基于条带配对合并算法的局部可修复码冗余度转换机制

杜清鹏，许胤龙，吴思

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.221100257

基于条带配对合并算法的局部可修复码冗余度转换机制

杜清鹏[1]　许胤龙[2]　吴　思[2]
中国科学技术大学计算机科学与技术学院　合肥 230027
(duqingpeng@mail.ustc.edu.cn)

摘　要　相比传统的多副本技术,纠删码是一种以高修复代价换取低存储开销的数据冗余机制。局部可修复码是一类具有低修复代价的纠删码,被广泛应用在大数据存储系统中。为了应对动态变化的工作负载和存储介质动态改变的故障率,现代存储系统需要对纠删码数据进行冗余度转换,以调节数据访问性能和可靠性。设计了一种基于条带配对合并的局部可修复码冗余度转换方法,通过选择特定位置的条带进行配对合并,实现了冗余度转换与数据布局的解耦合;进一步通过设计代价量化方法与最优化模型,降低了冗余度转换的网络通信开销。相比设计数据布局的算法,所提算法有与其近似的性能,但对数据布局无限制,可级联迭代地多次运行。实验结果表明,在两种冗余度转换设置下,所提算法均近似于理论最优值,相比随机布局的朴素算法,网络流量分别降低了 27.74%和 27.47%,耗时分别缩短了 39.10%和 22.32%。

关键词:局部可修复码;冗余度转换;容错存储技术;网络流量优化;分布式存储系统
中图法分类号　TP302.8

Stripe Matching and Merging Algorithm-based Redundancy Transition for Locally Repairable Codes

DU Qingpeng[1], XU Yinlong[2] and WU Si[2]
School of Computer Science and Technology, University of Science and Technology of China, Hefei 230027, China

Abstract　Compared with traditional replication technique, erasure coding is another data redundancy mechanism with lower space overhead at the cost of high repair cost. Locally repairable code is a special kind of erasure code with low repair cost, which is widely deployed in big data storage systems. In order to accommodate itself to dynamic workload and varying failure rate of storage media, modern storage systems make better trade off between access performance and reliability for erasure coded data by means of redundancy transitions. A redundancy transition method, which selectively matches and merges stripes of specific layout, decouples data placement and redundancy transition, is proposed. Furthermore, it reduces the cross-rack network traffic by designing cost quantification and optimization model. In contrast to those algorithms that designing data placement, the proposed algorithm has almost the same performance but eliminates the constraints on data layout and can be run iteratively. Experiment results show that under the two common transition setups, compared with naive approach of random layout, the proposed algorithm approximates the theoretical optimum, mitigates network traffic by 27.74% and 27.47%, and shortens transition time by 39.10% and 22.32% respectively.

Keywords　Locally repairable codes, Redundancy transition, Fault-tolerant storage technique, Network traffic optimization, Distributed storage system

1　引言

大数据时代下,互联网、传统工业、金融等各行业每年产生的数据量都呈指数级增长,高效地存储数据至关重要。由于单台计算机有限的存储能力不足以匹配爆炸式增长的数据,采用横向扩展的方法,使用多台计算机组成一个分布式存储系统来存储海量的数据成为主流的解决方案。然而,分布式系统中单台计算机的故障、定期下线维护,或是网络的不稳定性导致节点不可访问是常见的情形。因此,容错是分布式存储系统必须具备的性质。传统的容错存储技术多副本存在空间开销过大的弊端,而纠删码在提供相同容错能力的同时,能极大地节省存储空间,在存储海量数据场景下逐渐取代了多副本[1]。

纠删码通过额外的编解码计算来换取空间的占用量。

到稿日期:2022-11-29　返修日期:2023-04-07
基金项目:国家自然科学基金(62172382)
This work was supported by the National Natural Science Foundation of China(62172382).
通信作者:许胤龙(ylxu@ustc.edu.cn)

首先，需要将数据切分成若干个数据块，再用编码矩阵对数据块进行编码运算得到若干块检验块，将数据块和检验块作为一个条带存储在不同的计算机上。当某些数据块或检验块因存储的节点故障而丢失时，可以通过条带中剩余的块进行解码计算，重新恢复出丢失的块，并将其存储到其他节点中。以参数为(k,g)的RS码[2]为例，系统对k个数据块进行计算，产生g个校验块，这$k+g$个块组成一个条带，当任一块丢失时，需要访问条带中k个块进行修复，因此纠删码的修复开销非常大。

局部可修复码[3]利用修复局部性减少了修复时需要读取的块。在传统纠删码基础上，它将数据块进一步分成若干较小的组，组内编码得到局部校验块，各个组称为局部校验组。单个数据块修复时，局部可修复码先尝试在局部校验组内进行修复，仅在局部校验组内不能修复时才进行全局修复。

随着存储设备的老化或升级、用户对数据读写负载的改变，存储系统需要动态调整容错能力和访问性能。在基于纠删码的分布式存储系统中，调整的方式是改变纠删码的参数，该过程称作冗余度转换。冗余度转换需要重新计算新的校验块，从而造成大量的网络传输。当用户为了提升数据的可靠性而需要容忍额外的一个节点故障时，至少需通过网络传输与原始数据同等规模的数据量。已有的冗余度转换机制在数据规模较大的情形下，会产生极大的网络流量开销。更糟糕的是，在多机架部署场景下，网络通信产生的是代价昂贵的跨机架网络流量。虽然存在优化网络流量的冗余度转换方法，但大多限定了冗余度转换的目标参数，且数据布局局限性过大。

鉴于当前已有研究工作存在的不足，本文以局部可修复码为研究对象，研究其在层次化网络结构下[4]的冗余度转换机制。从优化网络流量的角度出发，设计了一种基于条带配对合并的算法，能够有效降低局部可修复码冗余度转换过程的跨机架流量，且算法对数据布局限制小，可以推广使其适用于多种纠删码策略。

