分　数：	
评卷人：	

華中科技大學

研究生（数据中心技术）课程报告

学　　号 M202273526

姓　　名 桂天炜

专　　业 计算机技术

课程指导教师 施展

院（系、所） 武汉光电国家研究中心

2022 年 12 月 30 日

纠删码在分布式存储中的性能优化

桂天炜[1)]

[1)](华中科技大学 武汉光电国家研究中心, 湖北省武汉市 435400)

摘 要 如今，各种工业数据中心为了保持较高的数据可靠性，都使用纠删码作为低成本的数据冗余方案，而不是采用简单的数据复制方案。纠删码方案相较于数据复制方案，虽然节省了很多的存储空间，但是数据存储和修复时的开销相对较大。对于分布式存储系统，数据更新、数据修复和数据传输策略影响着纠删码的性能瓶颈，本文调研了近几年在这些方向上做出的优化方案。Cheng 和 Hu 等人[1,2] 提出了一种新的基于奇偶校验日志的架构 HybridPL，创建了一个就地更新（针对数据和 XOR 奇偶校验块）和基于日志的更新（针对剩余奇偶校验块）的混合体，以便平衡数据更新性能和内存成本，同时保持高效的单次故障修复；Liu 等人[3] 提出了一种适用于纠删码分布存储系统的机架感知的流水线修复方法 RPR，它首次研究了机架的内部架构，探索了节点级和机架级之间的联系，以帮助提高数据中心发生单故障或多故障时的修复性能。

关键词 纠删码；对象存储；热数据；校验日志；分布式存储

Performance Optimization of Erasure Codes in Distributed Storage

Gui Tian-wei[1)]

[1)](Wuhan National Laboratory for Optoelectronic,Huazhong University Of Science and Technology,Wuhan,Hubei)

Abstract Nowadays, various industrial data centers are using erasure codes as a low-cost data redundancy scheme instead of a simple data replication scheme in order to maintain high data reliability. Although the erasure code scheme saves a lot of storage space compared to the data replication scheme, the overhead in data storage and repair is relatively large. For distributed storage systems, data update, data repair, and data transfer strategies affect the performance bottleneck of erasure codes, and this paper investigates the optimization schemes made in these directions in recent years. Cheng et al. propose a new parity log-based architecture HybridPL, which creates an in-place update (for data and XOR parity blocks) and a log-based update (for remaining parity blocks) in order to balance data update performance and memory cost while maintaining efficient single-failure repair; Liu et al. propose a rack-aware pipelined repair approach RPR for erasure code distributed storage systems, which for the first time investigates the internal architecture of the rack and explores the connection between the node level and the rack level to help improve data center single- or multi-failure The RPR approach is the first to investigate the internal architecture of the rack and explore the connection between the node level and the rack level to help improve the repair performance when single or multiple failures occur in the data center.

Key words Erasure coding; Object storage; Hot data; Parity logging; distributed storage

1 介绍

在分布式存储领域，数据可靠性是大家关心的问题，由于数据复制方案消耗大量的存储容量，最近的研究利用纠删码以更低的成本来保证可靠性。纠删码将原始数据块编码为奇偶校验块，通过数据块和奇偶校验块组成的数据条带可以在数据损坏时重建原始数据。对于许多大数据分析工作负载来说，数据更新往往是不可避免的，甚至是相当频繁的[4]（例如，读与更新的比例为 50%：50%），在纠删码存储中，数据更新代价很大，因为任何更新的数据块都会导致同一条带的所有奇偶校验块被更新。在最近的研究方向中，纠删码的更新被考虑在一个大规模的集群中（称之为宽条带），其中每个条带的尺寸都很大，并且其所有的小块都分散在几十个甚至几百个节点上。通过调研，目前有三种主要的方式来实现纠删码的更新：

