复带宽率. 可以得出 RS 码的修复带宽比率始终为 1, DHitchhiker 码的修复带宽率会因 (k,r) 的不同而变化, 但始终低于 RS 码. 低可信节点的修复带宽可降低 25% ~40.7%, 如图 7(a) 所示. k 相同时, 修复带宽率随着 r 的增大而增大; 在 r 相同时, $k=10$ 的

修复带宽率始终高于 $k = 20$. 当 $k = 10$ 可以节约 0.014% ~ 0.263% 的修复带宽;当 $k = 20$ 可以节约 0.187% ~ 1.389% 的修复带宽,如图 7(b)所示. 用 DHitchhiker-C、DHitchhiker-N 分别表示区分节点和不区分节点的修复带宽率. DHitchhiker-C 进一步降低了约 0.198% 的修复带宽,如图 7(c)所示. 图 7(d)为在常用的参数下,三种编码的对比图,其中 DHitchhiker-C 最低.

编解码时间:实验仿真了($\tau = 5, k = 10$)时,DHitchhiker 码与 RS 码编解码时间开销. RS 码的编码时间低于 DHitchhiker 码,如图 7(e)所示. DHitchhiker 码的解码时间低于 RS 码,如图 7(f)所示. DHitchhiker 码的修复时间低于 RS 码,在常用的参数取值下,DHitchhiker 分别降低约 2.55%, 3.35%, 3.24% 和 2.77% 的修复时间,如图 7(g)所示.

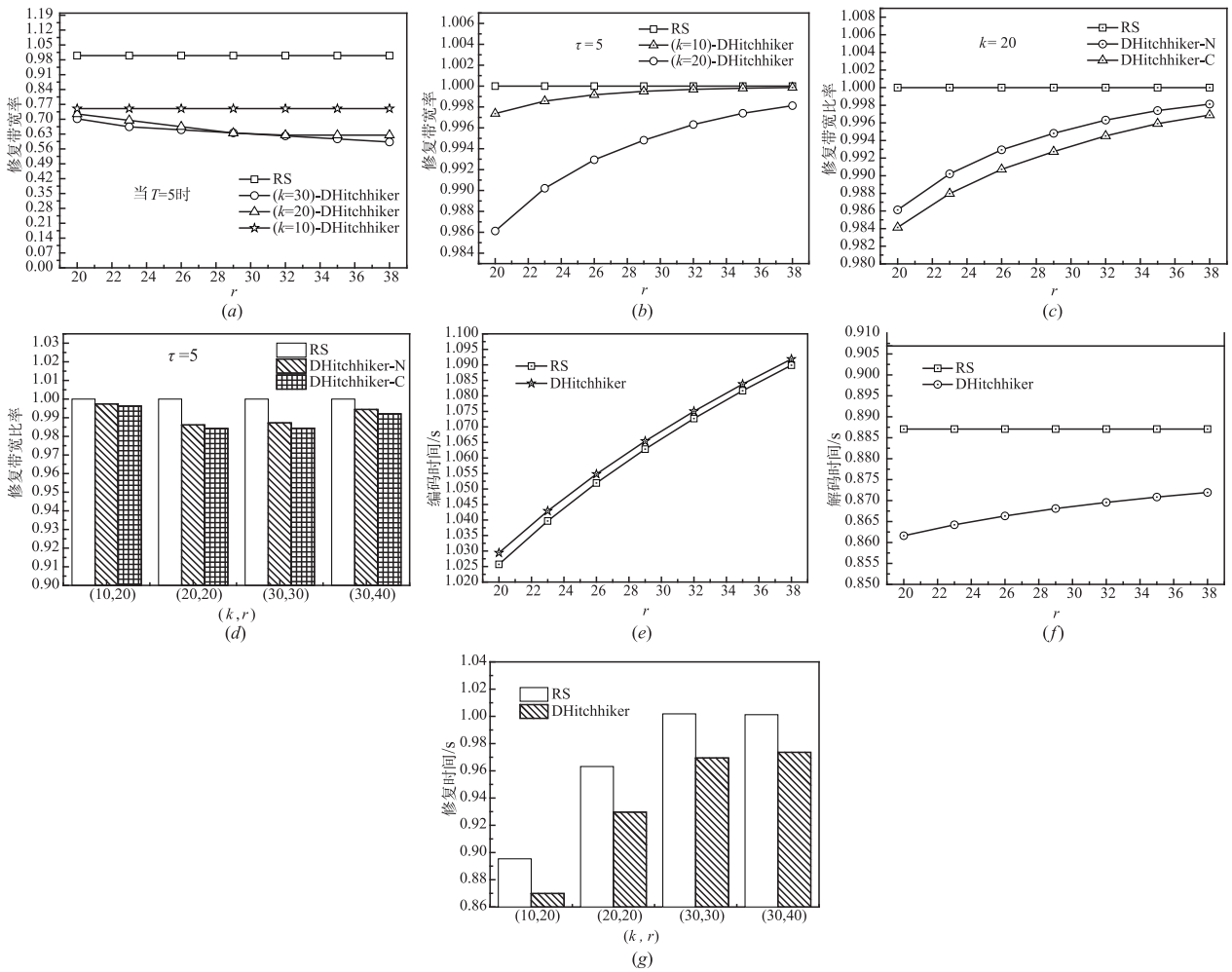

图7　实验结果

6　结束语

本文针对去中心化存储的特殊环境和修复带宽高的问题,提出了去中心化存储中基于可信度的 DHitchhiker 码. 将数据节点和部分校验节点捎带在余下的校验节点上,同时让节点采用特定的存放策略,降低了捎带数据节点约 25%、整体上约 1% 的修复带宽和整体上约 2.5% ~ 3.3% 的修复时间. 该编码着力于减少了修复带宽,会导致少部分情况磁盘 I/O 高于原始的 RS 码.