本文的贡献具体为以下几方面：

1)以局部可修复码为例，设计了基于条带配对合并、对多个条带间布局无限制、节省网络传输的冗余度转换算法，同时使得冗余度转换后，条带修复性能和容错性能较优。

2)实现支持上述算法的分布式存储系统，并开源以支持相关研究工作。

3)通过模拟和系统实验，与基于数据布局的冗余度转换方法进行对比，本文算法的性能近似，但限制少，通用性高。与随机布局的朴素算法进行对比，在两种冗余度转换场景下，本文算法将网络流量分别降低了27.74%和27.47%，将耗时分别缩短了39.10%和22.32%。

2 背景和动机

2.1 背景

局部可修复码(Locally Repairable Codes，记为LRC码)是一种针对修复性能进行优化的纠删码[3]。具体而言，相对具有k个数据块、g个全局校验块的RS码，它将数据块分成l个局部校验组，组内编码得到共l块局部校验块，与k块数据块和g块全局校验块共同构成参数为(k,l,g)的LRC码。LRC码在数据块丢失时仅需要读取同一个局部检验组的数据块进行解码计算。如图1所示，在参数为(6,2)的RS码条带的基础之上，D_0，D_1，D_2数据块编码计算得到一块局部校验块L_0，构成了一个局部校验组。同理，D_3，D_4，D_5与L_1也构成了一个局部校验组。此时，数据块D_0-D_5、局部校验块L_0-L_1、全局校验块G_0-G_1组成一个参数为(6,2,2)的LRC码条带。当某个节点发生故障时，仅需要与同一局部校验组剩余的3个节点交互即可解码恢复。这对于数据块数目与全局校验块数目之比很大的宽条带[4]，能极大提升修复性能。

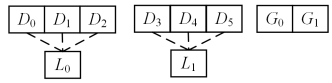
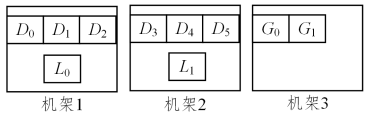


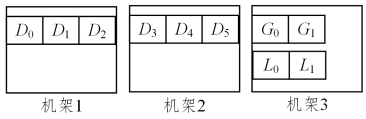
图1 参数(6,2,2)的局部可修复码

Fig. 1 Locally repairable code of parameter(6,2,2)

现代分布式存储系统通常部署在层次化的网络拓扑结构下[5]，根据系统规模不同，从底向上可包括存储节点、机架、机房、数据中心、城市等多个级别，其特征为故障发生率和网络带宽均呈现由高到低的趋势。因此，在典型的两层结构下部署局部可修复码时，为使数据能够在机架故障时被修复，且修复时产生的跨机架网络通信最小，需要对数据布局进行设计。图2(a)给出了局部可修复码条带布局，虽然此时系统能容忍任一节点的故障，但是若整个机架1故障，由于丢失的数据块数目超过了局部可修复码的容错上限，此时便不能正常解码恢复。与之相对，图2(b)给出了条带布局，不仅能在单节点故障时读取同一局部组的剩余数据进行修复，而且在整个机架故障时，仍然能恢复丢失的数据。具体地，机架3故障时，所有数据块仍可正常访问，系统可重新编码生成局部校验块和全局校验块。而当机架1或机架2中任一机架故障时，可通过读取机架3中的两块全局校验块以及故障机架所属局部校验组中的一块局部校验块进行解码修复。



(a) 无机架容错性质的数据布局



(b) 可机架容错的数据布局

图2 两种数据布局的机架容错性质

Fig. 2 Rack fault tolerance of two data layouts

2.2 相关工作和动机

2.2.1 相关工作

以多副本容错的存储系统为研究对象的工作中，PaRS[6]根据数据热度的变化，动态调节数据块的副本数目，以优化内存占用和访问效率。在热度变化之前，ARM[7]利用机器学习方法预测文件是否需要执行冗余度的调节。HACFS[8]的研究工作指出，为了应对动态变化的数据访问负载，系统需要相应地动态权衡纠删码的修复性能和存储效率，HACFS的

主要解决思路是根据数据读写操作的频率及最后访问时间决定纠删码条带是增加还是减少校验块数目，在读写次数达到一定阈值时进行冗余度转换。但这 3 项研究未考虑数据的冷热与数据的容错需求无关的情况。在 RAID 码应用的场景下常有动态插拔新存储盘的需求。因此，部署 RAID 码的存储系统需要具备良好的可扩展性。HP 的 AutoRAID[9] 能够自动地将数据在不同 RAID 级别间转换。DiskReduce[10] 通过异步编码，将多副本数据编码为 RAID 码，同时引入 RAID5 混合单副本的过渡状态，以实现双节点容错。HeART[11] 指出，大型存储系统需要应对动态变化的设备故障率，该研究针对纠删码和多副本，通过对可观测到的磁盘故障建立模型，构建了一套在线冗余度转换的框架，其优点是冗余度转换是动态的，对数据布局没有预先限制。上述研究都强调了冗余度转换的必要性，不足之处是均未对冗余度转换过程进行优化，产生的网络流量开销较大。

