基于 XOR 的 TAR-CAU 数据更新方法

肖逸飞*，周世杰

(电子科技大学信息与软件工程学院 成都 611731)

【摘要】在基于纠删码的云存储系统中，数据更新的性能往往受到网络带宽的限制。鉴于此，提出了基于异或(XOR)的 TAR-CAU 数据更新算法，该算法基于以下设计原则：1) 利用数据更新量普遍较小的特点，将多条带的数据更新打包处理，减少网络往返次数，加快数据传输效率；2) 采用基于 XOR 的更新，提高编解码效率。仿真实验和本地集群实验结果表明，相比于 CAU 算法，在数据更新量较小时，TAR-CAU 算法能够提高至少 44% 的数据更新吞吐量。

关 键 词 云存储； 数据更新； 纠删码； 网络； 打包

中图分类号 TP393 **文献标志码** A doi:10.12178/1001-0548.2022156

TAR-CAU: An XOR-Based Data Update Scheme

XIAO Yifei* and ZHOU Shijie

(School of Information and Software Engineering, University of Electronic Science and Technology of China Chengdu 611731)

Abstract In an erasure-coded cloud storage system, the performance of data updates is often limited by network overhead. To end this, based on Cross-Rack-Aware Update (CAU) data update scheme, we propose an XOR based Tape ARchive Cross-Rack-Aware Updates (TAR-CAU) data update scheme. TAR-CAU is designed in terms of the two design primitives: 1) as the updates are small, we can pack several data blocks of data update into one block to reduce the number of network round trips, and accelerate the data transmission; 2) XOR-based data update scheme is used to accelerate data encoding and decoding. The simulation experiments and local cluster experiments show that, when the data updates are small, TAR-CAU can increase the data update throughput by at least 44% compared with the CAU.

Key words cloud storage; data update; erasure coding; network; tar

纠删码 (erasure codes)[1-2] 是云存储中一种较为先进的数据容错技术，相较于传统的多副本技术，采用纠删码提供数据冗余存储，会极大地降低系统的存储开销。如 QFS 文件系统 (qunantcast file system) 和 MapReduce 框架的数据存储后台采用纠删码进行冗余存储，比原来的 HDFS 采用多副本技术节省了 50% 的存储空间[3]。然而，纠删码也引发了 2 个新问题：数据修复[4] 和数据更新[5]。在数据更新中，由于纠删码提供的冗余校验数据是多个原始数据的线性变换组合，因此，当原始数据更新时，为了保证数据一致性，其校验数据也需要进行更新 (称为校验更新)。根据文献 [6-7] 提供的数据访问记录，可以得出以下两个结论。

1) 更新非常普遍，在大约 1.73 亿次的写请求中，超过 91% 的请求是更新数据；

2) 更新的数据量小，在所有的更新请求中，超过 60% 的更新小于 4 KB。

数据中心由成千上万个节点组成，其网络拓扑结构非常复杂，数据中心的数据更新性能往往受到网络的限制[8]，如何降低校验更新的网络开销是纠删码中亟待解决的问题。为了优化网络开销，国内外学者提出了很多数据更新方法。如 PUM-P 算法是利用更新管理器 (update manager) 计算数据变化 (delta 值)，并传输 delta 值给相关的校验节点进行更新[9]；PDN-P 算法摒弃更新管理器，直接通过数据节点计算并传输 delta 值到相关的校验节点[9]；T-Update 算法发现传统的数据传输模型是星型结构，不利于充分利用网络带宽，同时容易造成单点

收稿日期：2022－05－23；修回日期：2022－10－10
基金项目：四川省科技厅重大专项 (2020YFG0460)
作者简介：肖逸飞 (1989－)，男，博士生，主要从事数据存储方面的研究.
*通信作者：肖逸飞，E-mail: xyf1989@uestc.edu.cn

瓶颈，因此，将传输模型改为树型结构，增加网络并行度[10]。文献[8]提出CAU(cross-rack-aware updates)算法，将数据中心的各个存储节点按照机架(rack)分组，为了减少机架之间的网络开销，提出了2种可选的更新方式：

