基于负载均衡的纠删码修复流水线

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

摘 要: 大数据分布式存储系统中,修复流水线(Repair Pipelining,RP)减少90%的修复时间,有效地解决由于修复时间开销较大,纠删码不适用于存储热数据的问题.然而,现有的RP存在节点负载不均衡的问题,导致系统性能下降.通过研究后,设计节点负载均衡的纠删码修复流水线(Node Load Balancing-based Repair Pipelining,NLB-RP),并根据性能评价指标提出计算节点负载的算法和计算修复时间的算法.理论分析及实验结果表明,在没有引入额外修复代价的情况下,NLB-RP从局部到整体有效地平衡并减少节点的负载.相比RP,NLB-RP的节点负载方差为0,即每个节点的负载相同.因此,NLB-RP具有最优的负载均衡性.

关键词: 大数据;分布式存储;纠删码;修复流水线;负载均衡
中图分类号: TP333 **文献标识码:** A **文章编号:** 0372-2112 (2020)05-0930-07
电子学报URL: http://www.ejournal.org.cn **DOI:** 10.3969/j.issn.0372-2112.2020.05.013

Repair Pipelining for Erasure-Coded Storage Based on Load-Balanced

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

Abstract: In distributed storage system for big data,the repair pipelining (RP) reduces repair time by 90%,which effectively solves the problem that erasure code is not suitable for storing hot data due to the heavy overhead of repair time. However,the existing RP has the problem of unbalanced load among nodes,which leads to the degradation of system performance. In this paper,a repair pipelining based on node load balancing (NLB-RP) is designed,and then the algorithms for calculating the load of nodes and repair time according to the evaluation index of performance are proposed. Theoretical analysis and experimental results both show that,from local to global,the NLB-RP can effectively balance and reduce the load of nodes without introducing extra repair cost. Compared with the RP,the load variance of the NLB-RP is zero,that is,the load of each node is the same. Thus,the proposed NLB-RP has the optimal load balance.

Key words: big data;distributed storage;erasure code;repair pipelining;load balancing

1 引言

纠删码[1,2]作为一种数据保护机制,因其高容错性和低冗余度,广泛应用于分布式存储[3,4]、区块链、机器学习等前沿技术中.比如,RS(Reed-Solomon)[5]码可用于GFS[6]、HDFS[7]、Storj[8]、Filecoin[9]等系统中保障数据的可靠性、减少存储开销;也可用于神经网络中改善复杂的非线性计算[10,11].但是,纠删码修复时需要下载传输数倍于失效数据大小的数据量,集群中的修复带宽约占网络总平均流量的10%~20%,网络容易拥堵,导致修复时间急剧增加[12],降低修复效率.因此,在实际的存储系统中,纠删码不适用于存储热(频繁读取的)数据[13].

如何提高修复效率是纠删码研究领域的一个关键点[14],近年来,有关方面的研究成果不断地被提出[15~19].其中,Patrick Lee等人在2017年提出RP[17],将工业上的流水线概念用于加速纠删码修复过程,减少90%的修复时间,并且对RP中节点负载(工作量)不均衡的问题做了相应处理,但仍没有达到完全平衡.节点负载不均衡使得部分节点资源闲置,部分节点工作任务繁重而成为瓶颈节点.如果出现拥塞情况,节点资源被占用,无法及时完成其他工作请求,会导致系统整体性能下降,更严重的会直接造成"雪崩".

基于此,本文提出基于负载均衡的纠删码修复流水线—NLB-RP.主要从两个方面改进RP:首先,构造数据传输结构Comns,从局部平衡节点负载;然后,增加帮助(参与修复的)节点个数,降低单个帮助节点分摊的工作量,从整体降低负载.NLB-RP基于RP理论框架,不改变底层的纠删码,对系统的存储效率、容错能力、修

收稿日期:2019-07-04;修回日期:2020-02-07;责任编辑:马兰英
基金项目:国家自然科学基金(No.61701331)

复带宽等性能不产生负面影响,仍保持良好的修复效率. 理论和实验证明,NLB-RP 具有最优的负载均衡性.

2　相关理论基础

2.1　参数

为了方便叙述,将后文需要用到的参数罗列成表,如表 1 所示.

表 1　参数列表

参数	描述	参数	描述
n	节点数量	d	修复一个数据块时连接的节点数
k	原始数据块个数	B	数据块的大小
m	产生的冗余数据块个数	s	数据切片的个数

2.2　RS 码简介

RS 码是应用得最广泛的一类纠删码,具有容错力高,冗余度低的特点,且编解码技术成熟. RS 具有 n、k、m 参数,编码时将原始文件切分为大小相同的 k 个数据块,通过计算生成 m 个冗余编码块,然后存储在 $n(n=k+m)$ 个独立的节点上. RS 具备 MDS(Maximum Distance Separable)性质,解码时可以通过读取 n 中任意 k 个节点上的数据,与解码系数计算得到其他 m 个节点上的数据. RS 的计算在有限域上进行,计算后不会改变数据块的大小,具体的编码和解码原理可以查看相关文献[20,21].