FastScale[12] 开创性地通过设计数据布局来优化 RAID 码伸缩过程中的数据块迁移流量。同时，还有一些优化多副本到纠删码转换过程的相关工作，以 EAR[13] 为例，它同样通过精巧的数据布局策略，使得多副本转换成纠删码条带过程中，跨机架的编码计算网络流量和数据块迁移流量最少。TEA[14] 与 EAR 的思路相同，但其进一步考虑了机架级别容错和负载均衡。另一个方向是研究不同纠删码参数之间的冗余度转换。典型地，ERS[15] 给出了 RS 码在层次化网络拓扑结构下进行高效冗余度转换的方法，一方面，该研究使用了一个较大的编码矩阵，复用了转换前的全局校验块计算结果；另一方面，则利用数据布局策略减少了数据块的迁移。2022 年 INFOCOM 的研究[16]（Optimal Placement For Locally Repairable Codes，下文简称 LRC-P）则首次针对性地研究了局部可修复码的高效冗余度转换，其创新之处是使用了数据布局算法结合条带合并的方法，极大地减少了校验块计算和数据块迁移产生的跨机架流量。

2.2.2 动机

上述相关的研究工作的总结如表 1 所列。

表 1 冗余度转换方法的总结

Table 1 Summary of redundancy transition methods

研究名称	研究内容	限制数据布局	优化流量
PaRS	多副本间转换	有	无
ARM	多副本转换到纠删码	无	无
HACFS	纠删码参数间转换	无	无
AutoRAID	RAID 码不同级别间转换	无	无
DiskReduce	多副本转换到 RAID 码	无	无
HeART	纠删码参数间转换	无	无
FastScale	RAID 码不同级别间转换	有	有
EAR	多副本转换到纠删码	有	有
TEA	多副本转换到纠删码	有	有
ERS	RS 码参数间转换	有	有
LRC-P	局部可修复码参数间转换	有	有

从表中可以发现，已有工作或是将冗余度转换自动化，牺牲了网络流量优化的机会；或是将冗余度转换的目标与数据布局耦合，限定冗余度转换的目标参数，最小化产生的网络流量，而其数据布局仅在该冗余度转换目标下才能达到最优。实际上，数据的可靠性需求随实际工作负载和存储介质的故障率是动态变化而非静态不变的。因此，数据布局前所作的假设可能与将来实际需求恰恰相反。另外，后文将介绍，即使假设成立，在一轮冗余度转换后的新条带，可能不再满足布局策略所定义的最优布局要求，反而是最坏的布局情况。

鉴于此，本文期望设计转换算法，使冗余度转换兼具通用性和高效性。首先，为实现通用性，算法将转换决策延迟到数据存储后动态发起且支持多次连续的冗余度转换，这意味着算法不能限制数据布局。其次，算法在转换后保持较好的修复性能和容错能力，以实现存储系统的高效性，并且通过最小化转换过程中的跨机架网络流量来实现算法本身的高效性。

3 目标与解决思路

3.1 设计目标

3.1.1 冗余度转换的目标可动态决定

如前文所述，用户在首次写入数据时，并不能预知数据未来的读写负载及可靠性需求，系统应当随读写负载的变化、存储介质的故障率变化而动态地对数据进行冗余度转换。然而，若根据读取频率或访问时间决定数据的可靠性需求，该决策可能与实际用户需求相悖。最恰当的方式是将冗余度转换的具体需求延迟，由用户动态发起请求。

3.1.2 冗余度转换应降低跨机架网络流量

在现代数据中心的架构下，相对机架或数据中心内部，跨机架或跨数据中心的网络带宽更为稀缺，过载比通常在几十的数量级[5]。因此，冗余度转换产生的网络流量开销主要是跨机架通信所致，算法设计应当尽可能减少跨机架网络的传输。

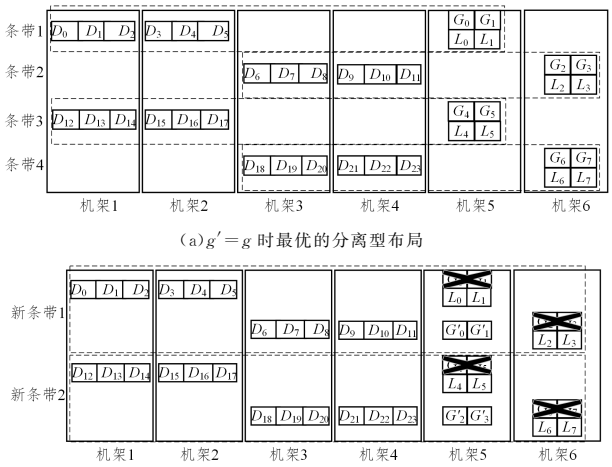
3.1.3 冗余度转换后保持较优的修复性能和容错能力

在层次化的网络拓扑结构下，局部可修复码的修复性能和容错能力存在矛盾。理想的冗余度转换方法需要转换后的数据在新纠删码参数配置下尽可能地保持较优的修复性能的同时，具备至少单机架级别的容错能力。

3.2 解决思路

3.2.1 多条带间采用随机数据布局策略

本文算法对多个局部可修复码条带间的相对布局不加限制，可采用随机布局策略。随机布局策略有以下优点：1）有助于负载均衡，提高存储设备利用率，也有利于读写并行化；2）它是一种通用的、无先验假设的布局策略，转换后的新条带仍可视作随机布局，冗余度转换可以迭代多轮，直至满足目标的冗余度需求。相反，LRC-P[16] 提出的最优布局策略，需要根据预设的转换目标将多条带相对分离或聚集地布局。然而，在一轮冗余度转换后，这种相对分散或聚集的布局假设可能不再成立，甚至与期望完全相反。如图 3(a) 和图 3(b) 所示，条带 1、条带 2、条带 3 和条带 4 均满足 LRC-P 中所设计的分离型布局，此研究证明该布局在第一轮转换后新条带的全局校验块数目 g' 与转换前条带全局校验块数目 g 相同时，即在 $g'=g$ 的情况下是最优的。虽然在第一轮合并时产生的跨机架流量最小，但是第一轮合并后的新条带 1、新条带 2 却呈现相对聚集的数据布局。若此时再进行第二轮转换，如果目标恰好是保持第二轮转换后全局校验块数目 g'' 与第一轮转换后全局校验块数目 g' 不变，则产生的跨机架流量反而是最大的。