1) 就地更新：用新的数据块和奇偶校验块替换旧的数据和奇偶校验块。

2) 全条带更新：把新的数据块直接编码为新的条带

3) 奇偶校验日志：将奇偶校验块的更新插入到日志节点。

在这三种方案中，就地更新要消耗大量的传输带宽，全条带更新要花费高额的存储开销，而奇偶

校验记录在更新过程中以低存储开销消除了奇偶校验块的读取，是理想性能最佳的。然而，现有的分布式存储中纠删码更新方案的研究只考虑了就地更新或全条带更新，而没有考虑奇偶校验日志，主要原因是基于磁盘的日志节点传输速率远低于内存节点，处理内存和基于磁盘的日志节点之间的性能差距是很有挑战性的。具体来说，即使奇偶校验日志可以减少更新过程中的数据传输，日志节点仍然会大大降低更新性能；此外，通过驻留在日志节点的奇偶校验块来修复故障，修复性能也会大大降低。在 Cheng 和 Hu 的论文[1] 中，他们提出了一种新的存储架构，称为 HybridPL，它采取了就地更新和奇偶校验日志的混合方式。首先，HybridPL 允许所有的数据和一些奇偶校验块在 DRAM 中执行就地更新，以实现快速的单次故障修复，同时保持低内存开销。其次，HybridPL 使其他奇偶校验块执行奇偶校验记录，以减少更新过程中的块传输，此外，HybridPL 开发了一个基于缓冲区记录的方案，以加速日志节点中奇偶校验块的更新。我们将上述架构实现为一个名为 LogECMem 的内存 KV 存储，并进一步在 LogECMem 之上设计了高效的多故障修复方案。这项工作的贡献包括：

- 通过工作负载观察到，现有的纠删码的内存 KV 存储的更新方案要么产生许多块传输，要么需要高内存开销，因此需要开发一个基于奇偶校验记录的方案来平衡更新和内存的成本。
- 通过可靠性分析表明，改善单次故障修复可以显著提高可靠性，因此提出了 HybridPL，它以较低的内存占用率执行就地更新，以实现快速的单次故障修复，同时利用奇偶校验记录以及缓冲区记录实现快速更新。
- 在 HybridPL 之上建立了一个内存 KV 存储 LogECMem，它实现了基本请求（包括单次故障修复）、基于奇偶校验日志的更新和缓冲区日志，还通过批量合并奇偶校验更新来改进缓冲区日志。
- 在 LogECMem 之上设计了高效的修复方案，用于多故障修复，比最先进的基于奇偶校验记录的修复方案减少了磁盘 IO，同时仍然保持高修复性能。

在现代数据中心中，存储节点通常被划分到不同的机架中，以提供机架级容错。同一个机架内的节点通过一个机架顶部 (T0R) 交换机连接，多个机架通过聚合交换机连接。为了恢复一个机架内的数据节点，需要传输多个数据节点，要么在同一个机架内，要么跨越多个机架。根据 Facebook 的数据，为了恢复数据，每天通过机架顶部交换机传输的数据位数超过 180TB。此外，机架间带宽通常为 1Gb/s，而机架内带宽为 10Gb/s，这意味着跨架带宽是一种比较稀缺的资源。因此，虽然 RS 代码提高了存储效率，但它导致数据中心的磁盘和网络流量显着增加，具体来说，它极大地消耗了宝贵的跨机架带宽。此外，传统的 RS 码在 Repair 时会向恢复节点和机架传输非常大的数据量，使得数据中心负载不均衡，进一步削弱跨机架的数据传输性能。Liu 等人[3] 提出了一种纠删码分布式存储系统中基于机架感知的流水线修复算法 RPR, 当系统发生单故障或多故障时，该算法能够显著减少跨机架的数据传输次数和修复时间。从负载平衡方面来看，RPR 对数据中心也是有益的。RPR 机制包括三种技术：

1) 数据奇偶校验放置：生成奇偶校验块后，按照一定的规则将其放置在数据架上，以提高解码速度；
2) 机架内部分解码：在数据块传输到恢复节点/机架之前，先进行部分解码，减少了跨机架的数据传输，提高了负载均衡；
3) 跨架流水线修复：在部分解码后，修复流水线算法调度机架内和跨架传输的序列，以实现最优的总修复时间。

他们的贡献总结如下：

- 在纠删码分布式存储系统中，提出了一种新的修复方案 RPR, 该修复方案能够容忍多个故障，并显著减少了跨机架数据传输量和当故障发生时的总修复时间。
- 提出了一种基于流水线的贪婪修复算法，该算法可调度般 RS(n,k) 码机架内解码/传输和机架间解码/传输的最优修复流水线。
- 对 RPR 方案进行数学分析、模拟器评估和真实世界评估。实验结果表明，对于单块故障，RPR 比传统修复方法减少了 81.5% 的修复时间，比 CAR 减少了 50.2%。对于多分组故障，与传统的 RS 编码修复相比，RPR 可减少最多 64.5% 的修复时间和最多 50% 的数据传输流量。

