分布式系统纠删码研究进展

汪睿阳

华中科技大学计算机学院技术学院

摘 要 当今现实世界的数据爆炸增长，如何有效保护和存储数据已经成为一个热门话题，由于纠删码在相同的保障可靠性下能大幅减少内存的开销，因此纠删码存储已经成为了取代副本存储的有效方案。然而，尽管如此，纠删码还是会产生一定的数据冗余，且为了跟上数据的增长速度，纠删码的性能和延迟也必须要进一步提高。为了解决这些问题，学术界针对纠删码的两个主要方面进行研究：1) 减少纠删码带来的数据冗余；2) 提高纠删码的性能（降低纠删码的延迟）。本文将分别对纠删码的相关论文进行研究：1) 使用程序优化技术加速基于 XOR 的纠删码；2) 组合局部宽带条纠删码在分布式存储中的应用；3) 利用 CRaft 降低存储成本和网络成本；4) 使用 LogECMem 耦合纠删码内存键值存储和奇偶校验日志；5) 利用 INEC 构造快速和连贯的网络内纠删码。通过分析上面几篇论文，总结纠删码的发展现状。

关 键 词 纠删码；数据存储；分布式系统

Survey on Erasure Codes for Distributed Systems

Abstract With the explosive growth of data in the real world, how to effectively protect and store data has become a hot topic. Since erasure code can greatly reduce the overhead of memory under the same guarantee reliability, erasure code storage has become an effective solution to replace replica storage. However, in spite of this, erasure codes still produce a certain amount of data redundancy, and in order to keep up with the growth rate of data, the performance and latency of erasure codes must be further improved. In order to solve these problems, the academic community has focused on two main aspects of erasure codes: 1) reducing the data redundancy caused by erasure codes; 2) improve the performance of erasure codes (reduce the latency of erasure codes). In this paper, we will study the related papers of erasure codes: 1) using program optimization techniques to accelerate XOR-based erasure codes; 2) Application of combined local wideband erasure codes in distributed storage; 3) Using CRaft to reduce storage cost and network cost; 4) Using LogECMem coupled erasure code memory key-value store and parity check log; 5) INEC is used to construct fast and coherent in-network erasure codes. By analyzing the above papers, this paper summarizes the development status of erasure codes.

Key words Erasure code; Data storage; Distributed system

1 引言

纠删码（Erasure Coding，EC）是一种编码容错技术，最早是在通信行业解决部分数据在传输中的损耗问题。其基本原理就是把传输的信号分段，加入一定的校验再让各段间发生相互关联，即使在传输过程中丢失部分信号，接收端仍然能通过算法将完整的信息计算出来。在数据存储中，纠删码将数据分割成片段，把冗余数据块扩展和编码，并将其存储在不同的位置，例如磁盘、存储节点或者其他地理位置。

当今大数时代背景下，海量大数据的存储备份时刻冲击着当前先进的数据存储与纠删技术。

分布式数据存储系统作为经典数据容错技术，在进行数据保障的过程中采用容错技术、多副本存储备份技术以及误码数据纠删等方式来保证数据存储的可靠性。纠删码技术以其数据存储过程中资源消耗低、可靠性高等优点在数据纠删存储领域得到了广泛应用，但是传统纠删技术依然存在数据修复速度低、修复率低等缺点，且集群的扩展导致节点性能降级现象频发, 出现严重的长尾延迟问题, 尾延迟虽然占比少, 但在分布式存储系统中却成为了系统的性能瓶颈。

最常用的纠删码是 Reed-Solomon (RS) code，从宏观层面来看，RS code 包含两个参数 n，k。k 表示一个数据条带未编码的原始数据块，n 表示一个数据条带内所有的数据块，n-k 则对应于从原始数据块中编码的校验块数量。RS code 具有最小的冗余度为 (n/k) x，而且可以容忍任何 n-k 个节点的丢失。而副本技术在同样的容错情况下的冗余度为 (n-k+1) x。Facebook f4 使用了 n=14，k=10 的 RS code 可容错任意四个节点。此时的冗余度仅为 1.4x，但是如果使用副本技术的话，冗余度就为 5x。不过过去的经验通常是使用较为居中的数字。一般系统都是会容忍三个或者四个节点的失效，而且数据条带的大小通常都是 20 以内。由于在修复任何一个块都需要检索在同一条带内其他的多个块来恢复这个损坏的块（比如，在 (n, k) 的 RS code 中，需要检索 k 个块），来恢复一个，所以，一般在选择条带的尺寸时都会选择中等大小的尺寸。如果条带尺寸过大的话，在校验块数量相同时就会导致 k 较大，这就意味着在修复过程中需要更大的带宽和 I/O 请求。

确保数据冗余是分布式存储等大规模系统的最关键任务。复制——分配数据的副本是最简单的解决方案。擦除编码（EC）由于其空间效率，已经引起了人们的极大关注。例如，著名的分布式系统 HDFS（Hadoop 分布式文件系统）提供了编解码器 RS（10，4），Reed-Solomon EC，有 10 个数据块和 4 个奇偶数块。在 RS (10, 4) 上，可以存储比通过复制多 10 倍的对象；但是，如果有 5 个节点发生故障，就无法恢复数据。另一个分布式系统 Ceph 为任何n和p提供 RS (n, p)。在 Linux 上，可以使用 RAID-6，一个类似于 RS (n, 2) 的编解码器。使用 EC 而不是复制会降低系统性能，因为 EC 的编码和解码是繁重的计算，每次存储到系统中和从系统中加载时都需要计算。