数据修复过程相当于解码过程,当节点 i 发生故障时,存储的数据 C_i 失效,需要被计算恢复. 根据 MDS 性质,下载任意 k 个节点上的可用编码块,记为 $(C_1,C_2,\cdots,C_k)$,并求出解码系数,记为 $(a_1,\cdots,a_k)$. 可计算得到与失效编码块内容相同的数据块,记为 C_i',则:

$$C_i' = \sum_{i=1}^{k} a_i C_i \tag{1}$$

2.3　RP 的原理及结构

常规修复和 RP 网络拓扑如图 1 所示,N_i 表示第 i 个节点,R 表示替代节点(代替故障节点的新节点). 数据 $b_1 \sim b_4$,p_1,p_2 分别存放在 $N_1 \sim N_6$ 上,如果 N_5 发生故障,p_1 需要被恢复. $N_1 \sim N_4$ 作为帮助节点,设解码系数为 $(a_1,\cdots,a_k)$,根据式(1),可得 $p_1 = a_1b_1 + a_2b_2 + a_3b_3 + a_4b_4$. 常规修复中,帮助节点分别发送数据给 R,R 接收完所有数据后才做解码运算,当带宽有限时,进入 R 的链路之间存在网络资源竞争,容易变得拥塞. RP 将节点互联,N_1 将 b_1 乘以系数 a_1 后发送给 N_2,N_2 接收后与本地的数据计算,然后将数据 $(a_1b_1 + a_2b_2)$ 发给 N_3,以此类推,直到将恢复后的数据 p_1 发送给 R. RP 重新分配数据流量,将本应全部发往 R 的流量分配到其他节点上,消除网络性能瓶颈,缓解常规修复中拥塞的状态,减少 90% 的修复时间.

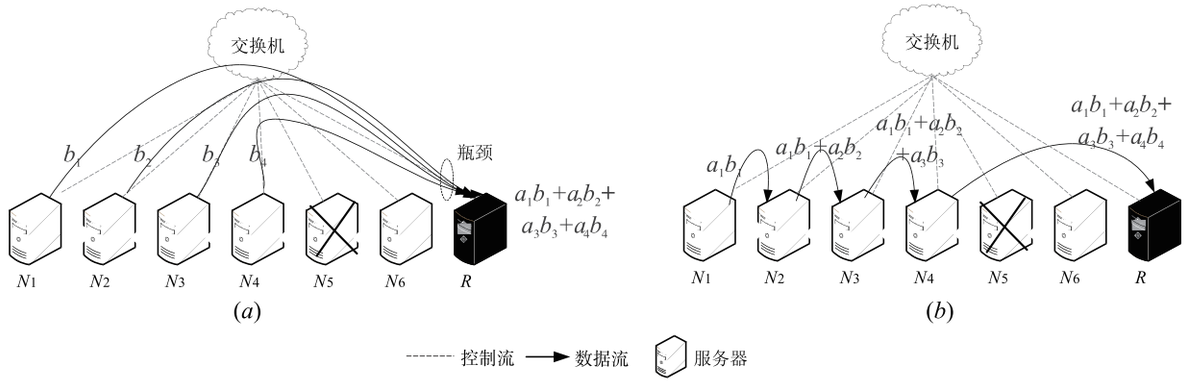

图1　常规修复和RP网络拓扑图

RP 中整个修复操作被划分成多个修复子操作. 如图 2 所示,一个包含 n 个编码块的集合被称为条带,大型存储系统中存储多个条带数据,每个条带独立编码,条带上相同偏移位置的数据被编码在一起. 解码时,数据块被分解为更小尺寸的单元,称为数据切片(子块),包含 n 个数据切片的集合被称为子条带. 一个数据块的修复过程就是多个数据切片修复过程的集合.

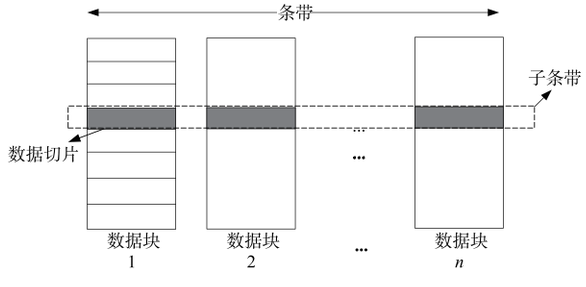

图2　条带示意图

Patrick Lee 在文献[17]中设计了基本网络结构(Basic),示例如图 3 所示,$N_1 \rightarrow N_2,\cdots,\rightarrow N_k \rightarrow R$ 表示流水线路径,节点间的箭头表示数据流向,$S_1,\cdots,S_s$ 表示当前在修复第 s 个数据切片. t_0 时刻 N_1 发送数据;t_1 时刻