2　背景和调研

2.1　纠删码入门知识

一个 (k,r) 码表示信息位为 k 位、校验位为 r 位的码，信息位表示为 $D_i(1 \leqslant i \leqslant k)$，校验位表示为 $P_j(1 \leqslant j \leqslant r)$。校验位通过对数据位进行线性运算得出：$P_j = \sum_{i=1}^{k} \alpha_j^{i-1} D_i$，其中 $\alpha_j^{i-1}(1 \leqslant i \leqslant k, 1 \leqslant j \leqslant r)$ 是编码系数。例如，对于一个 $(3,1)$ 码，校验位为 $P_1 = D_1 + \alpha_1 D_2 + \alpha_1^2 D_3$ 基于 RS 码的定义，有以下定义：

- XOR 奇偶校验块：指每个条带的第一个奇偶校验块（P_1）往往是所有数据块的 XOR（$\alpha_1 = 1$），这可以简单地帮助修复一个数据块，例如 $D_1 = P_1 - D_2 - D_3$
- *Delta*：指的是新旧数据块之间的变化，例如当一个旧的数据块 D_1 被更新为一个新的数据块 D_1' 时 $\Delta D_1 = D_1' - D_1$
- *Parity delta*：指的是新旧奇偶校验块之间的变化，例如当一个旧的奇偶校验块 P_1 被更新为一个新的奇偶校验块 P_1' 时 $\Delta P_1 = P_1' - P_1$

注意 RS 码是基于线性编码的，所以当更新数据块时，可以发现有以下两个属性：

1. 奇偶校验值可以从 *Delta* 中产生：同一条带的所有奇偶校验块的奇偶校验值可以根据同一个 *Delta* 来计算。例如对于一个$(3,2)$码，当 D_2 被更新为 D_2' 时，我们可以计算出 $\Delta P_1 = P_1' - P_1 = (D_1 + \alpha_1 D_2' + \alpha_1^2 D_3) - (D_1 + \alpha_1 D_2 + \alpha_1^2 D_3) = \alpha_1 \Delta D_2$，同理可以计算出 $\Delta P_2 = P_2' - P_2 = (D_1 + \alpha_2 D_2' + \alpha_2^2 D_3) - (D_1 + \alpha_2 D_2 + \alpha_2^2 D_3) = \alpha_2 \Delta D_2$。$\Delta P_1$ 和 ΔP_2 共享相同的 ΔD_2
2. 对于同一条带的多个数据块更新，一个奇偶校验块的对应 *Parity delta* 可以减少到 1 个。例如对一个（3，1）码，当 D_1 首先被更新为 D_1'，然后 D_2 被更新为 D_2' 时，我们可以计算出 P_1 对应的两个 *Parity delta* 分别为 $D_1' - D_1$ 和 $\alpha_1(D_2' - D_2)$，它们可以结合为 $D_1' - D_1 + \alpha_1(D_2' - D_2)$，这样最新的奇偶校验块可以通过 P_1 和 $D_1' - D_1 + \alpha_1(D_2' - D_2)$ 计算出来

2.2　奇偶校验更新方案

纠删码存储中的奇偶校验更新会产生大量的网络传输，因为它们需要重新计算奇偶校验块以保证数据一致性。通过调研现有的奇偶校验更新方案，将它们分为四类：直接重建、就地更新、全条带更新和奇偶校验日志更新。

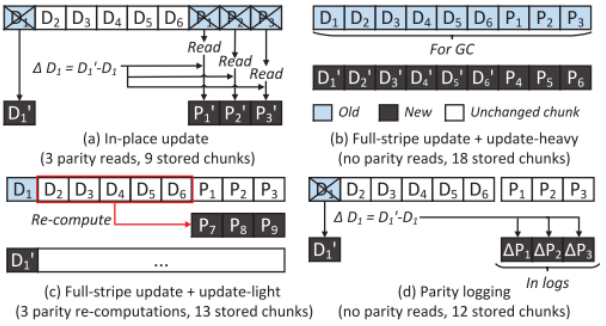


图 1　不同更新方案的对比

- 直接重建：一个直接的方法是首先读取所有没有参与更新的数据块，然后用读取的数据块和新的数据块重建新的奇偶校验块，这显然需要花费大量的数据块传输。
- 就地更新：就地更新读取旧的奇偶校验块，通过 *Delta* 来计算 *Parity delta*，通过 *Parity delta* 生成新的奇偶校验块，并替换掉旧的奇偶校验块。显然，就地更新会产生额外的奇偶校验块的读取，这阻碍了更新的性能。
- 全条带更新：为了消除对奇偶校验块的读取，全条带更新直接将新的数据块编码到新的条带中，并将旧的数据块标记为无效，从而可以通过垃圾收集（GC）直接释放它们。它已经被部署在实际的系统中，例如，QFS、BCStore 和 Giza。然而，全条带更新可能会产生很高的存储开销来存储陈旧的数据块，并且在 GC 期间需要对剩余的活跃的未改变的数据块重新计算奇偶校验码。
- 奇偶校验记录：奇偶校验记录（PL）通过在日志设备中记录奇偶校验差值来改进就地更新，从而减少更新期间旧奇偶校验块的读取开销，同时保持比全条带更新更低的存储开销。