当冗余度减小时可以大大缩减存储成本，在实际应用中，冗余度减小 14%（从 1.5x 减小到 1.33x）可以节约上千万元的成本，因此，使用宽条带的纠删码是非常有必要的，也就是说，要使得 n 和 k 都尽可能的大，同时 n-k 任然保持在 3 或者 4。宽条带的存储一直是研究热点，而且有机会实现最优的冗余度。比如在 VAST 中考虑了 (n, k)=(154, 150) 的配置，这种情况下，冗余度仅仅只有 1.027。因此，不论对于热数据或者冷数据的分布式系统，宽条带擦除码都具有良好的发展前景。擦除码过去常常使用在冷数据的存储系统中，这些数据很少被获取。宽条带擦除码允许数据实现在较低的成本下实现持久的存储。擦除码也同样被热数据存储系统所采用，这些数据通常时会被频繁访问的键值存储的对象。对于热数据使用可以减少硬件的开销。

虽然宽条带擦除码可以实现极大地节约成本，但是他们进一步加剧了修复时地惩罚。因为修复带宽，即修复期间数据传输的带宽会随着 k 的增加而增加。许多现有的高效修复的擦除码存储方法是利用一些局部性来减小修复带宽。最常用的局部性有两种：校验块局部性，通过引入额外的本地校验块，在修复某些丢失块的同时来减小可用块的数量；拓扑局部性，考虑到系统拓扑的层次性，并执行局部修复操纵，以缓解跨机架或跨集群修复带宽。

2 原理与优势

2.1 纠删码的性能加速

RS 有两种主要的加速方法：(1) 紧密耦合 MM 和有限场乘法的复杂优化方法。英特尔提供了一个 EC 库，ISA-L（智能存储加速库），基于这种方法。ISA-L 针对 F28 的 MM 进行了特别的优化，并为每个平台提供不同的汇编代码，以最大限度地提高 SIMD 指令的性能。英特尔在中报告了 ISA-L 对 RS(10,4)的编码吞吐量约为 6.0GB/s。在评估中，ISA-L 的得分约为 6.7 GB/s。(2) 基于 XOR 的 EC 将 F28 上的 MM 转换为 F2 上的 MM，其中 F2 是比特{0, 1}的有限域。

研究人员使用语法压缩算法 RePair 和基于 Xor 的语法压缩算法 RePair，通过不断减少 XOR 的使用数量，从而达到程序在多项式时间之内实现的效果。为了减少内存访问，作者使用

Deforestation 减少 SLP 直线程序的内存访问量。最后，再通过使用 (red-blue) pebble game 缓存优化减少高速缓存的缺失。优化后的 EC 库在编码方面超过了 ISA-L，在解码方面超过了 parallels。

优化前的基于 XOR 的 EC 方法（编码吞吐量为 4.9 GB/s）与基于另一种方法（编码吞吐量为 6.7 GB/s）的 Intel 高性能 EC 库之间的一个显著性能差距，作者优化后的实验库吞吐量为 8.92 GB/s，优于 Intel 的库。如图 1 所示。

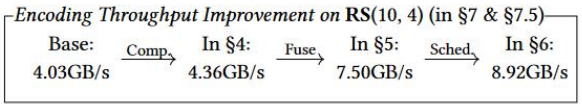

图 1 基于 XOR 的 EC 方法优化前后吞吐量

2.2 纠删码的存储节省

降低冗余度和提高修复时间

近期提出的宽条带可以抑制条带中校验块的比例，以达到极大的存储节省。然而，宽条纹加剧了修复代价，而现有的纠删编码高效修复方法不能有效地处理宽条纹。对于宽条带擦除编码，我们考虑表 1 中所示的典型容错级别的 k（例如，k=128）。组合局部性机制通过结合奇偶局部性和拓扑局部性来系统地解决宽条纹修复问题。进一步用高效的编码和更新方案增强组合局部性。在 Amazon EC2 数据集上的实验结果表明，与基于局部性的方法相比，该方法在冗余度低至 1.063 倍的情况下，单块修复时间最多可降低 90.5%。

Storage systems	(n,k)	Redundancy
Google Colossus [25]	(9,6)	1.50
Quantcast File System [49]	(9,6)	1.50
Hadoop Distributed File System [3]	(9,6)	1.50
Baidu Atlas [36]	(12,8)	1.50
Facebook f4 [47]	(14,10)	1.40
Yahoo Cloud Object Store [48]	(11,8)	1.38
Windows Azure Storage [34]	(16,12)	1.33
Tencent Ultra-Cold Storage [8]	(12,10)	1.20
Pelican [12]	(18,15)	1.20
Backblaze Vaults [13]	(20,17)	1.18

表 1 最先进的纠删码部署中的(n, k)常用参数

研究人员设计的 ECWide，是一种实现局域联合的宽条擦除编码存储系统。ECWide 解决了在宽条擦除编码中实现高效修复、编码和更新的挑战，目标如下：最小跨机架维修带宽，ECWide 通过组合地点使跨机架维修带宽最小化；高效编码，ECWide 采用多节点编码，支持对宽条带进行高效编码；高效的奇偶校验更新，ECWide 应用内部机架奇偶校验更新，允许全局和本地奇偶校验块主要在本地机架中更新。