1) 校验增量更新(parity-delta update)，当数据机架(专用于存放数据节点)的更新量大于校验机架(专用于存放校验节点)的更新量时，选择将同一机架中的所有delta值都汇聚到一个数据节点(数据转发节点)，再由数据转发节点计算并转发校验更新给各个相关的校验节点；

2) 数据增量更新(data-delta update)，当数据机架的更新量小于校验机架时，分别将各个数据节点的delta值发送给同一个校验节点(校验转发节点)，再由校验转发节点计算校验更新并转发给其他校验节点[8]。

本文的主要目标是对数据更新的网络传输进行优化，基于CAU算法的思想，提出了改进算法—TAR-CAU，该算法针对更新数据量普遍较小的现象，借鉴tar打包原理，提出将同一个节点中的多个更新数据打包成一个块，再利用CAU算法更新，从而减少网络往返时间，降低发送端与接收端的更新处理频率，提高数据更新的效率。

本文的主要研究工作有以下3点。

1) 基于CAU算法，提出了TAR-CAU算法。该算法基于XOR进行数据更新，同时，利用更新数据量小的特点，将多个更新打包传输，从而减少网络往返次数，提高数据更新效率。

2) 实现原型系统。本文基于Go语言在Ubuntu 18.04平台实现了TAR-CAU原型系统，该系统包含中央控制器、算法调度器和节点代理的统一调度框架，不仅可以稳定运行TAR-CAU算法，同时，可以方便扩展并运行其他数据更新算法。

3) 验证算法的有效性。本文基于仿真实验和本地集群实验，利用微软剑桥研究院和哈佛NSR提供的真实数据集进行实验，与CAU算法进行了对比，从实验的结果来看，本文提出的算法能够有效提高数据更新吞吐量。

1 相关工作

1.1 纠删码概述

纠删码也称为纠错码，它将原始数据编码为数据量更大的编码数据，并能利用编码后的部分数据恢复出原始数据。纠删码一般需要指定n和k两个参数，用k份数据进行编码，产生n份数据。RS编码是最经典的一种纠删码[1]，图1为一个典型的RS(5, 3)的云存储系统($n=5,k=3$)，其中有3个数据节点和2个校验节点，每个节点中的数据都按照大小固定的块存储(块大小一般为1～64 MB)，编码后的数据块与校验块组成一个条带(stripe)，大多数数据更新算法都是按照条带顺序一条一条进行更新。图1展示了一个条带信息，其中，$d_{i,j}$表示数据块，$p_{i,j}$表示校验块，同一条带中属于同一节点的数据块或校验块称为条块(strip)[11]。

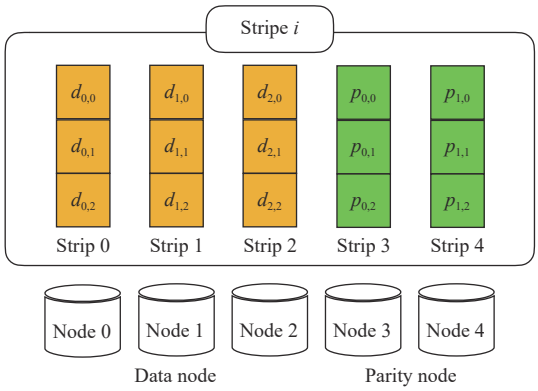


图1 RS(5, 3)云存储系统

1.2 数据更新

纠删码的数据更新主要有基于RS的更新和基于XOR的更新。

1) 基于RS的更新。基于RS的更新主要利用范德蒙德矩阵或柯基矩阵生成校验数据[3]，图2为图1的编码过程，其中，$d_i=(d_{i,0},d_{i,1},d_{i,2})$，$p_j=(p_{j,0},p_{j,1},p_{j,2})$。