2.3　传统修复方案

给定一个 $RS(n,k)$ 代码，为了确保单架容错，每个机架最多可以包含来自同一条带的 k 个节点。为了简化讨论，假设每个机架包含相同数量的 k 个节点，同一条带中的 $n+k$ 个节点分布在 q 个机架上。图2显示了一个传统 $RS(4,2)$ 码修复过程的例子。四个原始数据块 (d0, d1, d2, d3) 和两个奇偶校

验块 (p0, p1) 均匀地分布在三个机架 (r0, r1, r2) 上。假设节点 $d1$ 发生故障，选择节点 (d0, d2, d3, p0) 来恢复它。在传统的解码过程中，四个被选中的可用节点被转移到恢复节点/机架上。假设一次跨架转移需要 t_c 时间，那么恢复节点收到所有必要信息并开始解码过程的时间为 $4 * t_c$。在图2中，时间步数 1 到 4 表示这个单节点故障恢复的总数据传输的步骤。

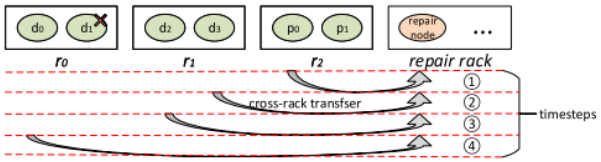


图 2 当单块故障发生时，传统 RS(4,2) 码修复过程的示例

假设跨架带宽为 ω，每个节点的数据为 B，解码速度为 δ。那么传统修复过程的总修复时间 t_{total} 是数据传输时间和解码时间之和：

$$t_{total} = 4 * \frac{B}{\omega} + 1 * \frac{B}{\delta}$$

在实践中，跨架带宽约为 1Gb/s（128MB/s），而 RS 码的解码速度约为 1000MB/s。显然，修复节点/机架的跨架带宽是整个修复过程的瓶颈。为了修复一个故障节点，需要将四个节点跨架传输到修复节点/机架；由于带宽较低，这消耗了大量宝贵的跨架带宽，并需要很长的时间。此外，所有的数据上传都发生在恢复节点/机架上，这使得系统的负载不平衡。

3 HybridPL 架构

Cheng[1] 的主要目标是将内存中的 KV 存储与奇偶校验记录结合起来以获得较高的修复和更新性能。HybridPL 重点关注以下目标：

1. 低内存开销：HybridPL 通过对数据块的就地更新，减轻了内存开销

2. 高效的更新：HybridPL 对非 XOR 奇偶校验块进行奇偶校验记录，并利用缓冲区记录日志节点的奇偶校验差值来加速更新

3. 高效的单次故障修复：HybridPL 在 DRAM 节点中保留 XOR 奇偶校验块，以确保降级读取的效率

3.1 对数据和 XOR 奇偶校验块进行就地更新

就地更新数据块：就地更新不会产生额外的内存开销，而全条带更新则会，因为后者在内存中保留了多个旧版本的数据块，这一点在更新量大的工作负载中表现得更为明显[4,5]。因此，与全条带更新相比，HybridPL 可以通过部署数据块的就地更新来减少内存开销。

基于 DRAM 的 XOR 奇偶校验块：修复单节点故障的节点的数据传输率对数据可靠性有很大的影响，也就是说，我们只能在 DRAM 中保留 XOR 奇偶校验块以及数据块来获得高性能的单故障修复。具体来说，当一个请求的数据块由于单一故障而不能直接读取，HybridPL 检索剩余的 k-1 个数据块和同一条带的 XOR 奇偶校验块，以解码 DRAM 节点内的请求数据块，这确保了单一故障修复的高性能。因此，HybridPL 实现了目标 1 和 3

