云存储系统纠删码研究进展

万宪欣1)

1)(华中科技大学计算机科学与技术学院 湖北 武汉 430074)

摘 要 在当前的云计算时代，存储在云中的数据正在以惊人的速度生成，云存储系统成为云计算的关键组成部分之一。通过将大量数据存储在托管云的数据中心内的商用磁盘中，云存储系统必须非常仔细地考虑一个问题:我们如何在存储开销和数据完整性方面既可靠又高效地存储数据?虽然很容易存储复制的数据以容忍一定数量的数据丢失，但它的存储效率非常低。传统的擦除编码技术，如Reed-Solomon编码，能够实现低得多的存储成本，同时对磁盘故障具有相同的容忍水平。但是，它会导致更高的修复成本，更不用说更高的访问延迟了。在这个意义上，为云存储系统设计新的编码技术已经在学术界和业界获得了极大的关注。在本文中，我们研究了云存储系统编码技术的现有成果。具体来说，我们将讨论三类利用局部性原理的编码技术分别是:本地局部性，拓扑局部性和结合这两种局部性。此外，我们还从高层次介绍了一些特定编码技术的主要思想，并讨论了它们的动机和性能。

**关键词** 纠删码；云存储；局部性修复

Survey on Erasure Code in Cloud Storage

Xianxin Wan1)

1)(School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, Hubei 430074)

**Abstract** In the current era of cloud computing, the data stored in the cloud is generating at an amazing speed. Cloud storage system has become one of the key components of cloud computing. By storing a large amount of data in commercial disks in the data center hosting the cloud, the cloud storage system must consider a problem very carefully: how can we store data reliably and efficiently in terms of storage overhead and data integrity? Although it is easy to store replicated data to tolerate a certain amount of data loss, its storage efficiency is very low. Traditional erasure coding technologies, such as Reed Solomon coding, can achieve much lower storage cost and have the same tolerance level for disk failure. However, it will lead to higher repair costs, not to mention higher access delays. In this sense, designing new coding technologies for cloud storage systems has attracted great attention in academia and industry. In this paper, we study the existing achievements of cloud storage system coding technology. Specifically, we will discuss three kinds of coding techniques using locality principle: local locality, topological locality and combining these two kinds of locality. In addition, we also introduce the main ideas of some specific coding technologies from a high level, and discuss their motivation and performance.

**Key words** erasure code; cloud storage; locality repair

# 引言

目前，据统计，以腾讯为例，qq每月活跃用户要超过8亿，同时在线用户要大于2亿，同时，微信月活跃用户要大于6亿。对应的qq空间相册和微信朋友圈的图片共计4000亿张，占用存储量200PB，日上传量10亿张，日下载量1200亿张。全球的数据还在以每年150PB加速增加。云存储系统(cloud storage system)面临巨大的存储压力。于此同时，大规模的云存储系统也不断面临着数据丢失的风险，这种风险的发生有可能是因为软件故障，也可能是硬件故障。为了提高数据的可靠性和可用性，最常用的做法便是采用数据冗余手段，这种手段可以通过副本(duplication)或擦除码(erasure code)来实现。副本是指在多个在多个不相同的物理位置重复存储数据的存储，但是这种机制会导致相当大的存储开销。与副本相比，擦除码可以显著减小存储开销。擦除码的低存储开销的特性对于许多云计算提供商非常有吸引力，擦除码技术正在逐渐被云存储领域的许多企业所采用。

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。表一展示了许多前沿的系统所使用的n，k。我们可以看见，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。

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。

# 原理和性能指标

## 原理

在一个存储系统中，数据存储在大量的商品磁盘上，很明显，磁盘故障不能仅仅视为例外，而是作为一个规则。因此，存储系统必须存储冗余，当一定数量的硬盘丢失数据时，其他硬盘仍然可以访问数据。例如，N-way复制，即在N个不同的磁盘上存储N个副本，最多能容忍N个磁盘故障。举个3-way复制的例子，其中原始数据被分散到3个不同的磁盘中，任何一个磁盘都能够修复或访问原始数据。而N路复制最多只能达到1/N的存储效率。另一方面，Erasure编码能够容忍相同数量的磁盘故障，但具有更好的存储效率。在erasure编码中，MDS (Maximum Distance可分离码)的存储效率最高。