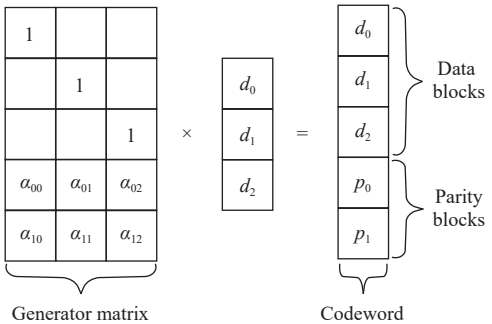


图2 RS (5, 3)编码过程

编码时，利用生成矩阵(generator matrix)左乘各个数据节点的数据向量(d_0,d_1,d_0)，生成码字(d_0,d_1,d_0,p_0,p_1)，其中，p_0和p_1为校验块，表示为：

$$p_j = \sum_{i=0}^{k-1} \alpha_{j,i} d_i \quad j \in [0, m-1] \tag{1}$$

当数据i更新时，用d_i'替换d_i，或在数据节点i中计算 delta 值 ($d_i' \oplus d_i$)，然后将 delta 值传输给p_i所在的校验节点，最后计算出最新的校验数据。本文参考的 CAU 算法就是基于 RS 的数据更新。

2) 基于 XOR 的更新。从式 (1) 可以看出，基于 RS 的更新会产生大量的乘法运算 ($\alpha_{j,i} d_i$)，消耗 CPU 资源。因此，可以利用有限域 (galois field) 将所有的乘法和加法运算转化为 XOR 运算[12-13]。

如图 3 所示，利用GF(2^3)的矩阵表示法可以将柯基矩阵转换为位矩阵 (binary distribution matrix, BDM)，GF(2^3)表示所有数据均用 3 位 2 进制数表示，数据范围为 0～7。

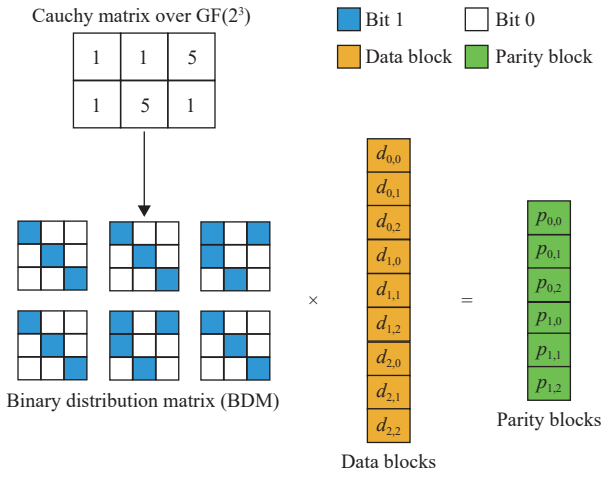


图 3　基于 XOR 的编码

图中，深色表示 1，白色表示 0，这样，所有的校验数据都可仅用 XOR 公式表示：

$$p_{0,0} = d_{0,0} \oplus d_{1,0} \oplus d_{2,0} \oplus d_{2,2} \tag{2}$$

$$p_{0,1} = d_{0,1} \oplus d_{1,1} \oplus d_{2,0} \tag{3}$$

$$p_{0,2} = d_{0,2} \oplus d_{1,2} \oplus d_{2,1} \tag{4}$$

$$p_{1,0} = d_{0,0} \oplus d_{1,0} \oplus d_{1,2} \oplus d_{2,0} \tag{5}$$

$$p_{1,1} = d_{0,1} \oplus d_{1,0} \oplus d_{2,1} \tag{6}$$

$$p_{0,0} = d_{0,2} \oplus d_{1,1} \oplus d_{2,2} \tag{7}$$

基于 XOR 的数据更新方法仅依赖简单的 XOR 运算，CPU 可以直接执行 XOR 操作，相比于乘法运算，XOR 运算效率更高，因此，数据更新效率也更高。同时，根据文献 [3] 研究表明，基于异或的编码方式更适合采用现代 CPU 的 SIMD 技术执行并行加速计算。如采用 AVX2 指令，可同时进行 256 位异或运算。本文在一台配有 4 核 2.2 GHz Intel CPU、16 GB DDR3 内存、1 TB SSD 存储的 Mac 操作系统环境中进行测试，发现采用 AVX2 指令、基于 XOR 进行数据更新，更新 1 MB 的数据仅需 20 ms，而采用基于 RS 的数据更新，需要 300 ms。因此，与 CAU 不同，本文选择基于 XOR 进行数据更新。