3.2 非 XOR 奇偶校验块的奇偶校验记录

使用奇偶校验日志以消除奇偶性块的转移：HybridPL 通过奇偶校验日志更新除了 XOR 的其他奇偶校验块，这样 HybridPL 只需要将奇偶校验更新延迟存储到记录设备中，以更新奇偶校验块。显然，HybridPL 既不像就地更新那样读取旧的奇偶校验块以更新新的奇偶校验块，也不像全条带更新那样检索未改变的数据块以重新计算奇偶校验块。

使用缓冲区日志用于奇偶校验：然而，与 DRAM 节点相比，日志节点的低传输率将成为降低写（更新）性能的瓶颈。为了解决这个问题，cheng 引入了 RAMCloud 中提出的缓冲区日志方法，它为 DRAM 和磁盘分配了副本。如图3(a) 所示，RAMCloud 将数据存储在主服务器的 DRAM 中，而复制在备份服务器的磁盘上。在写操作过程中，主服务器将数据写入 DRAM 中，并将数据副本转发给所有的备份服务器。一旦副本被写入备份服务器的 DRAM 中，RAMCloud 就认为写操作已经完成，其中 DRAM 中的这些磁盘副本可以被异步刷新到磁盘上。这里需要注意，日志节点的异步 IO 可以加

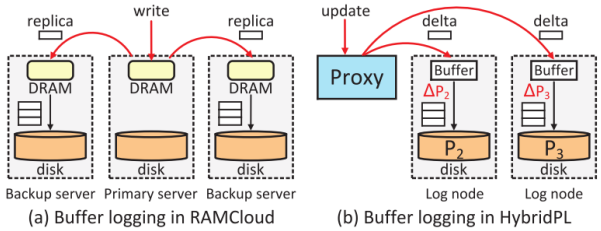


图 3 RAMCloud 和 HybridPL(6,3) 代码中的缓冲区日志

速写/更新操作，但是需要保持崩溃一致性，这样当缓冲区崩溃时，可以从磁盘日志中重建数据。cheng 将缓冲区日志也应用到 HybridPL 中，如图3(b) 所示。具体来说，在更新操作里，代理也可以向所有日志节点发送相同的 *delta*，类似于 RAM-Cloud 将相同的副本转发给备份服务器。然后，每个日志节点可以根据 *parity delta* 的第一个属性，通过 *delta* 计算其 *parity delta*。通过这种方式，HybridPL 可以进行快速的写入和更新操作，因为只要奇偶校验值被存储到日志节点的 DRAM 中就视为完成了操作，DRAM 中的奇偶校验值将被异步更新到磁盘上，因此，HybridPL 实现了目标 2。

3.3　基于 Lazy merging 的 PLM 方案

为了处理多块故障，除了使用 XOR 奇偶校验块，HybridPL 还需要使用更多的奇偶校验块，所以此时需要使用日志节点来进行多故障修复，但是 HybridPL 在使用日志节点修复的过程中会产生大量的磁盘 I/O 开销。因为 PL 可能会因为附加策略而使同一奇偶校验码的多个奇偶校验码分散，所以计算最新的奇偶校验码必须花费多个随机磁盘 IO。之前的一项工作研究了一种最先进的 PL 方案——带有保留空间的奇偶校验记录（PLR），该方案在计算最新奇偶校验块时，将 *parity delta* 保留在旧奇偶校验块的旁边，以减少磁盘寻址 IO。如图4所显示，每个奇偶校验块及其相应的 *parity delta* 被放置在连续的保留空间中，例如，对于奇偶校验块 $c+2d$，其 *parity delta*——$c'-c$ 和 $c''-c$ 一起放在磁盘中，从而确保 PLR 可以通过一次磁盘寻道计算最新的奇偶校验块 $c''+2d$，即 $(c+2d)+(c'-c)+(c''-c)$。然而，PLR 需要将奇偶校验码写入不同条带的特定保留空间，从而产生大量的随机写入磁盘 IO。如图 9(a) 所示，PLR 的 P_2 日志节点为了在两个不同的保留空间中写入奇偶校验码而产生了 8 次磁盘写入。因此，PLR 是用写（更新）性能换取修复性能。