(a)$g'=g$ 时最优的分离型布局

(b)合并后反而最差的聚集型新条带布局

图 3 合并前后的条带布局

Fig. 3 Stripe layout before and after stripe merging

3.2.2 采用条带配对合并算法

本文提出了冗余度转换优化的一种新思路。本质上,现有的设计数据布局的算法属于在数据存储前静态决定了冗余度转换目标和具体合并的条带配对方案。而本文所提的条带配对合并方法是在数据已经存储后,动态地根据不同的转换目标生成配对方案,由算法将各条带与其他所有条带合并时,将要产生的跨机架网络流量量化成数值代价,并进一步通过最优化模型最小化该代价。与 LRC-P,EAR,TEA,ERS 等工作相比,本文方法将最小化网络流量的时机从数据存储前延迟到了转换运行时,将利用数据布局最小化流量的方式从静态设计转换为动态选取,实现动态冗余度转换。这意味着条带配对合并算法可复用任何基于数据布局算法的策略,适用于不同参数、种类的纠删码,从而实现了算法的通用性。

3.2.3 冗余度转换后通过迁移数据块保持修复性能和容错能力

条带配对合并算法通过将相同局部校验组的数据块存储在最少的机架中,来实现修复代价的最小化。同时,针对不同的转换目标参数迁移不同数目的数据块,使得单个机架故障后,丢失数据块数目不超过新参数配置下条带的容错限度,从而使其能够被解码恢复。

4 条带配对合并算法的设计

图 4 给出了条带配对合并算法的执行流程。算法总体分为两个阶段:第一阶段根据当前条带的布局,求解出最优配对方案;第二阶段由第一阶段输出的配对方案,执行条带合并的过程,得到新条带布局。注意到第二阶段的输出可作为第一阶段输入,因此算法可级联迭代运行。

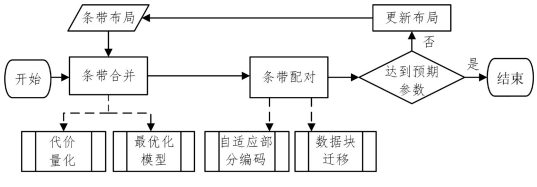


图 4 条带配对合并算法的执行流程

Fig. 4 Process of stripe matching and stripe merging algorithm

为了更好地理解第一阶段条带配对的设计,本章先介绍了第二阶段条带合并的过程。

4.1 条带合并框架

条带合并算法首先在宽条带生成[17]的相关研究中被提出。本文推广了该算法,使其适用于任意数据布局的纠删码条带。本文考虑两种常见的条带合并设置:合并后新条带的全局校验块数目 g' 保持不变($g'=g$)和 g' 加倍($g'=2g$)。

条带合并算法的描述如算法 1 所示。

算法 1 条带合并算法

输入:两个待合并条带 stripe1 和 stripe2,新条带的全局校验块数目 g'(g 或 $2g$)

1. 初始化:C_i/* 初始化每一个机架集合 C_i,包含 stripe1 和 stripe2 存储于该机架中的数据块 */
2. $S_{g'}$←pick an empty C' and g' hosts in C' randomly /* 随机选取一个新机架和其中的 g' 个节点 */
3. for each non-empty C_i
4. AdaptivePartialEncode(C_i,g',$S_{g'}$)/* 执行自适应编码 */
5. DataMigrate(stripe1,stripe2,C_i,g') /* 执行数据块迁移 */
6. for each host s in $S_{g'}$
7. s performs aggregate encoding /* 执行聚合编码 */

输入是两个需要合并的条带编号以及新条带的全局校验块数目 g'。算法随机选取一个新机架并随机在其中选取 g' 个节点(见算法 1 的第 2 行),由它们计算新条带全局校验块。之后,算法对各集合运行 AdaptivePartialEncode 算法和 DataMigrate 算法,即分别对该集合执行自适应编码和数据块迁移(见算法 1 的第 3—5 行)。最后,由选取出的 g' 个节点分别进行编码计算,将自适应编码阶段得到的编码块进行聚合生成全局校验块(见算法 1 的第 6—7 行)。

条带合并算法的两个关键子过程是自适应编码和数据块迁移。自适应编码在机架内部自适应地进行一次部分编码(Partial Encoding),以此减少编码全局校验块产生的跨机架流量。数据块迁移负责迁移部分数据块,从而保持单机架容错性质。它们的实现分别如算法 2、算法 3 所示。

算法 2 自适应编码的实现

输入:源机架数据块所在节点集合 C_i,新条带的全局校验块数目 g'(g 或 $2g$),新条带全局校验块存储节点集合 $S_{g'}$

1. if $|C_i| > g'$
2. s_i←pick a worker in C_i as s_i
3. else
4. s_i←pick a worker in $S_{g'}$ as s_i
5. s_i downloads data blocks from hosts in C_i and partially encode as g' blocks
6. for each s in $S_{g'}$
7. s downloads corresponding partially encoded block from s_i

自适应编码根据集群 C_i 包含的数据块数目与 g' 的大小关系,确定工作节点 s_i:如果机架中数据块的数目大于 g',则从本机架中随机挑选一个节点作为工作节点;反之,则在目标机架中选择一个节点作为工作节点(见算法 2 的第 1—4 行)。工作节点下载所有相关的数据块并进行一次部分编码,将

结果缓存在本节点内存或磁盘中（见算法 2 的第 5 行）。最后，由目标机架 $S_{g'}$ 中的各节点，从该工作节点 s_i 中下载部分编码后的缓存块（见算法 2 的第 6－7 行）。

算法 3 数据块迁移的实现

输入：两个条带 stripe1 和 stripe2，源机架集合 C_i，新条带的全局校验块数目 g'