2　基于 XOR 的 TAR-CAU 算法

CAU 算法的核心思想可以用图 4 表示，如前所述，CAU 按照节点所在的机架进行分组，其中，R_i和R_j分别表示数据节点机架和校验节点机架，CAU 并不是实时处理每一个更新请求，而是设置了一个批处理阈值 (如 100)，当请求数量超过该阈值时，分条带批量处理更新请求。当同一条带中，R_i的数据更新量小于R_j的校验数据更新量时，采用 data update 方法，即分别将数据节点的 delta 值发送给某一校验转发节点，再由校验转发节点通过式 (1) 计算校验更新并传输给相关的校验节点，如图 4a 所示；相反，当R_i的数据更新量大于R_j的校验数据更新量时，采用 parity update 方法，即汇总所有的 delta 值到数据转发节点，再由数据转发节点利用式 (1) 计算校验更新并传输给相关的校验节点，如图 4b 所示。

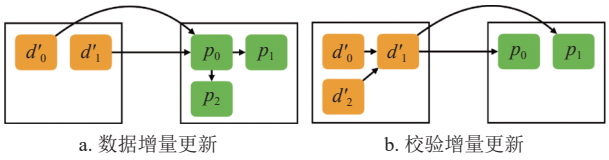


图 4　CAU 的 2 种更新方法

相比于传统的星型数据更新方式 (各个数据节点各自将数据传输给相关的校验节点)，CAU 利用汇聚节点的转发，确实可以减少网络开销，尤其是跨机架的网络开销。然而，数据更新性能可以进一步优化。

1) 采用打包更新。因为大部分的数据更新量小 (不超过 4 KB)，而一般设置的块大小为 1～64 MB。因此，在同一个机架中，如果同一节点有多个数据块进行了更新，可以采用 tar 打包模型进行校验数据更新。

2) 基于 XOR 进行更新。如前文所述，基于 RS 计算更新会产生大量的乘法运算，对于 CPU 资源开销较大，鉴于此，本文采用基于 XOR 的更新方法，在计算校验更新时，采用式 (2)～式 (7)，将

有限域乘法与加法运算都转化为 XOR 运算。

如图 5 所示，在同一数据节点中，可能有多个数据块被修改 (比如Node 0的 0 号、6 号数据块)，考虑到修改的数据量较小，本文采用 tar 打包的方式，如图 6 所示，将同一个节点的数据进行打包，增加一个头数据 (用 H 表示)，头数据中记录该块所在的条带编号、块内起始位置、修改数据大小等信息。

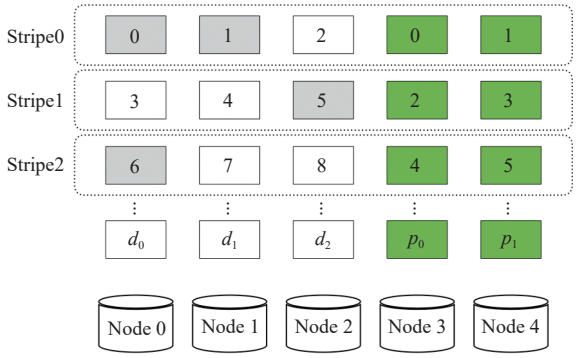


图 5 数据更新示例 (CAU 模型)

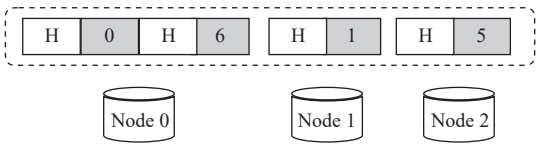


图 6 TAR-CAU 模型

如此，同一个块的大小可以容纳至少 200 个小更新 (假设块大小为 1 MB)，从而可在处理一个条带时，同时处理多个更新数据，然后按照 CAU 的传输模型进行数据更新。

同时本文发现在进行批处理更新时，同一个数据块 (如Node 0的 0 号数据块) 可能会被多次修改，但是每一次修改的位置和大小不一定相同。这就需要在进行 tar 压缩时，对同一个数据块的多次访问进行记录和合并。如图 7 所示，假设数据块大小为 1 MB，在进行批处理更新时，发现 0 号块有多次修改记录，于是，本文将多次修改的数据进行合并 (XOR)，同时找出其中的最小和最大范围 (rangeL 和 rangeR)，并将这 2 个值转化为块内起始位置和修改数据大小，记录到 H。