纠删码降低存储成本和网络成本

共识协议能够提供高可靠、高可用的分布式服务。在这些协议中，日志条目被完全复制到所有服务器。这种全表项复制会导致较高的存储和网络成本，影响性能。纠删码是一种常用的技术，可以在保持容错能力不变的情况下降低存储和网络成本。如果将一致性协议中的完全复制替换为纠删码复制，可以大大降低存储和网络成本。

存活性能是共识协议最重要的性质之一。但是现有的支持纠删码的协议 RS-Paxos 无法达到活跃度。该协议基于 Raft，继承了 Raft 的基本概念，其目标是设计一个新的支持纠删码的协议（这样它可以节省存储和网络成本），具有 F 活性级别。

为了降低网络成本，新协议中的领导者应该能够通过使用编码片段将其日志条目复制给追随者，就像 RS-Paxos 一样。然而，仅采用编码片段复制方式的协议无法达到 F 活性水平，Raft 中的完全复制方法是必需的。在新协议中，需要同时进行编码片段复制和完整项复制。使用 codedfragment 复制可以节省存储和网络开销，而完全复制可以保持活跃。

RS-Paxos 是第一个支持纠删码数据的一致性协议，但与常用的一致性协议 (如 Paxos 和 Raft) 相比，其可用性较差。在 Raft 协议的基础上，提出了一种新的协议 CRaft。CRaft 提供两种不同的复制方法，可以像 RS-Paxos 一样使用纠删码来节省存储和网络成本，同时保持与 Raft 相同的活性。作者基于 CRaft 构建了一个 key-value 存储，并对其进行了评估。实验结果表明，与原始 Raft 相比，CRaft 可节省 66%的存储空间，提高 250%的写吞吐量，降低 60.8%的写延迟。

2.3 平衡性能和内存开销

研究人员提出一种新的纠删码内存 KV 存储架构 HybridPL，采用就地更新和奇偶校验日志的混合。首先，HybridPL 允许所有数据和一些校验块在 dram 中执行就地更新，以快速进行单次故障修复，同时保持低内存开销。其次，HybridPL 允许其他校验数据块进行校验记录，以减少更新过程

中的数据块传输量; 此外，HybridPL 还提出了一种缓冲记录机制，以加速日志节点中校验数据块的更新。将上述架构实现为内存 KV 存储 LogECMem，并在 LogECMem 之上设计了高效的多故障修复方案。

作者构建了 LogECMem 原型，并通过 Amazon EC2 实验将其与就地更新和全条带更新进行了比较。实验表明，LogECMem 在保持较高的基本 I/O 和修复性能的同时，更新时间分别减少了 37.8%和 58.0%，内存开销分别减少了 22.2%和 49.0%。

2.4 纠删码的降低延迟

作者提出了一套连贯的网络内原语——INEC，利用一个性能模型（即 α - β 模型）来分析 INEC 原语在多个最先进的 EC 方案中的性能提升。INEC 使最先进的 EC 方案能够充分利用现代 SmartNIC 上的 EC 卸载能力，INEC 的性能增益在五个最先进的 EC 方案中得到了验证。最终发现使用 INEC 的 TriEC 在编码带宽方面明显优于其他 EC 方案，使用 INEC 的 LRC 获得了最佳的解码带宽性能。研究人员在商用 RDMA 网卡上实现了 INEC，并将其集成到 5 个最新的 EC 方案中。实验表明，INEC 原语显著降低了第 50、95 和 99 个百分点的延迟，并将与 INEC 协同设计的键值存储的端到端吞吐量、写和读性能分别提高了 99.57%、47.30%和 49.55%。

3 研究进展

3.1 使用程序优化技术加速基于 XOR 的纠删码

使用程序优化技术加速基于 XOR 的纠删码是利用软件上的算法对原有的基于 XOR 的纠删码进行优化[1]。

（1）压缩。作者使用压缩算法 RePair，如图 2 所示，该算法在语法压缩中被用来压缩上下文自由文法（CFG）。作者可以通过忽略 SLPs 的⊕，以及将 SLPs 的常量（即变量）识别为 CFG 的终端（即非终端）来立即调整它。RePair 通过提取一个程序（或 CFG）的隐藏重复结构来压缩它。对于P，RePair 提取了重复出现的子项$c \oplus d \oplus e$，并用一个新的变量λ代替它。它将七个 XOR 减少到五个，因此速度提高了 30%。作者将 RePair 扩展到 XorRePair，如图 3 所示，增加了 XOR 取消属性（$x \oplus \oplus y = y$），这在语法压缩中是没有考虑的。