N_2接收数据并计算,同时N_1继续发送第二个数据切片;t_3时刻N_4将修复完的第一个数据切片传输给R,其他节点依次处理相应数据;以此类推,直到整个数据块分步完成修复. Basic 并行调度修复子过程,修复效率显著提升. 然而,Basic 中节点负载不均衡,例如N_1只负责发送本地的数据,N_2、N_3需要接收、计算、转发数据,N_4还需与R交互,如果R位于网络边缘,$N_4 \rightarrow R$容易持续拥塞而成为系统瓶颈链路.

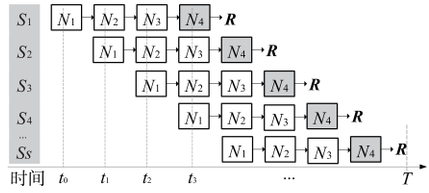

图3 Basic(n=6, k=4)

为了平衡节点的负载、消除性能瓶颈,Patrick Lee 设计新的网络结构(Cyclic). 如图 4(a)所示:将节点循环向前移位以构造多条不同的流水线路径,并分组修复. 每个组的修复分为两步:第一步,按流水线路径传输、计算数据;第二步,流水线上的终点节点将修复后的数据切片传给R,R收集完数据切片后组合成数据块. 每一组的第二步可以和下一组的第一步并行. 从t_0时刻开始,Cyclic 中各个帮助节点循环位于流水线各站,承担不同的修复任务,平衡节点的负载;R从多个节点获取数据,消除性能瓶颈. 但是,Cyclic 中仍存在问题:(1)节点负载不够均衡,如上述例子中的N_3没有与R交互,负载更小;(2)所有数据切片都利用相同的k个帮助节点完成修复,其他节点的网络资源没有得到充分利用,这k个节点在完成修复任务时也可能难以兼顾其他的工作请求,导致系统整体性能下降.

3 负载均衡的修复流水线

在 RP 理论框架和现存问题的基础上,提出基于节点负载均衡的修复流水线—NLB-RP. 真实网络环境中,节点的带宽和其他资源是动态变化的,难以实际测量,无法直接避开瓶颈节点,只能通过均匀分配降低瓶颈出现的概率,保证没有瓶颈的流水线路径能尽快完成修复. 因此,NLB-RP 的核心思想在于:构造更多的流水线路径,让节点均匀地参与修复中的各项工作. 具体操作为:(1)当$d=k$时,构造k条不同的流水线路径;(2)令$d>k$,就是尽可能多的选择节点分摊修复工作量.

3.1 优化负载均衡性

当$d=k$时,设计网络结构 Comns 以解决节点负载不够均衡的问题. 在 Comns 中构造k条不同的流水线路径,并且第一个分组中并行调度k条流水线. 假设$n=6, k=4, d=k=4, s=7$,分组数$g=2$,示例如 4(b)所示,修复过程如下:t_0时刻第一组修复操作启动,所有帮助节点开始工作;$t_1 \sim t_3$时刻,持续修复操作;t_4时刻,第一条流水线上N_4将修复完成的数据切片发送给R,同时,第二组的$k-1$条流水线启动,此时仍然有k个帮助节点在工作;以此类推,直到完成修复.

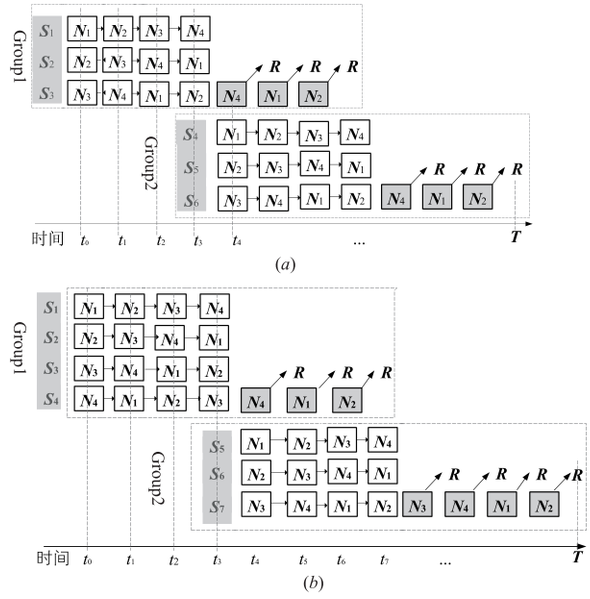

图4 Cyclic和Comns网络结构图(n=6, k=4, d=4)

将图 4(a)与 4(b)对比,可以看出 Comns 不仅均衡了节点的负载,而且提高了修复效率. Cyclic 中只有$k-1$条不同的流水线路径,因此有 1 个帮助节点没有轮询到终点站,节点负载不够均衡,而 Comns 中有k条不同的流水线路径,帮助节点循环位于流水线各站,节点负载完全均衡. 并且修复过程中,Comns 的前$g-1$个分组里每个时刻所有的帮助节点都在工作,并行度达最大化,而 Cyclic 在第二组才实现了并行度最大化.