1. 初始化 r /* r 表示除 C_i 外，其他机架中可用于修复 C_i 中数据块的校验块总数 */
2. while $|C_i|>$r do
3. pick an empty $C_{j\neq i}$ randomly as C_j
4. migrate a local parity group having minimum data blocks from C_i to C_j

迁移某机架的数据块前，需要统计除本机架外，可用于修复本机架数据块的局部校验块和全局校验块总数（见算法 3 的第 1 行）。仅当该机架数据块数目超过了可修复限度时需要执行数据块迁移操作。为了尽可能保持迁移后条带的修复局部性，迁移的基本单元是存储在该机架中的局部校验组。同时，为了减少跨机架网络的流量，应当从最小的局部校验组开始，直到剩余数据块数目在可修复限度内（见算法 3 的第 2－4 行）。

图 5 为随机布局的两个（6，2，2）的 LRC 码条带执行合并算法的示意图。图 5 中机架 1、机架 2、机架 5 分别在机架内对数据块进行部分编码（虚线）后，由最终机架 3 中的节点跨机架读取，并进行全局校验块 G_0' 和 G_1' 的编码计算（实线）。同时，为了保持单机架的容错性质，机架 1 需要迁移出条带 1 的 3 块数据块（D_0，D_1，D_2）至机架 4。

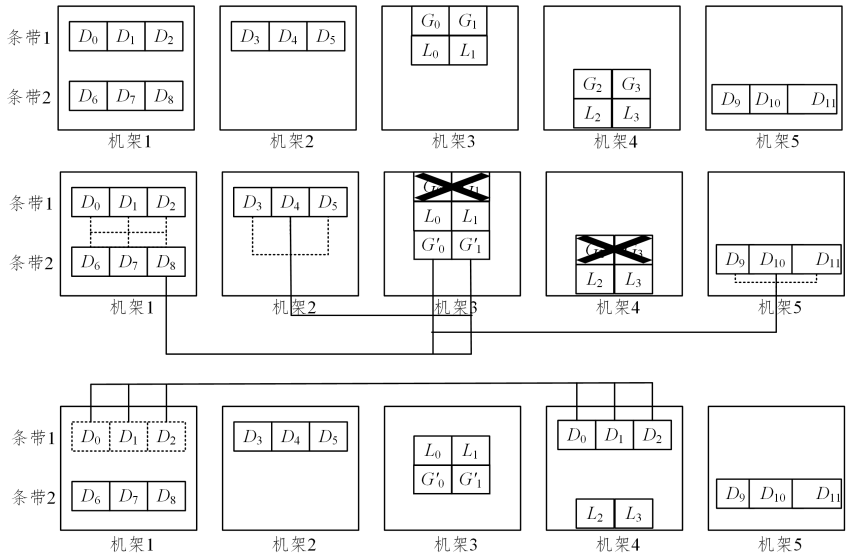


图 5 2 个随机布局的（6，2，2）LRC 码条带合并过程

Fig. 5 Stripe merging process of two(6，2，2) locally repairable codes stripes of random placement

4.2 代价量化方法

从条带合并的执行过程中可以发现，冗余度转换产生的跨机架网络传输主要来自两方面，分别是新全局校验块的编码计算和数据块迁移产生的流量。因此，代价量化方法主要度量这两部分流量的总和。具体而言，该方法计算这两条带数据块分布的所有集群中运行自适应部分编码算法而产生的跨机架传输的流量以及数据迁移流量。

对于编码产生的跨机架流量可由式（1）计算：

$$\sum_{C_i}\min(|stripe1.d|+|stripe2.d|,g') \tag{1}$$

其中，$|stripe1.d|$ 和 $|stripe2.d|$ 分别表示条带 1、条带 2 在机架 C_i 中的数据块数目。

当某机架故障后，其他机架中的校验块不能解码恢复丢失的数据块时，为了保证冗余度转换后新条带仍具备单机架的容错能力（见 3.1.3 节），需要迁移出一部分数据块。迁移数据块应当满足两个原则：1）尽可能使得迁移的数据块同属于一个局部校验组，且迁往相同的目标机架，以此保持解码修复时，可以充分利用局部可修复码的修复局部性；2）迁移出的数据块数目要尽可能少，以此减少冗余度转换过程中跨机架的网络流量。在 $g'=g$ 的情况下，可以验证迁移出的数据块仅可能是存储于某一机架中，且属于合并前两条带中某一条带的数据块。综上，所有机架数据块迁移总代价可表示为式（2）：

$$\sum_{C_i} m_i, m_i=\begin{cases}\min(|stripe1.d|,|stripe2.d|), & g'<=|C_i|\\ 0, & g'>|C_i|\end{cases} \tag{2}$$

通过上述的代价量化方法，我们可以进一步将条带配对的问题转换为可用数值方法求解的最优化问题。

4.3 最优化模型

总结前文的分析建模过程，条带配对合并问题可形式化地做如下定义：

记冗余度转换的输入是 S 个编码参数为 (k,l,g) 的局部可修复码条带的数据布局 L，以及冗余度转换后的目标参数为 $(2k,2l,g')$，其中 $g'=g$ 或 $2g$，冗余度转换的输出是 $S/2$ 个转换后局部可修复码条带的新数据布局 L'。

任意一种配对方案 M 可以表述为：

$$M:=\{(s_i,s_j)\,|\,i\neq j,i=1,2,\cdots,S\} \tag{3}$$

式（3）中，s_i 与 s_j 表示选择的是条带 i 与条带 j 配对执行条带合并。对于每一个配对方案 M，记共产生跨机架网络流量为 $cost_{\text{total}}(M)$。因此，寻求冗余度转换的条带配对方案，等价于求解使得 $cost_{\text{total}}$ 最小化的最优化问题。