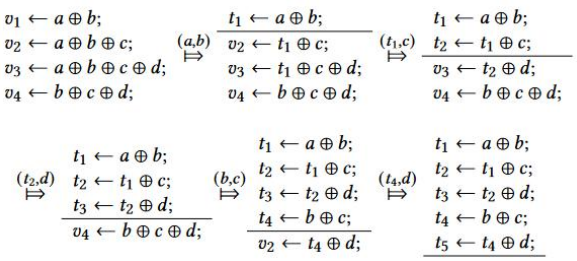

图 2 RePair 算法

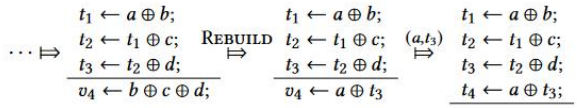

图 3 XorRePair 算法

（2）融合。为了减少内存访问，作者采用了函数式程序优化中的一种叫做 "Deforestation"的技术，如图 4 所示。Deforestation 通过融合函数消除了中间数据。虽然它有很深的背景理论，但由于 SLP 的简单性，即一个二进制算子，没有分支，也没有函数。在作者的例子中，$c \oplus d \oplus e$调用了六个内存访问，因为$c \oplus d$调用了三个，用于加载c和d，并将结果存储到一个中间数组$Ic \oplus d$，然后$Ic \oplus d \oplus e$调用三个。将$c \oplus d \oplus e$融合到É(c, d, e) 中，作者消除中间数组$Ic \oplus d$。融合后的 XOR 只调用四个内存访问：加载c，d，和e，并存储结果数组。

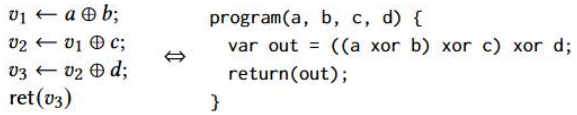

图 4 Deforestation 技术

（3）调度。为了减少缓存的缺失，作者重新审视了众所周知的（但模糊的）缓存优化的格言，增加数据访问的位置性。缓存优化的格言太模糊了，无法自动优化 SLP 并将其纳入作者的优化器。因此，作者引入了缓存效率的措施，并将作者的优化问题具体化为减少给定 SLP 的措施。为了正式确定作者的 SLPs 的缓存优化问题，作者采用了程序分析的（红蓝）卵石博弈，如图 5 所示。这个博弈是一个简单的抽象的计算模型，有快有慢的设备。在作者的设定中，快慢设备分别是高速缓存和主内存。

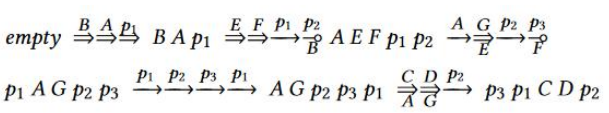

图 5 (red-blue) pebble game

3.2 分布式存储中组合局部宽条带纠删码的开发

ECWide 为两种类型的修复操作实现了联合局域性: 单块修复和全节点修复[2]。

单节点修复

ECWide 实现了两步联合局部修复。假设一个存储系统将数据组织在给定 k、f 和 γ 的固定大小的块中。ECWide 确定参数 n 和 r。然后将数据块编码为 n 个局部/全局奇校验块。ECWide 为每个本地组选择 (r+1)/f 个机架，并将每个本地组的所有 r+1 个块均匀地放置到这些机架上的 r+1 个不同的节点中 (即每个机架 f 个块)。由于上述两个步骤确保单个块修复的跨机架修复带宽最小化 (r+1)/f-1 块， ECWide 只需要为修复操作提供以下细节。图 6 描述了机架 R1 中丢失的块 D1 的修复。具体来说，ECWide 选择 R1 中的一个节点 N1 (称为请求者) 来负责重建丢失的块。它还在机架 R2 中选择一个节点 N4 (称为本地修复器) 来执行本地修复。然后 N4 收集 R2 内的所有区块 D5 和 P1 [1-5]，计算一个编码的区块 P1 [1-5]-D4-D5 (假设 P1 [1-5]是 D1,..., D5 的异或和)，并将编码后的块发送给请求者 N1。最后，N1 在 R1 内收集数据块 D2 和 D3，并通过从接收到的编码块 P1 [1-5]-D4-D5 中减去 D2 和 D3 来求解 D1。

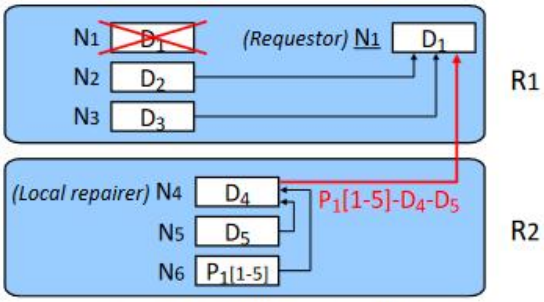

Figure 5: Repair in ECWide.

图 6 单节点修复

全节点修复

一个全节点修复可以被视为多个条带的多个单块修复 (例如，每个条带一个丢失的块)，可以并行化。然而，每个单块修复涉及一个请求者和多个本地修复者，因此多个单块修复可能会选择相同的节点作为请求者或本地修复者，从而使所选节点超载，降低全节点修复的整体性能。因此，作者的目标是选择尽可能多的不同节点作为请求者和本地修复者，以有效地并行化多个单块修复。为此，ECWide 设计了一种最近最少选择的方法来选择节点作为请求者或本地修复者，并通过一个双链接列表和一个 hashmap 实现它。双链接列表保存所有节点 ID，以跟踪最近选择了哪个节点或其他节点，而 hashmap 保存列表的节点 ID 和节点地址。然后，通过简单地选择列表的底部之一，并在 O (1) 时间内通过 hashmap 更新列表，作者可以获得最近最少选择的节点作为请求者或本地修复者。