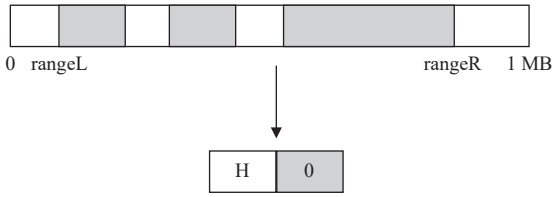


图 7 TAR-CAU 对同一块多次访问的合并处理

从图 7 可以看出，TAR-CAU 算法的优化空间是blockSize – (rangeR – rangeL)，优化代价是进行了打包与解包处理，增加了 CPU 开销。因此，TAR-CAU 算法性能对于数据访问有一定的依赖性，优化效果与具体的数据访问记录有关。

3 实验

3.1 仿真实验

本文通过微软剑桥研究院提供的真实数据集进行仿真实验，数据集记录了来自微软 13 个服务器，179 块磁盘中 36 个分区一周内的访问日志，每条记录包含访问时间、请求地址、访问数据大小等信息，本文随机选择了 3 个数据集进行仿真测试，仿真实验平台采用 Go 语言环境搭建，通过改变块大小、节点数量等参数，以平均更新时间、吞吐量为指标点，与 CAU 算法、基本更新算法 (简称 Base) 进行了比较。仿真环境如表 1 所示。

表 1 仿真环境

属性	值
CPU	Intel Core i7
内存	16 GB DDR3
磁盘	1 TB SSD
网络带宽/MB	100
操作系统	Mac OS
数据集	hm_0, hm_1, rsrch_1
仿真实验平台	自建(Go语言)

1) 平均更新时间

如图 8 所示，本文比较了 3 个数据集的平均单块更新时间，发现 TAR-CAU 算法更新效率最高，平均更新时间为 0.009 4 s，时间效率比 BASE 提高 54%，比 CAU 提高 30%。

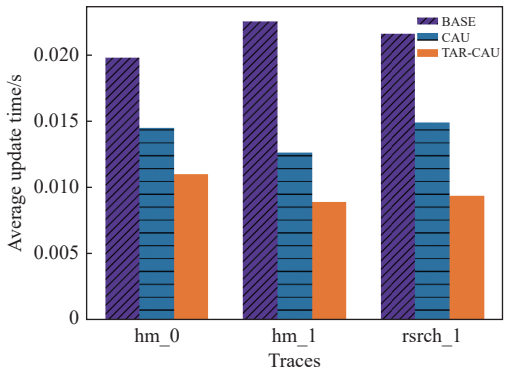


图 8 平均更新时间 (不同数据集)

2) 吞吐量

如图 9a 中，尽管本文提出的算法 TAR-CAU 需要在进行网络传输之前进行打包处理，增加了

CPU 开销，但是由于降低了网络传输的频率，因此，更新效率大大提高，在 3 种数据集中表现最佳，吞吐量比 BASE 提高了 119%，比 CAU 提高了 43%。

如图 9b 中，本文比较了常见的 RS(n, k) 配置：RS(4, 2)、RS(9, 6)、RS(16, 12)，机架容量 (rack size) 分别设置为 2、3、4。仿真结果表明，TAR-CAU 的吞吐量比 CAU 提高了 44%。