3.2 增加帮助节点个数

当一个节点发生故障,系统中实际还有$n-1$个可用节点,可以选用更多节点参与修复,进一步降低节点负载. 数据切片利用不尽相同的k个节点进行修复,虽然连接的节点数多于k,但只需要传输部分数据切片,不会增加总的修复带宽. 修复步骤如图 5 所示.

第一步,确定帮助节点. 从$n-1$个可用节点中选取带宽较大的d个节点,形成帮助节点集合.

第二步,确定每个数据切片的帮助节点. 任意一个数据切片的修复都遵从 MDS 性质,从帮助节点集合中选取k个节点进行修复计算.

第三步,构造流水线路径. 为了让修复过程更有序,将帮助节点按编号大小排列,然后循环向前移位,构造流水线路径. d个帮助节点可以构造d条不同的流水线路径.

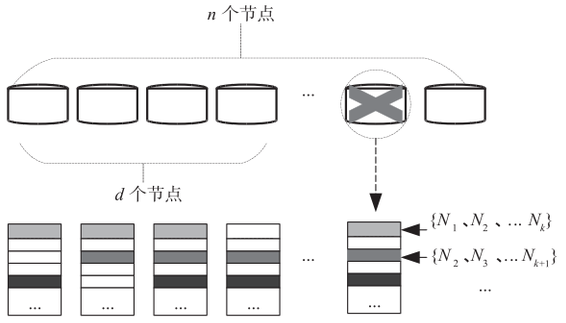

图5　NLB-RP修复步骤图

第四步,分组修复. 分组数 $g=\lceil (s-d)/(k-1)\rceil+1$,如果$(s-d)$不能被$(k-1)$整除,最后一组的切片数少于 $k-1$. 第一个分组中同时启动所有帮助节点,d 条流水线并发. 第二组的修复操作与第一组中终点节点将修复完成的数据切片依次发送给 R 的操作并行. 为了提高修复效率,希望有更多节点同时工作,但是一个帮助节点同一个时刻在流水线上不可重叠. 第一个与 R 交互的节点为 N_k,此时 N_k不能作为流水线的起始节点,这一组中只能调用 $N_1\sim N_{k-1}$为起始节点的流水线,否则 N_k会因为有多个数据出口而陷入繁忙状态,造成网络拥塞. 同理,第二组之后的分组,每组都只能调度 $k-1$条流水线.

根据 d 取值的不同,网络结构也不相同,扩展 Comns 构造一个更具一般性的网络结构. 用一个三元组 $G=\{V,E,W\}$ 表示此结构,其中:

V 表示节点集合,$V=\{R,N_1,\cdots,N_d\}$;

E 表示流水线路径,$E=\{e_i\mid i=1,\cdots,d\}$,$e_i$表示第 i 条路径;

W 表示节点带宽,$W=\{w_i\mid i=1,\cdots,d\}$.

假设 $n=6,k=4,d=5,s=8$,示例如图 6 所示,修复过程如下:t_0时刻第一组调度流水线 $e_1\sim e_5$,所有帮助节点发送本地数据切片;t_1时刻节点接收上一个节点发送的数据,与本地的切片进行计算组合后发送给下一个节点;t_3时刻第一组的 5 个数据切片修复完成;t_4时刻 N_4将修复完的第一个数据切片发给 R,同时,第二组调用 $e_1\sim e_3$修复第 6 ~ 8 个数据切片;t_5时刻 N_5将修复完成的第二个切片传输给 R,第二组继续修复操作;以此类推,分组循环调用 $e_1\sim e_5$,直至完成修复. 从第二组开始,每组只能调用 3 条流水线,因为 t_4时刻 N_4在往 R 发送数据,数据出口被占用,无法调用 e_4. 并且如果 t_4时刻调用 e_5,虽然此时 N_5能正常工作,但破坏了流水线路径的循环结构,在后续的修复过程中必然会造成其他帮助节点在流水线上的重叠(保证 s 足够大).