ECWide-C。ECWide-C 主要用 Java 实现，大约有 1500 个 SLoC，而编码模块是用 c++实现的，大约有 300 个 SLoC，基于 Intel ISA-L[1]。它有一个 MasterNode，用于存储元数据并使用 Scheduler 守护进程组织修复和编码操作，还有多个 datanode，用图 6 单节点修复图 7 不同情况下的修复时间于存储数据并执行修复和多节点编码操作。

ECWide-H。ECWide-H 构建在 Memcached 内存 key-value store (v1.4) 和 libMemcached (v1.0.18) 热存储之上。它是用 C 语言实现的，有大约 3, 000 个 SLoC。它遵循客户机-服务器体系结构。它包含存储键值项的 MemcachedServers，以及执行修复和奇校验更新操作的 memcachedclient。它还包括用于管理元数据的协调器。协调器包括 Scheduler 守护进程，它协调修复和奇偶校验更新操作，以及 Updater 守护进程，它分析更新频率状态。ECWide-H 不包括 ECWide-C 编码模块, 自 erasure-coded 内存块大小键值存储通常是小和一个单节点的 CPU 缓存足够大, 预取的所有数据块的宽条纹高编码性能。

图 7 (a)-8 (e) 显示了在网关带宽为 1gb/s 和 500mb /s 时，对于不同的 k 值和 f 值，LRC、TL 和 CL 的平均单块修复时间。在相同的 k、f 和网关带宽下，CL 总是优于 LRC 和 TL，而具有最小冗余的 TL 通常表现最差。如图 7(c) 所示，当网关带宽为 1gb /s 时，r = 7 的 CL 的单块修复时间为 0.8 s，而 r = 7 的 LRC 和 TL 的单块修复时间分别为 3.9 s 和 9.0 s; 同样，CL 可使 LRC 和 TL 的单块修复时间分别减少 79.5%和 91.1%。在较小的网关

带宽下，CL 比 LRC 具有更高的增益。例如，在图 7 (c) 中，当网关带宽为 500mb /s 时，CL 对 LRC 的增益为 82.1%，高于网关带宽为 1gb /s 时的 79.5%。原因是 CL 最小化了跨机架的修复带宽，因此当网关带宽受到更大的限制时，其性能增益更明显。

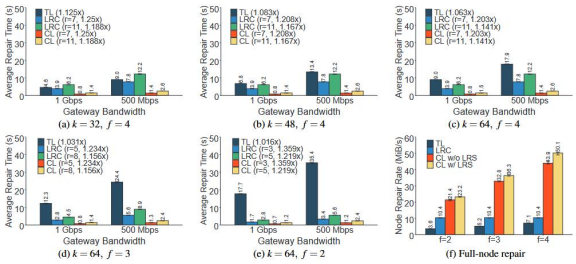

图 7 不同情况下的修复时间

最后，图 8 显示了平均单块修复时间和平均全节点修复速率随网关带宽的变化，范围从 1gb/s 到 10gb/s。这里，作者固定 k = 64 和 f = 4。从图 8(a) 中可以看出，在所有网关带宽设置下，CL 在单块修复方面仍然优于 LRC 和 TL，尽管随着网关带宽的增加，差异会变小。例如，当网关带宽为 10gb /s 时，r = 7 (0.34 s) 的 CL 的单块修复时间比 r = 7 (0.49s) 的 LRC 和 TL (1.11s) 的单块修复时间分别减少 30.6%和 69.4%。此外，从图 8(b) 中可以看出，在 LRC 和 TL 的全节点修复中，CL 保持了它的性能增益，而 LRS 带来了进一步的改进。

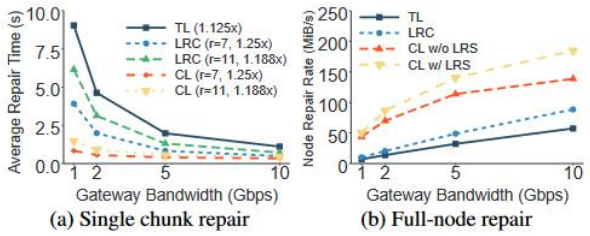

图 8 单块修复和全节点修复的平均时间

3.3 CRaft: 支持纠删码的 Ratft 版本，用降低存储成本和网络成本

纠删编码目前最关注的问题是其恢复成本相对于简单复制过高，有许多工作试图解决这个问题。大多数系统复制纠删码片段的方法类似于使用两阶段提交协议。这种方法在异步网络中失败的概率较高，而 CRaft 在这种情况下仍然可以很好地工作[3]。RS-Paxos 是第一个支持纠删码的共识协议。然而，它并不能达到最佳活性水平，并且错过了构建实用系统的重要细节。作者新的协议工艺解决了上述问题。Giza 使用元数据版本控制为纠删码对象提供一致性。然而，其方法主要考虑安全性，在传输用户数据时忽略了活性。

CRaft 在 Raft 的基础上，提供了两种不同的复制方法，以支持纠删编码并保持活性。基于最近心跳答案的预测可以帮助 leader 选择复制方法。此外，为了保证活性，LeaderPre 可以帮助新当选的领导人处理未应用的代码片段。作者假设协议中有 N = (2F + 1) 个服务器。因为 CRaft 应该具有与 Raft 相同的可用性，所以它的活性级别应该是 F，这意味着当至少 (F + 1) 个服务器是健康的时候，CRaft 仍然可以工作。作者选择一个 (k, m)-RS 编码工艺。k 和 m 满足 k + m = N，因此协议中的每台服务器对应于每条日志记录的一个编码片段。CRaft 支持纠删编码，因此它可以节省存储和网络成本，同时它具有 F 活性水平。