如图 9c 中，通过改变块大小 (block size) 的实验对比发现，TAR-CAU 的吞吐量提高了 60%。

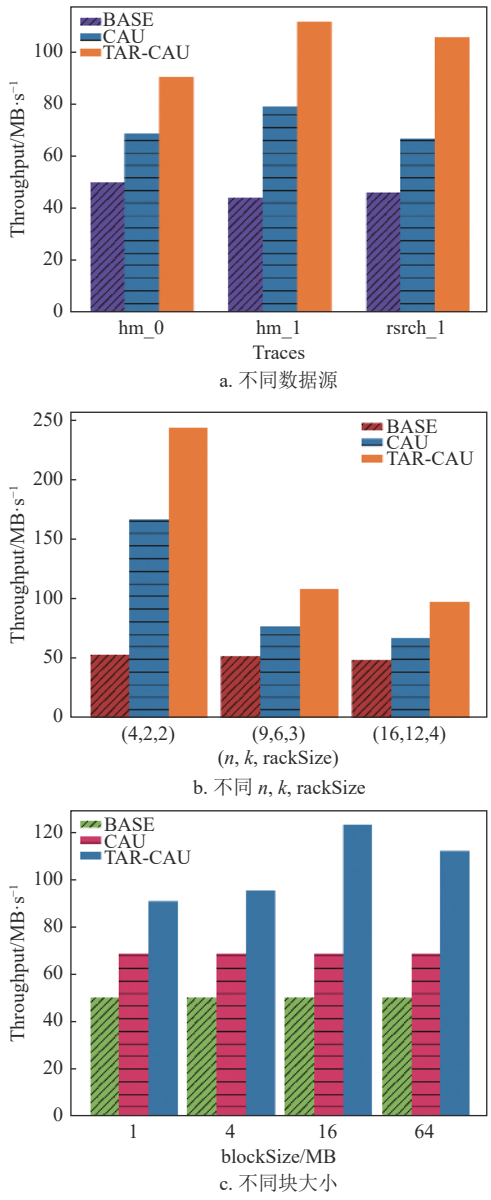


图 9　吞吐量

3.2　本地集群实验

本文采用 Go 语言在 Ubuntu 18.04 平台实现了基于 TAR-CAU 算法的原型系统，并在本地局域网搭建集群进行测试，以了解在较为真实的环境中 TAR-CAU 算法的性能。局域网内部署了 3 台服务器 (2 台华为 H12M-03，1 台华为 H22M-03)，利用 pve 虚拟管理平台[14] 搭建了虚拟机集群环境，共有 13 台虚拟机，网络拓扑结构如图 10 所示，主要由 3 个部分组成：中央控制器 (central controller)、算法调度器 (scheduler) 以及节点代理 (agent)。其中，中央控制器位于元数据服务器 (metadata server, ms) 中，负责整个流程控制，包括发送命令给节点代理、收集代理返回的响应 ACK、统计时间和跨域流量等；而算法调度器是整个系统的大脑，负责根据指定的算法指挥中央控制器进行调度，算法调度器也位于元数据服务器中；节点代理负责接受中央控制器的命令，执行相关任务，并返回给上级 ACK。

每台虚拟机的配置如表 2 所示。

为了模拟真实的网络环境，本文采用 Linux tc 工具[15] 进行网络带宽限制，设置机架内部带宽 1 Gbps，机架外部 200 Mbps，该设置可以根据具体的应用场景自行配置。与仿真实验一致，本文采用 hm_0、hm_1、rsrch_1 这 3 个数据集进行测试。

1) 平均更新时间

如图 11 所示，设置块大小为 4 MB，实验结果与仿真结果大体相同，TAR-CAU 算法表现最佳，平均单块更新时间为 0.454 4 s，相比 Base 节省约 78% 的时间开销，相比 CAU 节省近 70% 的时间开销。

2) 吞吐量

如图 12a 所示，设置块大小为 4 MB，通过 3 种数据集对比，在吞吐量方面，本文提出的 TAR-CAU 算法大大优于 Base 和 CAU，平均吞吐量比 Base 高出 9 倍，比 CAU 也高出 206%，这样的表现也是出乎意料。当然，值得注意的是，如前所述，TAR-CAU 的算法性能依赖于用户访问数据的位置，如果 rangeL 和 rangeR 过于靠近边界 (0 和 blockSize)，TAR-CAU 的算法表现甚至会略弱于 CAU。

图 12b 展示了不同块大小对各个算法的影响，当块大小较小时 (如 1 M)，CAU 算法略优于 TAR-CAU，原因可能是 rangeL 和 rangeR 过于靠近边界，导致打包产生的收益不如打包带来的额外开销。而随着块大小的逐渐增大，TAR-CAU 算法的优势愈发明显，尤其是在块大小达到 16 M 以上时，频繁 IO 的开销逐渐成为了性能的主导因素，所以，仅传输 delta 值的 TAR-CAU 算法表现更佳。