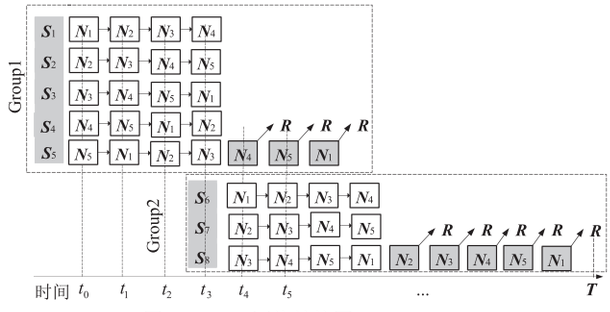

图6　Comns网络结构图(n=6，k=4，d=5)

4　性能分析

纠删码存储系统主要的性能指标有存储效率,容错能力,修复带宽,计算复杂度,负载均衡性,修复时间. 文中对改进后可能影响的负载均衡性和修复时间作出考量. 使用其他纠删码会影响其他几个性能,文中没有改变编码,因此不作讨论.

4.1　计算节点负载

本小节提出算法 1,计算修复单位数据量时的节点负载.

算法 1　计算节点负载

输入:θ、E、d. //分别表示单位数据量,流水线路径集合,帮助节点个数.

输出:L. //节点负载.

step1 $\alpha_1=1,\alpha_2=2,\alpha_3=3$. //初始化起始节点、中间节点、终点节点的工作量权重.

step2 计算 N_i的工作量权重 β_i:

For $i=1$ to d

If e_i in E Then $\beta_i+=1$

If $N_i\rightarrow N_{i+1}$ in E Then $\beta_i+=2$

If $N_i\rightarrow R$ in E Then $\beta_i+=3$

step3 $\rho_i=\beta_i\Big/\sum_{i=1}^{d}\beta_i$. //计算 N_i的工作量比例.

step4 $L_i=\rho_i\theta$. //计算修复单位数据量时 N_i的负载.

step5 $L_{avg}=\theta/d$. //计算节点的平均负载,平均负载 = 工作量/帮助节点数.

step6 输出节点负载,算法停止.

4.2　计算修复时间

本小节提出算法 2,计算修复时间.

纠删码系统中的修复时间由几部分组成,计算公式为 $T=T_{trans}+T_{I/O}+T_{cpt}$. 其中,$T$ 表示总的修复时间,T_{trans}表示数据传输时间,$T_{I/O}$表示磁盘 I/O 时间,T_{cpt}表示解码计算时间. 如今计算机的计算能力非常强,且数据传输时间占修复时间的 94%[14],因此,T_{cpt}忽略不计.

分组修复数据切片,分组数 $g=\lceil (s-d)/(k-1)\rceil+1$,当不能完整分组时,修复最后一组切片消耗的时间不同,需要分开计算. 令 $T_{trans}=t_{g-1}+t_{last}$,$t_{g-1}$表示前$(g-1)$组消耗的时间,$t_{last}$表示最后一组消耗的时间. t 表示一个数据切片一次传输消耗的时间,修复一个切片需要 k 次数据传输,消耗时间 kt,一组数据切片消耗时

间 $t_{gc}=2(k-1)t$. 从第二组开始,修复操作并行,前$(g-1)$组消耗传输时间为 $t_{g-1}=\frac{1}{2}\sum_{i=1}^{g-1}t_{gc}$.

算法 2 计算修复时间

输入:s、k、d、t、$t_{I/O}$. //分别表示数据切片数,原始数据块数,帮助节点数,一次传输消耗的时间,磁盘 I/O 时间.

输出:T. //修复时间.

step1 计算数据传输时间:$T_{trans}=t_{g-1}+t_{last}$.

1.1 $t_{g-1}=\frac{1}{2}\sum_{i=1}^{g-1}t_{gc}=(g-1)(k-1)t$;//计算前$(g-1)$组数据切片传输时间.

1.2 计算最后一组的数据传输时间:

$r=(s-d)\%(k-1)$;//最后一组修复的数据切片数.

If $r=0$ Then $t_{last}=(d+k-1)\ t$

Else $t_{last}=(d+r)\ t$

1.3 $T_{trans}=(s+k-1)\ t$. //数据传输时间.

step2 $T_{I/O}=dt_{I/O}$. //计算磁盘 I/O 时间.

step3 $T_{cpt}=0$. //初始化解码计算时间.

step4 $T=(s+k-1)t+dt_{I/O}$. //计算修复时间.

step5 输出 T,算法停止.

4.3 理论分析

4.3.1 节点负载

根据算法 1,计算 Basic、Cyclic 和 Comns 的节点负载. 并计算帮助节点负载方差,方差反映的是节点负载分布的变异范围和离散幅度,体现了一组数据波动的范围,计算公式为 $\sigma(L)^2=\frac{1}{d}\sum_{i=1}^{d}(L_i-L_{avg})^2$. 各结构的节点负载情况如表 2 所示.