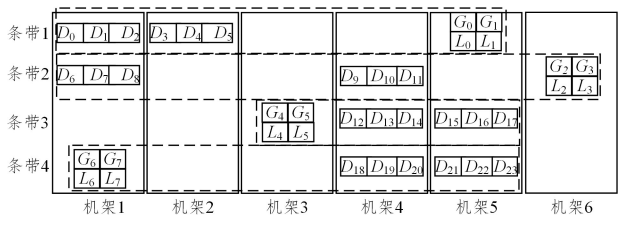


图 6 4 个参数为(6,2,2)的 LRC 码条带随机布局示意图

Fig. 6 Four(6,2,2) locally repairable codes stripes of random layout

可以进一步将其建模为图论中经典的最小完美匹配问题。为了更易于理解,我们将所有条带视为图中的节点,而连接两个节点的边的权重则通过本文前述的代价量化方法计算而得,它表示将这两个节点对应的条带配对合并时所产生的跨机架网络流量。而对任意两个条带均可以计算得到一个代价数值,这意味着,经过建模后所得到的图均为无向带权完全图。其邻接矩阵为:

$$\begin{pmatrix} w_{11} & w_{11} & \cdots & w_{1s} \\ w_{21} & w_{21} & \cdots & w_{2s} \\ \vdots & \vdots & \cdots & \vdots \\ w_{s1} & w_{s2} & \cdots & w_{ss} \end{pmatrix} \tag{4}$$

其中,W_{ij} 的计算式如式(5)所示:

$$w_{ij}=\begin{cases}\infty, & i=j \\ cost(stripei, stripej), & i\neq j\end{cases} \tag{5}$$

其中,$cost(stripei, stripej)$ 表示条带 i 与条带 j 配对合并时,代价量化方法求得的代价数值。

至此,我们将该问题转化为了一个数学模型。关于最小完美匹配问题,已经有许多高效的算法能够求解。典型地,如 blossom 算法,它的时间复杂度是关于条带数 S 多项式级别的,在目前大多数服务器上,并不会成为性能瓶颈。针对其细节,本文不作赘述,可参考文献[18]。

对于图 6 中的 4 个参数为(6,2,2)的局部可修复码条带的数据布局,假设需要进行 $g'=g$ 的冗余度转换,可计算得到其邻接矩阵。

$$\begin{pmatrix} \infty & 9 & 8 & 8 \\ 9 & \infty & 8 & 9 \\ 8 & 8 & \infty & 8 \\ 8 & 9 & 8 & \infty \end{pmatrix} \tag{6}$$

由 blossom 算法求解得到最小带权匹配,得到最优配对方案:

$$M_{\text{optimal}}=\{(s_1,s_4),(s_2,s_3)\} \tag{7}$$

该方案表示条带 1 与条带 4、条带 2 与条带 3 配对合并的方案。可以验证,按此方案产生的跨机架网络流量为 8 块,其中迁移的数据块为 0 块,编码为 8 块;同理可得条带 2 与条带 3 配对合并的跨机架网络流量为 8 块;故该方案的总代价为 16 块。作为对比,顺序两两配对的方案对应式(8)。

$$M_{\text{sequential}}=\{(s_1,s_2),(s_3,s_4)\} \tag{8}$$

可以验证其代价为 19 块,其中条带 1 和条带 2 产生编码流量 6 块,数据块迁移为 3 块,条带 3 和条带 4 则产生编码流量 4 块和迁移流量 6 块。若不使用 4.2 节中所提的条带合并算法,即进行朴素冗余度转换,则除了编码所需的 24 块外,为满足单机架容错性质,共需额外迁移 9 块数据块,总代价为更高的 33 块。通过对比发现,本文算法使得跨机架流量减少了 51.5% 。

5 实验与分析

本章将介绍基于条带配对合并的冗余度转换算法的模拟实验和系统实验。由于现有的开源分布式存储系统,如 Ceph[19],Azure[20]等仅以插件形式提供了基本的 LRC 码实现,但未提供自定义数据布局、自适应编码等高级功能,因此我们实现了条带配对合并算法以及实验所需对照组的各种冗余度转换算法,并定制化地实现了一个分布式存储系统,它支持局部可修复码的编解码、最优条带布局策略,且代码精简(约 5000 行核心代码),可扩展性强,开源代码仓库的网址为 https://github.com/adsl-lab/stripe-matching-merging.git。将随机布局下的条带配对合并算法(记作 random_match)与 LRC-P 中设计的最优布局(记作 opt_sparse 稀疏型布局,$g'=g$ 时最优)、紧凑型布局(记作 opt_compact,$g'=2g$ 时最优)、随机布局下朴素冗余度转换(记作 random_naive)这 3 组对照组进行对比。每一对照实验均包括 $g'=g$ 和 $g'=2g$ 两种情况。

5.1 模拟实验

5.1.1 不同局部可修复码参数下的对照实验

为了验证条带配对合并算法在不同的局部可修复码参数下均能够正常运行且有显著的优化效果,模拟实验在 10 个机架、100 个条带的条件下,分别对(4,2,2),(6,2,2),(8,2,2),(10,2,2),(12,2,3),(12,3,3)LRC 码参数记录在该参数下各对照组相对朴素冗余度转换、跨机架流量的优化比,结果如表 2 所列。

表 2 不同局部可修复码参数跨机架流量优化比

Table 2 Experiment results of different locally repairable code parameters

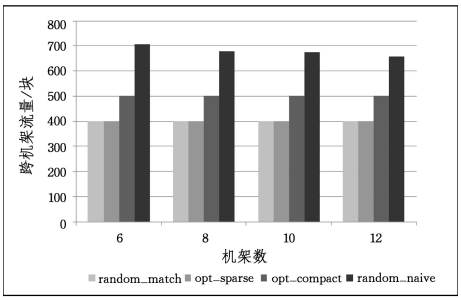
(单位:%)