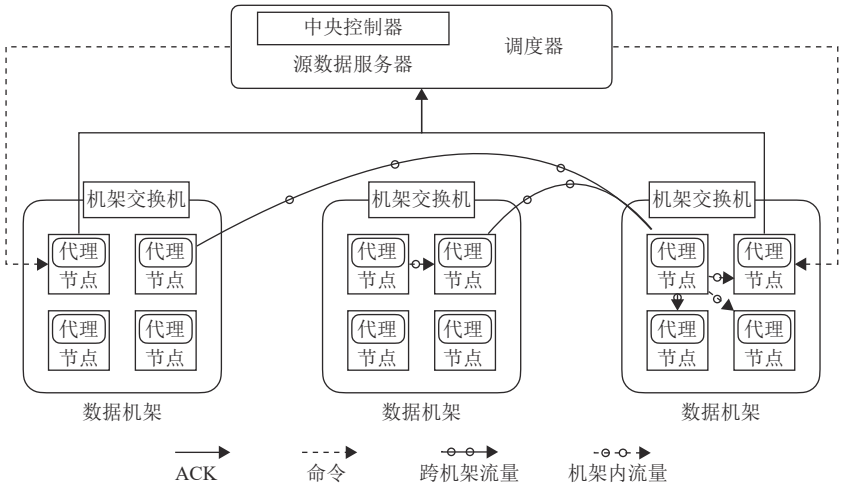


图 10 本地集群网络拓扑

表 2 虚拟机配置

属性	值
CPU	2核
内存/GB	2
磁盘/GB	32
带宽/GB·s^{-1}	1/0.2
操作系统	Ubuntu 18.04
数据集	hm_0, hm_1, rsrch_1

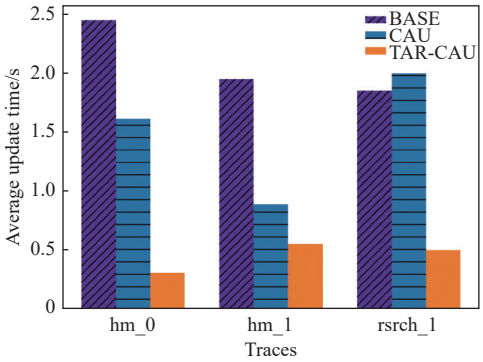
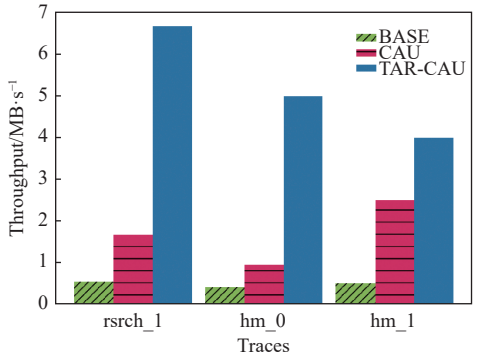
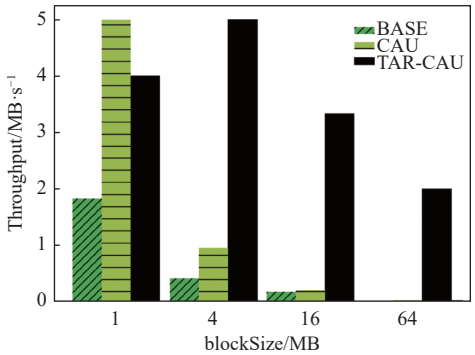


图 11 平均单块更新时间



a. 不同数据源



b. 不同块大小

图 12 吞吐量

由于引入了打包和解包过程，因此，TAR-CAU 算法相比于 CAU 算法会产生额外的 CPU 开销，如图 13 所示，设置数据块大小分别为 1、4、16、64 MB，同时随机产生 100 个数据块更新，每个数据更新的大小都不超过数据块大小的1/100，本文在一台虚拟机中测试打包与解包性能发现，打包与解包时间均不超过 300 ms。因此，对于整体的数据更新性能影响可以忽略不计。