表 2 节点负载情况

	L_1	$L_2\sim L_{k-2}$	L_{k-1}	L_k	L_{avg}	$\sigma(L)^2$
Basic	$\theta/2k$	θ/k	θ/k	$3\theta/2k$	θ/k	$\theta^2/4k^3$
Cyclic	θ/k	θ/k	$5\theta/6k$	$7\theta/6k$	θ/k	$\theta^2/36k^3$
Comns	θ/d	θ/d	θ/d	θ/d	θ/d	0

显然,$L_{avg}^{Basic}=L_{avg}^{Cyclic}\geqslant L_{avg}^{Comns}$,Basic 和 Cyclic 节点平均负载相同,且大于等于 Comns 的节点平均负载,这说明当 $d>k$ 时,Comns 从整体上减少了节点的负载. 并且,$\sigma(L^{Basic})^2>\sigma(L^{Cyclic})^2>\sigma(L^{Comns})^2$,这意味着 Comns 和 Cyclic 都从局部均衡了节点的负载. 但是 Cyclic 节点负载方差大于 0,均衡并不够完善,而 Comns 节点负载方差为 0,即各节点的负载是相同的,节点负载均衡性达到了最优.

4.3.2 修复时间

Basic 中数据切片作为一个整组进行修复,Cyclic 中第一组修复 $k-1$ 个数据切片,Comns 中第一组修复 d 个数据切片. 根据算法 2,计算 Basic、Cyclic 和 Comns 的修复时间,结果如表 3 所示.

表 3 修复时间对比表

Basic	Cyclic	Comns
$(s+k-1)t+kt_{I/O}$	$(s+k-1)t+kt_{I/O}$	$(s+k-1)t+dt_{I/O}$

Basic 与 Cyclic 的修复时间相同. Comns 和 Cyclic 的时间差记为 ΔT,$\Delta T=(d-k)t$. 当 $d=k$ 时,$\Delta T=0$,Comns 没有影响修复时间,它的负载均衡性是完全优于 Cyclic 的;当 $d>k$ 时,$\Delta T>0$,Comns 会增加部分 I/O 请求时间,但是修复时间中数据传输时间占比高达 94%,I/O 时间占的比例非常小,不会对总的修复时间造成太大的影响.

5 实验及评价

为了验证以上的理论推导,进行仿真实验论证 NLB-RP 的有效性,测定不同影响因素下节点负载的情况,并对比分析 NLB-RP 与 RP 的实验结果.

5.1 实验步骤

第一步:配置实验环境. 硬件环境:8GB 内存、Intel core i5-4570 处理器、3.40GHz、1T 硬盘. 软件环境:ubuntu16.04 系统,NS-3(Network Simulator version 3)运行环境.

第二步:配置实验参数. 在 NS-3 运行环境中构建网络拓扑,实验模型如图 7 所示. 配置链路上网络带宽为 1Gb/s、时延为 2ms,IP 地址为 10.0.0.x,以及其他相关参数.

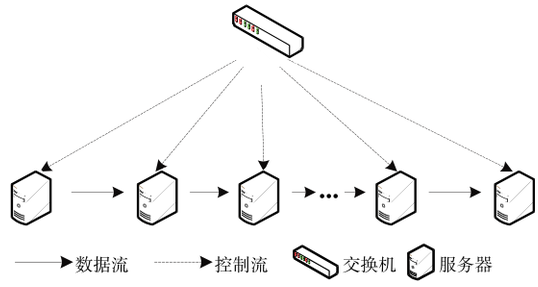

图7 网络拓扑模型

第三步:测试数据. 实验分别测定在不同数据块大小、数据切片大小、以及(k,m)-RS 编码方案下,Basic、Cyclic、Comns 的平均负载和修复时间,Comns 中帮助节点个数取最大值,即 $d=n-1$.

第四步:作图. 经实验发现 Basic 与 Cyclic 的节点平均负载和修复时间基本相同,为了更清晰的绘制实验结果图,只保留 Cyclic 与 Comns 的数据.

5.2 实验结果

数据大小:设定数据切片大小为 2KB,单位数据量为 2MB,总节点数为 9,编码方案为(6,3)-RS. 图 8(a)、(b)表示 Cyclic、Comns 的节点平均负载、修复时间随修复数据块大小改变的变化. 可以看出,节点平均负载和修复时间都与数据大小成正比,相比 Cyclic,Comns 最多可减少 25% 的平均负载,约增加 1% ~3% 的修复时间.

切片大小:设定数据块大小为 64MB,单位数据量为 2MB,总的节点数为 9,编码方案为(6,3)-RS. 图 8(*c*)、(*d*)表示 Cyclic、Comns 的节点平均负载、修复时间随切片大小改变的变化. 从图 8(*c*)可以看出,数据切片的大小并不影响节点平均负载,因为不管将数据块切分成多少个切片,数据总量固定,节点的负载量也就固定;从图 8(*d*)可以看出,随着切片大小增加,切片个数相应减少,修复时间也相应减少.