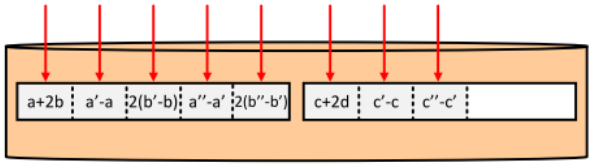


图 4　PLR 方案的 P_2 节点，产生 8 次磁盘写 IO

由于 cheng 的目标是为写（更新）和修复都提供高性能，因此传统的 PLR 方案无法达到他们的要求。因此，他们利用 *parity delta* 的第二个属性，通过简单地合并来增强 PLR（称为 PLR-m），它在写入磁盘之前合并内存中的同一条带的 *parity delta*。如图5所示，PLRm 产生了五个磁盘 IO。它合并了三次奇偶校验码，第一次合并后产生了合并的 *parity delta*——$a'+2b$ 和 $c+2d$，第二次合并后产生了 $2(b'-b)$ 和 $c''-c$，第三次合并后产生了 $(a''-a)+2(b''-b)$。很明显，PLR-m 只能合并紧密输入的 *parity delta*。由于内存空间的大小有限，PLR-m 只能紧密合并传入的 *parity delta*。为了解决这个问题，他们使用一个连续的磁盘空间来做懒惰的合并（称为带合并的奇偶校验记录，简称 PLM），它首先将奇偶校验数据按顺序写入一个额外的连续磁盘空间，并在以后读回这些数据用于合并相同条带的 *parity delta*，最后将合并的 *parity delta* 写入特定奇偶校验块的保留空间中。通过这种方式，PLM 可以比 PLR-m 合并更多的 *parity delta*，因为磁盘空间的大小更大，从而在更新过程中比 PLR 和 PLR-m 减少更多的磁盘 IO。如图6所示，PLM 只产生了四个磁盘 IO。它首先将包含所有奇偶校验块和 *parity delta* 的缓冲区按顺序冲入磁盘，通过一次连续的磁盘读取将它们读回来进行合并，并将两个合并的 *parity delta*——$a''+2b''$ 和 $c''+2d$ 写到特定的保留空间。

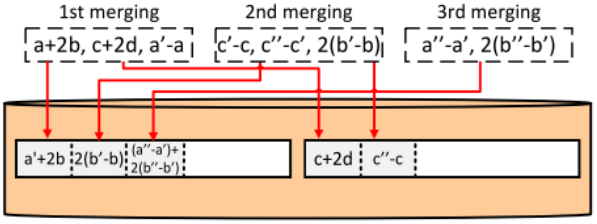


图 5　PLR-m 方案的 P_2 节点，产生 5 次磁盘写 IO

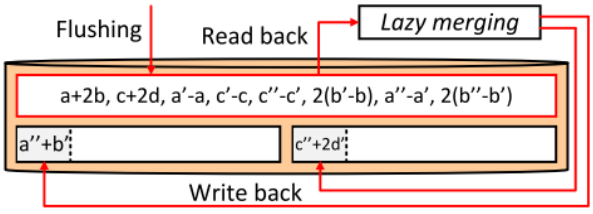


图 6　PLM 方案的 P_2 节点，产生 3 次磁盘写 IO 和 1 次读 IO

4　跨机架流水线调度设计

4.1　内架部分解码算法

为了进行跨机架流水线调度的设计，Liu 等人采用了一种名为部分解码的方法。部分解码利用了

RS 码的一个特性，通过该特性，数据通过线性组合进行编码和解码。传统方法为了修复一个数据块，所有相关的数据块都需要被转移到恢复节点/机架上。通过部分解码，同一机架上的节点可以先进行本地解码，然后生成一个中间编码块（与原始数据块的大小相同）。在这种情况下，每个机架的跨架数据传输最多只有一个块，由于内架带宽是跨架带宽的 10 倍左右，所以数据传输时间也可以大大减少。

图7显示了图2中的例子的修复过程，但是使用了内部机架进行部分解码。机架 r1 和 r2 首先并行地进行内部部分解码，而不是将四个块传输到修复节点/机架。这一次，timestep1 只需要一个内轨传输时间步骤 t_i。与传统的修复过程相比，其传输时间为 $4*t_c$，部分解码的传输时间仅为 $2*t_c+t_i$，其中 $t_c \approx 10*t_i$。以下是参与修复过程的每个机架的递归内架部分解码算法：