当一个行业领导者试图通过 codedfragment 复制方法复制一个条目时，它首先对条目进行编码。在 Raft 中，每个日志条目都应该包含来自客户机的原始内容，以及协议中的术语和索引。当工艺领导者试图编码一个条目时，内容可以被协议选择的 (k, m)-RS 编码成 N = (k + m) 个片段。Term 和 index 在协议中起着重要的作用，因此不应该进行编码。图 9 显示了编码过程。

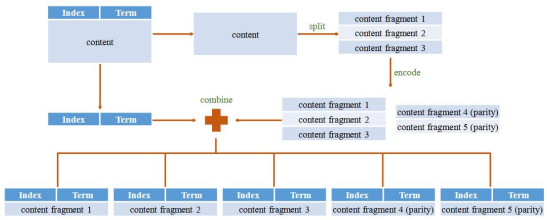

图 9 CRaft 中的编码过程

编码后，leader 将有 N 个词条的编码片段。然后，它将向每个追随者发送相应的编码片段。在收到对应的 codedfragment 后，每个 follower 会回复 leader。当领导者确认至少 (F + k) 个服务器存储了一个 codedfragment 时，条目和它之前的条目都可以安全地应用。leader 会提交并申请这些条目，然后通知 followers 去申请。编码片段复制的承诺条件比 Raft 更严格。此提交条件还意味着，当没有 (F + k) 个健康服务器时，leader 不能使用编码片段复制来复制条目，然后提交它。

当一个领导人下台时，新的领导人会被选举

出来。如果一个条目已经提交，Raft 的选举规则保证新领导人至少有一个条目的片段，这意味着安全属性可以得到保证。因为至少有 (F + k) 台服务器存储了提交数据项的片段，所以在任何 (F + 1) 台服务器中都应该至少有 k 个编码片段因此，当至少有 (F + 1) 个健康的服务器时，新的领导者可以收集 k 个编码片段，然后恢复完整的条目，这意味着活性可以得到保证。

为了降低存储和网络成本，鼓励 leader 使用编码片段复制。然而，当没有 (F + k) 个健康服务器时，编码片段复制将无法工作。当健康服务器的数量大于 F 且小于 (F + k) 时，leader 应该使用完全条目复制方法来复制条目。

在两种复制方法都能成功的情况下，使用编码片段复制比使用完全复制可以获得更好的性能。贪婪策略是 leader 总是试图通过编码片段复制来复制条目。如果它发现没有 (F + k) 个健康的服务器，它转而通过完全复制复制条目。然而，如果 leader 已经知道健康服务器的数量小于 (F + k)，那么通过 codedfragments 进行的第一次复制是没有意义的。

当 Leader 拥有所有完整条目时，两种复制方法都能保证安全性和活性。然而，当新当选领导人时，新当选领导人的日志很可能没有完整的副本，而只有一些条目的编码片段。当只有 (F + 1) 个健康服务器时，不能保证这些不完整的条目可以恢复。如果新当选的领导人没有申请这些不可恢复的项目，该领导人没有办法处理这些项目。leader 不能将包含这些条目中的任何一个的 AppendEntries rpc 发送给需要它们的 follower 4，因此这些不可恢复的条目将保留不应用。根据 Raft 的规则，leader 从客户端收到的新条目也不能复制到 followers。因此协议无法充分发挥作用，协议的活性也无法得到保证。因此，需要一些额外的操作来保证活性。

新当选领导人日志中的编码片段可以被领导人应用或取消应用。如果应用了一个编码片段，则条目必须由前一个 leader 提交。根据两种复制方式的提交条件，在任意 (F + 1) 服务器上至少存储 k 个编码片段或表项的一个完整副本。因此，当有 (F + 1) 台健康的服务器时，leader 总是可以恢复此条目。但是，如果未应用编码片段，则没有任何规则可以保证在存在 (F + 1) 个健康服务器时可以恢复此条目。

3.4 LogECMem: 耦合纠删码内存键值存储和奇偶校验日志

通过工作负载观察到，现有的纠删码内存 KV 存储的更新方案要么招致许多块传输，要么需要较高的内存开销，这促使研究人员开发了一种基于奇偶日志的方案，以平衡更新和内存的成本。

通过可靠性分析表明，改善单故障修复可以显著提高可靠性，因此本文提出 HybridPL，以低内存占用为快速单故障修复执行就地更新，同时利用奇偶日志和缓冲区日志进行快速更新[4]。

在 HybridPL 之上构建了一个内存中的 KV 存储 LogECMem，它实现了基本请求 (包括单次故障修复)、基于校验日志的更新和缓冲区日志。作者还通过批量合并奇偶校验更新来改进缓冲区日志记录。

基于 LogECMem 设计了高效的多故障修复方案，以减少磁盘 IOs，而不是最先进的基于奇偶日志的修复方案，同时仍然保持较高的修复性能。

为了解决如何将内存中的 KV 存储与奇偶校验日志耦合起来以获得高修复和更新性能的挑战。本文首先分析了单故障修复率对可靠性的影响，并表明提高单故障修复率可以显著提高可靠性。基于可靠性结果，作者设计了架构 HybridPL，通过在 DRAM 节点中进行就地更新来保持条带的数据和异或奇偶校验块，以获得低内存开销和高单次故障修复性能，同时将其他奇偶校验块的奇偶校验增量记录到磁盘节点，以及利用缓冲区日志技术来记录日志节点中的奇偶校验增量，以获得高更新性能。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。当请求的数据块因单点故障而无法直接读取时，HybridPL 通过检索同一分条中剩余的数据块和异或校验块，在 DRAM 节点内解码请求的数据块，从而保证单点故障修复的高性能。