编码参数:设定数据块大小为 64MB,切片大小为 32KB,单位数据量为 2MB. 选取主流数据中心常用的(k,m)-RS 编码方案,总节点数等于 $k+m$. 例如:QFS 中心的(6,3)-RS、Yahoo 平台的(8,4)-RS、Facebook 中的(10,4)-RS 等. 图 8(*e*)、(*f*)、(*g*)分别表示平均负载、修复时间、负载方差随编码参数改变的变化. 从图 8(*e*)可以看出,随着参数 k 的增大,节点的平均负载逐渐减小. 从图 8(*f*)可以看出,编码参数的变化几乎不对修复时间造成影响. 从图 8(*g*)可以看出,随着 k 值的增大,Basic、Cyclic 的节点负载方差逐渐减小,说明节点负载愈加均衡,增加帮助节点数来均衡负载的方法是可行的. 随着 k 值的增大,Basic、Cyclic 和 Comns 的节点负载方差趋近于相同,Comns 的节点负载均衡性优势降低. 但在纠删码系统中,为保证存储效率,不宜采用 k 值较大的编码方案,因此,Comns 仍然是可取的.

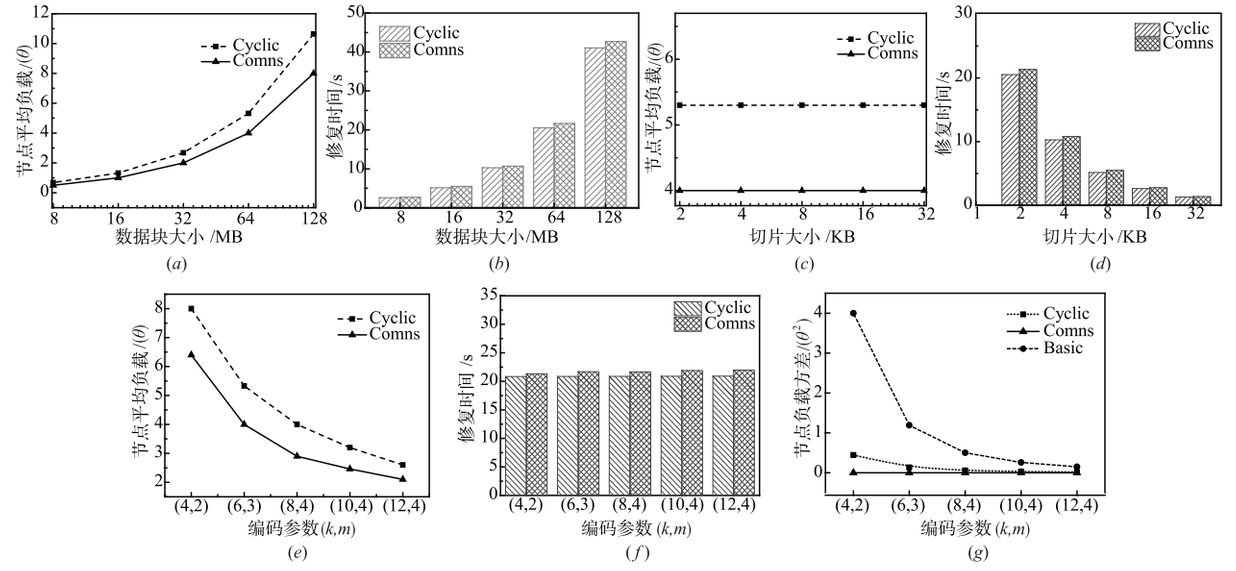

图8　实验结果

5.3　实验评价

NLB-RP 的节点负载均衡性主要与纠删码方案的编码参数有关,修复时选择的帮助节点个数越多,效果越好. 帮助节点均衡地参与修复工作,各节点的负载相同,均衡性达到了最优. 修复时,NLB-RP 中的帮助节点的个数是可选择的,动态均衡地调度节点参与修复,扩展了修复流水线在纠删码系统中的应用场景. 此外,NLB-RP 保持了 RP 优秀的修复效率,虽然帮助节点个数的增加导致 I/O 时间增加,但相比于总的修复时间,毫秒级的时间差异可以忽略不计.

6　结论

NLB-RP 基于 RP 的理论框架,不会对纠删码的其他性能(例如存储效率、网络带宽等)造成负面影响,在保持修复效率的情况下,有效地改善节点负载不均衡的问题. 本文首先提出网络结构 Comns,保证即使在帮助节点个数为 k 的情况下,也能从局部平衡节点的负载. 进一步地,增加帮助节点个数,构造更具一般性的 Comns,从整体降低节点的平均负载. 理论分析及实验结果证明了 NLB-RP 的可行性,它具有平衡节点负载的能力,并且,修复性能随着调用更多的可用节点参与修复而提升.