$g'=g$ 或 $2g$ 对照组参数	$g'=g$				$g'=2g$			
	random_match	opt_sparse	opt_compact	random_naive	random_match	opt_sparse	opt_compact	random_naive
(4,2,2)	11.11	11.11	11.11	0.00	33.33	0.00	33.33	0.00
(6,2,2)	40.74	40.74	25.93	0.00	30.00	0.00	33.33	0.00
(8,2,2)	37.11	37.11	16.14	0.00	24.13	0.00	25.00	0.00
(10,2,2)	35.86	36.10	20.13	0.00	20.00	0.00	20.00	0.00
(12,3,2)	45.48	49.75	34.05	0.00	23.17	0.00	25.00	0.00
(12,3,3)	35.71	35.71	25.00	0.00	20.83	0.00	25.00	0.00

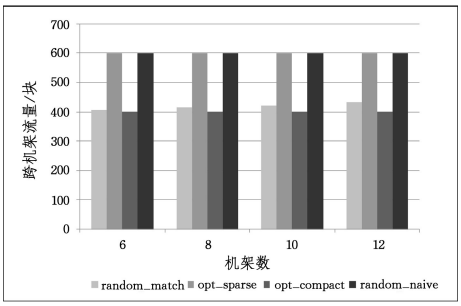
实验结果表明，能够同时在两种情况下均取得近似最优的效果的冗余度转换算法只有本文所提出的条带配对合并方法。通过两种情况的对比也可发现，基于最优数据布局的算法只能在其数据布局基于的假设成立时才能取得最优的跨机架网络流量，若实际冗余度转换的情况不同，则性能反而较差。

5.1.2 不同机架数下的对照实验

本次实验采用(6，2，2)参数的 100 个 LRC 码条带，在不同机架数条件下进行，以验证在各种规模的网络拓扑下，本文算法的跨机架流量指标，均能取得期望的优化效果。在 6，8，10，12 个机架下，本文算法在 $g'=g$ 均能取得与理论最优对照组 opt_sparse 相同的结果，在 $g'=2g$ 时，也能达到与 opt_compact 理论最优值 400 近似的结果，分别为 406，414，420，432。实验结果如图 7 所示。



(a)$g'=g$



(b)$g'=2g$

图 7 不同 g'值下不同机架数产生的跨机架流量

Fig. 7 Cross-rack traffic with different numbers of racks for different g'

5.2 系统实验

5.2.1 实验配置

由于实验所需的机架和服务器数目较多，本文参照 Hadoop HDFS 中伪分布式的部署方式，通过单个服务器上的多个进程来模拟单个机架中的多个物理节点。同时，由于配置千兆网卡(1 Gb/s)的服务器间网络通信带宽与服务器本地 SSD 盘的读写带宽(读 2GB/s，写 1GB/s)比例接近实际数据中心的过载比，因此用多个服务器模拟多机架的层次化网络环境是合理的。

系统实验中模拟机架设置为 6 个，局部可修复码参数为(6，2，2)，条带数为 100，条带中数据块大小为 64MB。实验指标选取冗余度转换的总耗时以及网络流量。对于网络流量，实验中使用 linux 下的 iftop 工具来统计网卡的流量。

系统实验中使用的服务器配置如表 3 所列。

表 3 服务器配置

Table 3 Server configuration

服务器组件	配置说明
OS	Ubuntu 18.04.1 LTS
CPU	Intel(R) Core(TM) i7-10700 CPU @ 2.90GHz
DRAM	64 GB DDR4(2666HZ)
SSD 读写性能	读 2GB/s、写 1GB/s
网卡	1Gb/s

5.2.2 实验结果

表 4 列出了实验中各对照组转换所用的耗时及其相对 random_naive 的优化比。

表 4 各对照组耗时和优化比

Table 4 Time consumption and optimization ratio of different control groups

$g'=g$ 或 $2g$ 实验指标对照组	$g'=g$		$g'=2g$	
	耗时/s	优化比/%	耗时/s	优化比/%
random_match	3501	27.74	3794	27.47
opt_sparse	3320	31.48	5331	−0.02
opt_compact	4956	−0.02	3661	30.01
random_naive	4845	0.00	5231	0.00

从耗时结果看，random_match 相比 random_naive，其冗余度转换耗时减少了 27.74%($g'=g$)，27.47%($g'=2g$)，分别近似于理论最优值 opt_sparse 和 opt_compact。然而，基于最优数据布局的 opt_sparse 和 opt_compact 的耗时在与期望不符合的情况下会退化为朴素的转换方法。opt_sparse 在 $g'=2g$的情况下以及 opt_compact 在 $g'=g$ 的情况下的耗时与朴素方法 random_naive 相比无明显提升。

使用 iftop 工具统计出跨机架流量，并计算优化比，结果如表 5 所列。

表 5 各对照组跨机架流量和优化比

Table 5 Cross-rack traffic and optimization ratio of different control groups

$g'=g$ 或 $2g$ 实验指标对照组	$g'=g$		$g'=2g$	
	流量/GB	优化比/%	流量/GB	优化比/%
random_match	57.73	39.10	61.25	22.32
opt_sparse	52.06	45.08	79.85	−1.27
opt_compact	68.00	28.27	58.50	25.81
random_naive	94.80	0.00	78.85	0.00

跨机架流量的优化比如表 5 所列，本文算法相比朴素冗余度转换的优化比在 $g'=g$ 时为 39.10%，接近于最优值 opt_sparse。在 $g'=2g$ 的情况下，优化比为 22.32%，接近于最优值 opt_compact。同样可发现，opt_sparse 和 opt_compact 在与其布局预期的转换目标不同的情况下，均不如本文算法。