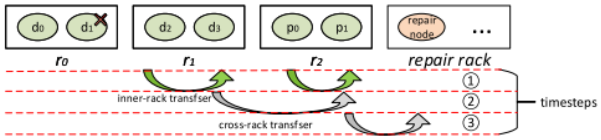


图 7　使用内部机架进行 RS(4,2) 码部分解码的例子

算法 1 Inner$(d_0, d_1, \cdots, d_{n-1})$

输入：参与修复过程的机架中的选定数据块 $(d_0, d_1, \cdots, d_{n-1})$

输出：一个中间块 I，将被跨架转移以进一步解码。

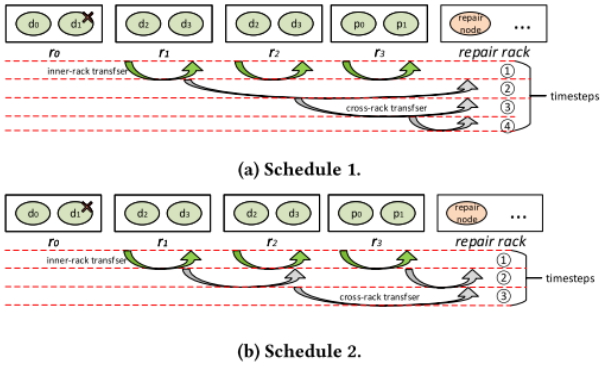
IF $n == 0$ **THEN**
  return I
END IF
IF $n\%2 == 0$ **THEN**
  FOR $a = 0 : \frac{n-2}{2}$ **DO**
    $i_a = d_{2a} \oplus d_{2a+1}$
  END FOR
  $Inner\left(i_0, i_1, \cdots, i_{\frac{n-2}{2}}\right)$
ELSE
  FOR $a = 0 : \frac{n-3}{2}$ **DO**
    $i_a = d_{2a} \oplus d_{2a+1}$
  END FOR
  $Inner\left(i_0, i_1, \cdots, i_{\frac{n-3}{2}} \oplus d_{n-1}\right)$
END IF

当修复过程开始时，为每个机架运行 Inner 算法，并将参与修复的节点作为输入。Inner 将节点分成若干对，并递归地执行部分解码，直到生成最终的中间块 I。在图7所示的例子中，在每个中间块生成后，将直接进行跨架转移计划。然而，当多个故障发生时，或有更多的节点参与修复过程时，内架和跨架转移时间表将更加复杂。原因是，当多个节点/机架发生多个故障时，不同节点/机架的内架部分解码的完成时间可能不同。为了开始跨架部分解码，必须首先由某些内架部分解码产生一些中间数据块。为了保持数据的一致性，以及充分利用内架和跨架传输的并行性，调度算法是必不可少的。

4.2　跨架调度算法

为了解决这个问题，Liu 提出了一种基于管道的贪婪算法，其目的是在整个修复过程中保持数据的一致性，并实现最佳的修复性能。

在图7的例子中，RS（4，2）代码的修复时间表看起来很简单。然而，当代码更加复杂时，内架和跨架数据传输可以选择各种修复时间表。考虑另一个简单的例子，RS（6，2）代码，如图8所示。图8a 和8b 显示了两种可能的内架和跨架传输时间表。为了最小化跨架传输量，我们选择机架 r1、r2 和 r3 中的节点来进行内架部分解码和跨架传输。在时间表 1 中，经过 timestep1，三个机架都完成了部分解码，并准备将中间数据发送到修复节点/机架。在不丧失一般性的情况下，r1 首先开始其跨机架传输。然后，由于修复机架正忙于接收来自 r1 的数据，机架 r2 和 r3 将不得不等待，直到传输完成。同样，在 r1 完成其跨机架传输后，r3 也需要等待，直到 r2 完成。对于时间表 2，类似地，在 timestep1 之后，所有三个机架都完成了部分解码并准备发送中间数据。不同的是，这一次，只有 r3 将首先被安排发送中间数据 I_3 到修复机架。同时，r1 将发送其中间数据 I_1 到 r2。在那里，I_1 和 I_2 将被进一步部分解码，形成中间数据块 I_{1+2}。这两个跨架传输将在 timestep2 中同时进行。在 timestep2 之后，中间数据块 I_{1+2} 将被跨架转移到修复架上，与 I_3 进行最后的部分解码，并获得重建的数据块 d_1。



(a) Schedule 1.

(b) Schedule 2.