参考文献

[1] Huang C, Simitci H, Xu Y, et al. Erasure coding in windows azure storage[A]. Proceedings of the 2012 USENIX conference on Annual Technical Conference[C]. Berkeley, CA: USENIX, 2012. 2 – 2.

[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(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(01):238 – 257. (in Chinese)

[3] Ford D, Labelle F, Popovici F I, et al. Availability in globally distributed storage systems[A]. Proceedings ofUsenix Conference on Operating Systems Design & Implementation [C]. Berkeley, CA: USENIX, 2010. 61 – 74.

[4] 吴炀,付印金,陈卫卫,倪桂强. 一种高效的混合内存布局机制与编码技术[J]. 计算机科学, 2017, 44(6): 57 – 62.
Wu Yang, Fu Yin-jin, Chen Wei-wei, Ni Gui-qiang. Efficient mechanism of hybrid memory placement and erasure code[J]. Computer Science, 2017, 44(6): 57 – 62. (in Chinese)

[5] 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.

[6] Ghemawat S, Gobioff H, Leung S, et al. The Google file system[J]. Symposium on Operating Systems Principles, 2003, 37(5): 29 – 43.

[7] The Apache Software Foundation. HDFSArchitecture[DB/OL]. http://hadoop. apache. org/docs/current/hadoop-project-dist/hadoop-hdfs/HDFSErasureCoding. html, 2018-11-13.

[8] Wilkinson S, Boshevski T, Prestwich J, et al. Storj a peer-to-peer cloud storage network[DB/OL]. https://storj. io/storjv2. pdf, 2016-12-15.

[9] Protocol Labs. FileCoin: A decentralized storage network [DB/OL]. http://filecoin. io/filecoin. pdf, 2017-6-19.

[10] KosaianJ, Rashmi K V, Venkataraman S. Learning a code: Machine learning for approximate non-linear coded computation[J]. arXiv: Learning, 2018.

[11] Karakus C, Sun Y, Diggavi S, et al. Redundancy techniques for straggler mitigation in distributed optimization and learning[J]. arXiv: Machine Learning, 2018.

[12] Sathiamoorthy M, Asteris M, Papailiopoulos D S, et al. XORing elephants: Novel erasure codes for big data[J]. very large data bases, 2013, 6(5): 325 – 336.

[13] 李元超. 面向冷数据存储的分布式编码技术研究与实现[D]. 武汉: 华中科技大学, 2016. 1 – 62.
Li Yuanchao. Research and Implementation of Distributed Coding Technology for Cold Data Storage [D]. Wuhan: Huazhong University of Science and Technology, 2016. 1 – 62. (in Chinese)

[14] Rashmi K V, Nakkiran P, Wang J, et al. Having your cake and eating it too: jointly optimal erasure codes for I/O, storage and network-bandwidth[A]. Proceedings of Usenix Conference on File & Storage Technologies [C]. Berkeley, CA: USENIX, 2015. 81 – 94.

[15] Rashmi K V, Shah N B, Gu D, et al. A "hitchhiker's" guide to fast and efficient data reconstruction in erasure-coded data centers[J]. ACM Special Interest Group on Data Communication, 2015, 44(4): 331 – 342.

[16] Mitra S, Panta R K, Ra M, et al. Partial-parallel-repair (PPR): a distributed technique for repairing erasure coded storage [A]. Proceedings of European Conference on Computer Systems[C]. Dresden, Germany: ACM, 2016. 1 – 30.

[17] Li R, Li X, Lee P P, et al. Repair pipelining for erasure-coded storage [A]. Proceedings of the 2017 USENIX Conference on Annual Technical [C]. Berkeley, CA: USENIX, 2017. 567 – 579.

[18] Yuan S, Huang Q, Wang Z. A repair-efficient coding for distributed storage systems under piggy backing framework[J]. IEEE Transactions on Communications, 2018, 66(8): 3245 – 3254.

[19] Pamiesjuarez L, Blagojevic F, Mateescu R, et al. Opening the chrysalis: on the real repair performance of MSR codes [A]. Proceedings of the Conference on File and Storage Technologies[C]. Berkeley, CA: USENIX, 2016. 81 – 94.

[20] Reed I S, Solomon G. Polynomialcodes over certain finite fields[J]. Journal of the Society for Industrial and Applied Mathematics, 1960, 8(2): 300 – 304.

[21] Plank J S, Xu L. Optimizing cauchy reed-solomon codes for fault-tolerant network storage applications [A]. Proceedings of Network Computing and Applications [C]. Cambridge, USA: IEEE, 2006. 173 – 180.

作者简介

江小玉 女,1994 年出生于四川自贡,现为四川师范大学计算机科学学院硕士研究生,主要研究方向为分布式存储、纠删码.
E-mail: jiang_xy0805@163. com

李贵洋(通信作者) 男,1975 年出生于四川宜宾,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