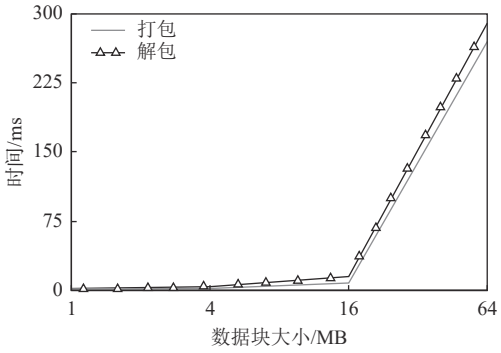


图 13 打包与解包计算时间

4 结 束 语

为解决云存储中数据更新的网络瓶颈，本文针

对数据更新的网络传输进行了优化，基于 CAU 算法，提出了 TAR-CAU，并针对数据更新量普遍较小的现象进行优化，将同一节点的多条带更新数据打包到同一条带进行处理。仿真实验和本地集群实验均表明，相比于 CAU 算法，当数据更新量较小时，本文的 TAR-CAU 算法能够至少提高 44% 的数据更新吞吐量。

参 考 文 献

[1] REED I S, SOLOMON G. Polynomial codes over certain finite fields[J]. Journal of the Society for Industrial and Applied Mathematics, 1960, 8(2): 300-304.

[2] NACHIAPPAN R, JAVADI B, CALHEIROS R N, et al. Cloud storage reliability for big data applications: A state of the art survey[J]. Journal of Network and Computer Applications, 2017, 97: 35-47.

[3] ZHOU T, TIAN C. Fast erasure coding for data storage: A comprehensive study of the acceleration techniques[J]. ACM Transactions on Storage (TOS), 2020, 16(1): 1-24.

[4] 傅颖勋, 文士林, 马礼, 等. 纠删码存储系统单磁盘错误重构优化方法综述[J]. 计算机研究与发展, 2018, 55(1): 1.
FU Y X, WENG S L, MA L, et al. Survey on single disk failure recovery methods for erasure coded storage systems[J]. Journal of Computer Research and Development, 2018, 55(1): 1.

[5] XIAO Y, ZHOU S, ZHONG L. Erasure coding-oriented data update for cloud storage: A survey[J]. IEEE Access, 2020, 8: 227982-227998.

[6] NARAYANAN D, DONNELLY A, ROWSTRON A. Writeoff-Loading: Practical power management for enterprise storage[J]. ACM Transactions on Storage, 2008, 4(3): 1-23.

[7] ELLARD D J. Trace-Based analyses and optimizations for network storage servers[D]. Cambridge, MA: Harvard University, 2004.

[8] SHEN Z, LEE P P. Cross-Rack-Aware updates in erasure-coded data centers[C]//Proceedings of the 47th International Conference on Parallel Processing. [S.l.]: IEEE, 2018: 1-10.

[9] ZHANG F, HUANG J, XIE C. Two efficient partial-updating schemes for erasure-coded storage clusters[C]// 2012 IEEE 7th International Conference on Networking, Architecture, and Storage. [S.l.]: IEEE, 2012: 21-30.

[10] PEI X, WANG Y, MA X, et al. T-Update: A tree-structured update scheme with top-down transmission in erasure-coded systems[C]//IEEE Infocom 2016 the 35th Annual IEEE International Conference on Computer Communications. [S.l.]: IEEE, 2016: 1-9.

[11] 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.
LUO X H, SHU J W. Summary of research for erasure code in storage system[J]. Journal of Computer Research and Development, 2012, 49(1): 1-11.

[12] BLAUM M, BRADY J, BRUCK J, et al. EVENODD: An optimal scheme for tolerating double disk failures in RAID architectures[C]//Proceedings of the 21st Annual International Symposium on Computer Architecture. [S. l.]: IEEE, 1994: 245-254.

[13] BLOEMER J, KALFANE M, KARP R, et al. An xor-based erasure-resilient coding scheme[R]. Berkeley, California: ICSI Technical Report No. TR-95-048, 1995.

[14] PROXMOX. Proxmox virtual environment[EB/OL]. [2022-04-28]. https://www.proxmox.com/en/proxmox-ve.

[15] KERRISK M. Tc(8)-Linux manual page[EB/OL]. (2021-08-27). https://man7.org/linux/man-pages/man8/tc.8.html.

编　辑　税　红