因此，HybridPL 实现了低内存开销（HybridPL 通过对数据块的就地更新来减轻内存开销）和高效的单故障修复（HybridPL 在 DRAM 节点中保持 XOR 校验块，以确保降低读取效率）两大目标。

校验日志消除校验块传输: HybridPL 通过校验日志来更新除异或以外的校验块，这样 HybridPL 只需要将校验增量存储到日志设备中就可以清楚地更新校验块，HybridPL 既不读取旧的校验块来更新校验块，也不检索活动的未修改的数据块来

重新计算校验块（如全条带更新）。

但是，与 DRAM 节点相比，日志节点的低传输速率将成为降低写（更新）性能的瓶颈。为了解决这个问题，作者引入了 RAMCloud 中提出的缓冲区日志记录方法，该方法将副本分发给 dram 和磁盘。RAMCloud 将数据存储在主服务器的 dram 中，并将副本存储在备份服务器的磁盘上，如图 10(a) 所示。在写操作时，主服务器将数据写入 dram，并将数据副本转发给所有备份服务器。RAMCloud 认为，一旦副本被写入备份服务器的 dram，写操作就完成了，在备份服务器中，这些 dram 中的磁盘副本可以异步地刷新到磁盘。

虽然日志节点上的异步 IOs 可以加速写/更新操作，但需要维护崩溃一致性，以便在缓冲区崩溃时从磁盘日志中重建数据。

作者观察到缓冲区日志记录也可以应用于 HybridPL，如图 10(b) 所示。具体来说，在更新操作期间，代理还可以将相同的 delta 发送到所有日志节点，这与 RAMCloud 将相同的副本转发到备份服务器类似。然后，每个日志节点可以根据 § 2.1 中的属性 1，通过 delta 计算其奇偶性 delta (例如%2 和%3)。通过这种方式，HybridPL 可以执行快速写（更新）操作，因为它们可以在奇偶校验增量存储到日志节点的 dram 中之后立即完成，并且 dram 中的奇偶校验增量将批量异步地刷新到磁盘。

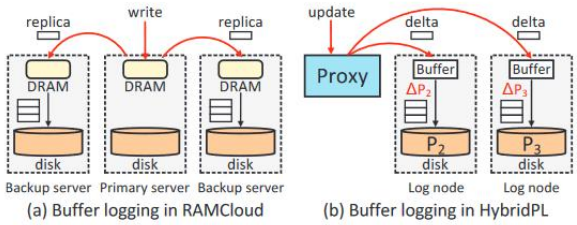

图 10 RAMCloud 和 HybridPL 中缓冲区日志示意图

因此，HybridPL 实现了高效更新（HybridPL 为非异或校验块执行校验日志记录，并利用日志节点中的校验增量缓存日志记录来加速更新）的目标。

作者基于 HybridPL 架构设计了 LogECMem，其减少了更新期间的块传输，在 LogECMem 中指定了如何使用合并奇偶增量来设计缓冲区日志记录，这可以进一步减轻磁盘 IOs 的开销。

LogECMem 支持 4 种基本请求: 写、读、降级读和删除，其中每个对象的键和值都是任意字符串。更新操作如图 11 所示。它可以通过简单地在 DRAM 节点中执行快速降级读取来修复单块故障，但在实际系统中有时会发生多块故障。存在两种多分块故障情况: (1) 同一分条的多个分块同时故障；(2) 单个节点故障包含多个故障分块，每个故障分块属于不同的分条。

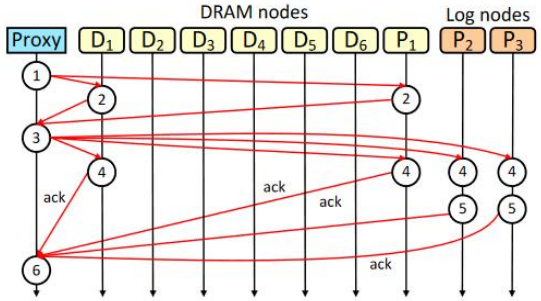

图 11 更新工作流

3.5 INEC: 快速和连贯的网络内纠删码

高速互连社区已经提出了几种在网卡上支持 EC 的商用智能。然而，当代智能设备的 api 分离地暴露了电子商务和网络功能，没有意识到电子商务计算与网络密切相关的事实。为了充分利用 SmartNICs 的性能并减少 CPU 参与以进一步缩短延迟，一种理想的设计方法是一致性 EC 计算和联网，或一致性网内 EC。相干网内 EC 提供了联合 api，如 recv-ec-send，用于一致性地执行 EC 计算和网络操作，而不是暴露单独的 api 来执行 EC 和网络操作。这些一致的 api 可以将任务卸载到 smartnic 上，smartnic 将负责等待接收完成、执行 EC 计算、等待 EC 完成和发送结果。如上所述，recv-ec-send 是 PPR 需要的原语，但需要的 CPU 数量要少得多[5]。如图 13 所示，使用一致的网内 EC，recvec-send 原语将任务列表卸载到 SmartNIC，每个任务的完成自动激活后续任务的执行，而无需 CPU 参与。尽管图 13 清楚地显示了一致性网内 EC 的好处，但要提供完整和高效的一致性网内 EC 功能，仍然有许多挑战有待解决。