参考文献

[1] Plank J S. Erasure codes for storage systems:A brief primer [J]. The Usenix Magazine,2013,38(6):44 - 50.

[2] 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(01):238 - 257.
Wang Yijie, Xu Fangliang, Pei Xiaoqiang. Research on error tolerance technology of error correction codes in distributed storage [J]. Journal of Computer Science, 2017, 40(01):238 - 257. (in Chinese).

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

[4] White T. Hadoop: The Definitive Guide[M]. American: O'Reilly Media, Inc. 2011.

[5] Xiaolu L, Runhui L, Lee P P, et al. OpenEC: Toward uni-

fied and configurable erasure coding management in distributed storage systems[A]. 17th USENIX Conference on File and Storage Technologies [C]. Boston, MA: USENIX, 2019. 331 – 344.
[6] Xue R, Guan Z, Gao S, et al. NM2H: Design and implementation of NoSQL extension for HDFS metadata management[A]. IEEE 17th International Conference on Computational Science and Engineering [C]. Chengdu, China: IEEE, 2014. 1282 – 1289.
[7] Nakamoto S. Bitcoin: A Peer-to-peer Electronic Cash System [EB/OL]. https://bitcoin. org/en/bitcoin-paper, 2008.
[8] Storj Labs. Storj: A Decentralized Cloud Storage Network Framework [EB/OL]. https://github. com/storj/whitepaper, 2018-10-16.
[9] NebulousLabs. Sia: Simple Decentralized Storage [EB/OL]. https://gitlab. com/ NebulousLabs/ Sia, 2018-12-21.
[10] Dau H, Duursma I, Kiah H M, et al. Repairing reed-solomon codes with multiple erasures[J]. IEEE Transactions on Information Theory, 2018, 64(20): 6567 – 6582.
[11] e M, Barg A. Explicit constructions of MDS array codes and RS codes with optimal repair bandwidth [A]. 2016 IEEE International Symposium on Information Theory (ISIT)[C]. Barcelona Spain: IEEE, 2016. 1202 – 1206.
[12] Hou H, Han Y S, Shum K W, et al. A unified form of EVENODD and RDP codes and their efficient decoding [J]. Communications IEEE Transactions on, 2018, 66 (11): 5053 – 5066.
[13] Rashmi KV, Shah N B, Ramchandran K. A piggybacking design framework for read-and download-efficient distributed storage codes[J]. IEEE Transactions on Information Theory, 2017, 63(9): 5802 – 5820.
[14] Guiyang L, Xing L, Xiaohu T. An efficient one-to-one piggybacking design for distributed storage systems[J]. IEEE Transactions on Communications, 2019, 67 (12): 8193 – 8205.
[15] Dimakis A G, Godfrey P B, Wu Y, et al. Network coding for distributed storage systems [J]. Information Theory IEEE Transactions on, 2008, 56(9): 4539 – 4551.
[16] Rashmi K V, Shah N B, Kumar P V. Optimalexact-regenerating codes for distributed storage at the MSR and MBR points via a product-matrix construction[J]. IEEE Transactions on Information Theory, 2010, 57(57): 5227 – 5239.
[17] Alrabiah O, Guruswami V. An exponential lower bound on the sub-packetization of MSR codes[A]. Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing[C]. Phoenix: ACM, 2019. 979 – 985.
[18] Jie L, Xiaohu T, Chao T. A generic transformation to enable optimal repair in MDS codes for distributed storage systems[J]. IEEE Transactions on Information Theory, 2018, 64(9): 6257 – 6267.
[19] Hou H, Lee P P. Binary MDS array codes with optimal repair[J]. IEEE Transactions on Information Theory, 2020, 66(3): 1405 – 1422.
[20] Rashmi KV, Shah N B, Gu D, et al. A "hitchhiker's" guide to fast and efficient data reconstruction in erasure-coded data centers[J]. ACM Sigcomm Computer Communication Review, 2014, 44(4): 331 – 342.
[21] Shen Z, Lee P P, Shu J, et al. Cross-rack-aware single failure recovery for clustered file systems[J]. IEEE Transactions on Dependable and Secure Computing, 2020, 17 (2): 248 – 261.
[22] 江小玉,李贵洋,周悦,等. 基于负载均衡的纠删码修复流水线[J]. 电子学报,2020,48(5):930 – 936.
Xiaoyu J, Guiyang L, Yue Z, et al. Repair pipelining for erasure-coded storage based on load-balanced [J]. Acta Electronica Sinica, 2020, 48(5): 930 – 936. (in Chinese).
[23] Bhagwan R. Total recall: System support for automated availability management [J]. USENIX Association, 2004, 01(04): 1 – 25.
[24] Kumar S, Amat A G I, Andriyanova I, et al. Code constructions for distributed storage with low repair bandwidth and low repair complexity[J]. IEEE Transactions on Communications, 2018, 66(12): 5847 – 5860.
[25] Amazon Inc. Amazon Simple Storage Service-Object-Metadata [EB/OL]. https://docs. aws. amazon. com/ AmazonS3/latest/dev/UsingMetadata. html, 2018.

作者简介

胡金平 女,1994 年生于重庆云阳. 四川师范大学计算机科学学院硕士研究生. 研究方向为分布式存储、纠删码.
E-mail: hu_jp1124@163. com

李贵洋(通信作者) 男,1975 年生于四川宜宾. 现为四川师范大学计算机科学学院教授. 研究生导师. 主要研究方向为大数据存储编码、机器学习.
E-mail: gyli@ sicnu. edu. cn