图 8　两个不同时间表的 RS(6,2) 代码修复过程的例子

这两个时间表都有三个跨架传输，并使用三个时间步长；但是，它们的总修复时间有很大的差异。这是由于内架和跨架传输之间的高带宽差异造成的。如前所述，一次跨架传输的时间是 $t_c \approx 10 * t_i$，其中 t_i 是一次内架传输的时间。那么对于时间表 1，总的修复时间大约为 $3 * t_c + t_i$，或 $31 * t_i$。（这里忽略了解码时间，因为在这个近似计算中，它与数据传输时间相比很小）。对于时间表 2，在内架传输后，r1 和 r2 不等待 r3 完成跨架传输；相反，r1 立即将中间块发送给 r2，而 r3 立即将中间块发送给恢复架。在这种情况下，timestep2 中的两次跨架传输可以并行进行，从而避免了因等待而浪费的时间。因此，总的修复时间可以估计为大约 $1 * t_i + 2 * t_c$，或 $21 * t_i$。

算法 2 Cross(节点和机架间)

输入：　失效块及其对应的机架号; 每个机架中属于同一条带的节点及其对应的机架号

输出：　重构数据块 d_r

IF 恢复节点获取到所有中间数据 **THEN**

　Inner(当前机架内的节点)

ELSE

　开始与其他没有内架转移的机架进行跨机架转移

END IF

IF 与任何其他没有跨机架传输的机架进行跨机架传输是可能的 **THEN**

　开始与选定的机架进行跨机架转移

ELSE

　等待先完成跨机架转移的机架，再开始转移

END IF

5　实验对比和总结

cheng 等人将 HybridPL 实现为一个名为 LogECMem 的键值存储，并将其与就地更新 IPMem 和全条带更新 FSMem 进行对比。结果如图9所示，可以发现 LogECMem 对于不同的码型和读/更新比，其内存开销和更新延迟性能表现的稳定且优秀，而另外两种方案的性能波动大，无法兼顾内存开销和更新延迟两方面的性能。

Liu 等人将自己提出的 RPR 流水修复方法与传统修复方法和 CAR 方法在亚马逊 EC2 平台上进行了对比测试，结果如图10所示。结果表明，与传统修复相比，RPR 平均可减少 67.6% 的总修复时间，最高可减少 80.8%。与 CAR 相比，RPR 可以平均减少 37.2% 的总修复时间，最多可减少 50.3%。

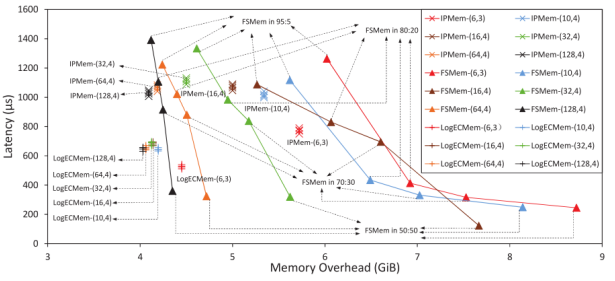


图 9　权衡分析不同 (k,r) 码的内存开销和更新延迟，以及 4KB 值大小的读/更新比

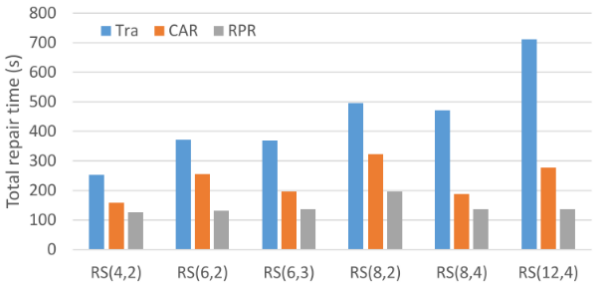


图 10　传统修复 (Tra)、CAR 和 RPR 修复在 AWS EC2 机器上使用不同 RS 代码的单块故障修复时间

参考文献

[1] CHENG L, HU Y, KE Z, et al. Logecmem: coupling erasure-coded in-memory key-value stores with parity logging[C]//Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. [S.l.: s.n.], 2021: 1-15.

[2] HU Y, CHENG L, YAO Q, et al. Exploiting combined locality for {WideStripe} erasure coding in distributed storage[C]//19th USENIX Conference on File and Storage Technologies (FAST 21). [S.l.: s.n.], 2021: 233-248.

[3] LIU T, ALIBHAI S, HE X. A rack-aware pipeline repair scheme for erasure-coded distributed storage systems[C]//ICPP '20: 49th International Conference on Parallel Processing. [S.l.: s.n.], 2020.

[4] YANG J, YUE Y, RASHMI K. A large scale analysis of hundreds of in-memory cache clusters at twitter[C]//14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20). [S.l.: s.n.], 2020: 191-208.

[5] SHI H, LU X. Inec: Fast and coherent in-network erasure coding[C]//SC20: International Conference for High Performance Computing, Networking, Storage and Analysis. [S.l.]: IEEE, 2020: 1-17.