结束语 本文针对在层次化网络拓扑下部署的纠删码存储系统，分析了现有冗余度转换方法的利弊，并提出了一种基于条带配对合并的局部可修复码冗余度转换方法，它与数据布局解耦，使得冗余度转换更加通用，且其产生的跨机架网络流量在各种条件下都近似达到理论最优值。相比随机数据布局的朴素方法，在 $g'=g$ 及 $g'=2g$ 的情况下，跨机架流量分别减少了 39.10%和 22.32%，且耗时分别缩短了 27.74%以及 27.47%，接近于理论最优。

本文以条带配对合并算法为核心实现了冗余度的转换，重点讨论了常见的两种合并情况，$g'=g$ 和 $g'=2g$。在之后

的研究中，我们将探索更普遍的形式，即目标参数 g' 取 g 到 $2g$ 之间任意值的情况。同时，本文的条带配对合并算法适用于同构的两条带合并，在实际部署环境中，可能存在多种不同配置参数的条带需要合并的情况。因此，另一个重点研究的方向是将本文算法推广至异构的条带合并，甚至是多条带合并，而不仅仅是两条带合并的情形。

参 考 文 献

[1] WEATHERSPOON H, KUBIATOWICZ J D. Erasure coding vs. replication: A quantitative comparison[C]// International Workshop on Peer-to-Peer Systems. 2002: 328-337.

[2] 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.

[3] PAPAILIOPOULOS D S, DIMAKIS A G. Locally repairable codes[J]. IEEE Transactions on Information Theory, 2014, 60(10): 5843-5855.

[4] HU Y, CHENG L, YAO Q, et al. Exploiting Combined Locality for Wide-Stripe Erasure Coding in Distributed Storage[C]// 19th USENIX Conference on File and Storage Technologies (FAST 21). 2021: 233-248.

[5] WANG T, SU Z, XIA Y, et al. Rethinking the data center networking: Architecture, network protocols, and resource sharing[J]. IEEE Access, 2014, 2: 1481-1496.

[6] ZHOU P, HUANG J, QIN X, et al. PaRS: A popularity-aware redundancy scheme for in-memory stores[J]. IEEE Transactions on Computers, 2018, 68(4): 556-569.

[7] BUI D M, HUSSAIN S, HUH E N, et al. Adaptive replication management in HDFS based on supervised learning[J]. IEEE Transactions on Knowledge and Data Engineering, 2016, 28(6): 1369-1382.

[8] XIA M, SAXENA M, BLAUM M, et al. A tale of two erasure codes in HDFS[C]// 13th USENIX Conference on File and Storage Technologies (FAST 15). 2015: 213-226.

[9] WILKES J, GOLDING R, STAELIN C, et al. The HP AutoRAID hierarchical storage system[J]. ACM Transactions on Computer Systems (TOCS), 1996, 14(1): 108-136.

[10] FAN B, TANTISIRIROJ W, XIAO L, et al. DiskReduce: RAID for data-intensive scalable computing[C]// Proceedings of the 4th annual workshop on petascale data storage. 2009: 6-10.

[11] KADEKODI S, RASHMI K V, GANGER G R. Cluster storage systems gotta have HeART: improving storage efficiency by exploiting disk-reliability heterogeneity[C]// 17th USENIX Conference on File and Storage Technologies (FAST 19). 2019: 345-358.

[12] ZHENG W, ZHANG G. FastScale: Accelerate RAID Scaling by Minimizing Data Migration[C]// 9th USENIX Conference on File and Storage Technologies (FAST 11). 2011.

[13] LI R, HU Y, LEE P P C. Enabling efficient and reliable transition from replication to erasure coding for clustered file systems[J]. IEEE Transactions on Parallel and Distributed Systems, 2017, 28(9): 2500-2513.

[14] XU B, HUANG J, CAO Q, et al. TEA: A traffic-efficient erasure-coded archival scheme for in-memory stores[C]// Proceedings of the 48th International Conference on Parallel Processing. 2019: 1-10.

[15] WU S, SHEN Z, LEE P P C. Enabling I/O-efficient redundancy transitioning in erasure-coded KV stores via elastic Reed-Solomon codes[C]// 2020 International Symposium on Reliable Distributed Systems (SRDS). IEEE, 2020: 246-255.

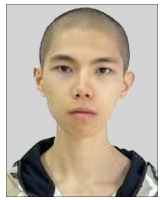
[16] WU S, DU Q, LEE P P C, et al. Optimal Data Placement for Stripe Merging in Locally Repairable Codes[C]// IEEE INFOCOM 2022 — IEEE Conference on Computer Communications. IEEE, 2022: 1669-1678.

[17] YAO Q, HU Y, CHENG L, et al. Stripemerge: Efficient wide-stripe generation for large-scale erasure-coded storage[C]// 2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS). IEEE, 2021: 483-493.

[18] KOLMOGOROV V. Blossom V: a new implem entation of a minimum cost perfect matching algorithm[J]. Mathematical Programming Computation, 2009, 1(1): 43-67.

[19] WEIL S A, BRANDT S A, MILLER E L, et al. Ceph: A scalable, high-performance distributed file system[C]// Proceedings of the 7th Symposium on Operating Systems Design and Implementation. 2006: 307-320.

[20] HUANG C, SIMITCI H, XU Y, et al. Erasure coding in windows azure storage[C]// 2012 USENIX Annual Technical Conference (USENIX ATC 12). 2012: 15-26.



DU Qingpeng, born in 1998, postgraduate. His main research interests include erasure coding and distributed storage system.

XU Yinlong, born in 1963, Ph.D, professor, Ph.D supervisor, is a member of China Computer Federation. His main research interests include storage system, graph computing and high performance computing.

（责任编辑：喻藜）