如图 12 所示，不同的 EC 协议需要不同的通信图和完成 EC 任务的步骤。作者需要找到一种快速的方法将许多后续任务连接在一起，以构建分布式 EC 管道，并在不需要 CPU 参与的情况下在网卡上触发预发布任务。

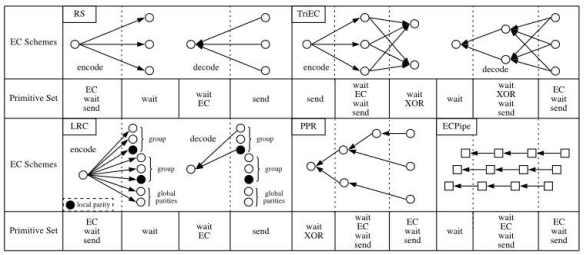

图 12 最先进的 EC 方案和层的基本集

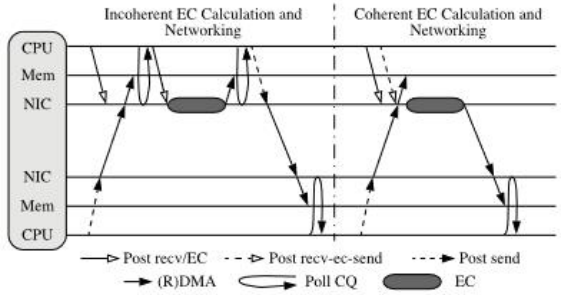

图 13 不连贯 vs.连贯

之前的一些研究揭示了为传统或特定的增强型 RS 码设计有效的相干网内 EC 的机会。然而，这些设计要么只能支持一种类型的代码，要么只是将一些步骤部分移交给网络，这使得它们不能完全解决作者在上面确定的挑战，并导致性能不佳。另一方面，最近的研究也指出，支持远程直接内存访问 (RDMA) 的现代网卡 (RNIC) 提供了 RDMA 等待等底层机制，可以在同一 RNIC 上的两个通信通道之间触发事件。为此，提出一种整体的方法来设计一组快速、一致的网内 EC 原语 INEC。

该方法有如下优势：1) 成功确定了仅需要 3 种基本的网内一致性 EC 原语 (即 EC /xor-send、recv-ec/xor-send 和 recv-ec/xor) 就能完全支持所有 5 种最先进的 EC 协议。2) 为了将这些 EC 原语和协议完全卸载到网络中，提出了高效的 RDMA 等待设计，以提供快速的 EC 和网络能力，并在 Mellanox OFED 驱动程序中实现了 INEC 原语。据作者所知，这是第一次提出一套完整的基于 RDMA 等待 rnic 的网内 EC 原语，以支持多种类型的 EC 协议。3) 提出了 α - β 性能模型，分析了 5 种最先进的 EC 方案的 INEC 原语性能增益。4) 通过与一个 key-value 存储系统的基准测试和协同设计，对所提出的 INEC 原语与 5 个主流 EC 协议进行了自下而上的验证。微基准测试表明，INEC 基元将现有 EC 方案的编解码带宽分别提升了 5.87 倍和 2.94 倍。在 YCSB 数据集上的实验结果表明，与 INEC 协同设计的 key-value store 在端到端吞吐量上最高提升了 99.57%，在写延迟和读延迟上分别最高降低了 47.30%和 49.55%。

4 论文总结

本文中对现在纠删码相关发展趋势进行总结，从多方面多角度分析流行的纠删码优化策略。针对纠删码的存储冗余的痛点，多队研究人员用不同的方法对纠删码进行改进，对存储结构进行优化，对延迟进行减少，对吞吐量等性能进行优化。

本文对使用程序优化技术加速基于 XOR 的纠删码，对性能进行提升；利用分布式存储中组合局部宽条带纠删码的开发和利用 CRaft 对存储进行节省；利用 LogECMem 进行耦合纠删码内存键值存储和奇偶校验日志平衡内存优化的性能优化；利用 INEC 构建快速和连贯的网络内纠删码，从而降低延迟。综上，学术界目前面对纠删码系统的修复问题已经有了比较成熟的方案，未来在纠删码提高性能和减少数据冗余仍然是热门趋势之一。

参 考 文 献

[1] UEZATO Y. Accelerating xor-based erasure coding using program optimization techniques[C]//SC21: International Conference for High Performance Computing, Networking, Storage and Analysis. .

[2] HU Y, CHENG L, YAO Q, et al. Exploiting combined locality for wide-stripe erasure coding in distributed storage[J].

[3] WANG Z, LI T, WANG H, et al. CRaft: an erasure-coding-supported version of raft for reducing storage cost and network cost[J].

[4] CHENG L, HU Y, KE Z, 等. LogECMem: coupling erasure-coded in-memory key-value stores with parity logging[C]//SC21: International Conference for High Performance Computing, Networking, Storage and Analysis. . DOI:10.1145/3458817.3480852.

[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. . DOI:10.1109/SC41405.2020.00070.