假设在存储系统中，数据被组织在数据对象单元中，它可能对应于不同存储系统中的一个文件或一个固定大小的块。假设一个数据对象可以存储在n个硬盘上。给定任意数量的k，其中k <n;(n, k) MDS码可以保证最多容忍n-k个硬盘故障，即k个硬盘就足以访问原始数据的任意位。具体来说，数据对象被编码成n个编码块，并均匀地分发到n个磁盘中。假设数据对象的大小为M位，如果不考虑数据对象的元数据，则每个编码块的大小应为M/k位。从这个意义上说，MDS码的存储效率最多为k n。与3次复制相比(5,3) MDS代码最多可容忍2个硬盘故障，存储效率提高80%。

为了访问该数据对象，系统需要访问k个不同的编码块(来自k个不同的硬盘)，并通过相应MDS码的解码算法恢复原始数据对象。显然，解码算法会增加访问延迟。为了提高访问延迟，存储系统可以使用缓存来存储原始数据对象的一个副本。

在大多数情况下，恢复整个数据对象以进行数据访问可能听起来是合理的。但是，从存储系统的角度来看，如果我们只需要修复一个丢失的编码块，就完全没有必要恢复整个数据对象，因为我们需要的只是数据对象中非常小的一部分。不幸的是，在再生码出现之前，据我们所知，所有MDS码至少需要访问k个磁盘才能修复一个磁盘，而在复制的情况下，修复一个副本只需要传输一个副本。这种需求会极大地增加数据中心的磁盘I/O和带宽开销，并显著地影响存储系统和托管在云中的其他应用程序的性能。

## 性能标准

在云存储系统中，仅仅考虑存储效率和对磁盘故障的冗余度是不够的。在为云存储设计擦除编码时应该考虑的性能指标也应该包括：

修复带宽：为了修复故障的磁盘，存储在该磁盘上的数据应该在一个新的磁盘中修复。更换磁盘的服务器(称为新服务器)需要从一些现有磁盘中检索数据。如果托管这些现有磁盘的服务器(称为提供商)发送其编码块的原始数据，用于传输现有编码块的带宽等于这些编码块的大小，然后新服务器自己对接收到的数据进行编码，生成丢失的数据。然而，如果编码操作可以同时对新来者和提供者进行，而不是只对新来者进行，则传输的数据量会小得多。如果存储节点中存储的数据采用矢量编码进行编码，使每个编码块包含多个编码段，则维修期间提供商将其编码段的线性组合发送出去，可以节省带宽消耗。

修复I/O：除了带宽，修复擦除编码的另一个性能指标是参与服务器上的磁盘I/O。具体来说，写入操作只在新写入的对象上执行，写入的数据量应该等于编码块的大小。由于写入操作是不可避免的，所以我们真正关心的是从提供程序的磁盘中读取的数据量。与带宽消耗类似，传统的擦除代码会要求提供商总共读取k个块来修复一个块。对供应商编码操作不能读取的数据量减少,然而只有新来的减少传输的数据量,自段发出的供应商编码编码段的提供者,都必须从磁盘读取。因此，需要提出新的编码技术来改善修复期间的磁盘I/O。

访问等待延迟：由于解码操作，通过擦除编码编码的数据的访问延迟必须比副本大得多。然而，系统代码，原始数据可以嵌入到代码块中，能够保持比副本更高的存储效率，同时实现较低的访问延迟，因为我们现在只需要访问系统部分，而不需要解码操作。系统编码的这种吸引人的特性具有较高的价格，这使得修复比非系统编码复杂得多，因为修复后的数据应该与丢失的数据完全相同，至少对于嵌入的原始数据来说是如此。因此，在设计用于云存储的系统擦除代码时，必须考虑代码的确定性构造。

存储效率：存储效率是指存储在磁盘上的原始数据量与实际数据量的比值。换句话说，给定相同的数据量，在更高的存储效率下，我们可以使用更少的存储空间来容忍相同数量的磁盘故障。MDS码具有最优的存储效率，即给定n和k，可以构造MDS码，使n个编码块中的任意k都足以恢复原始数据。例如，为了容忍两个磁盘故障，3M位必须存储复制，而只需要5M/3位存储MDS代码。但是，如果我们选择保留此属性，则在修复期间不可避免地会招致较高的磁盘I/O开销。由于大容量磁盘的成本降低，对存储效率的要求可以降低，因此在修复过程中需要访问的磁盘数量要少得多。

# 研究进展

## Exploiting Combined Locality for Wide-Stripe Erasure Coding in Distributed Storage[9]

这篇论文设计了ECWide，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。

（1）修复过程

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。

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。在步骤1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。在步骤2中，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1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。

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，我们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，我们可以获得最近最少选择的节点作为请求者或本地修复者。

（2）实现

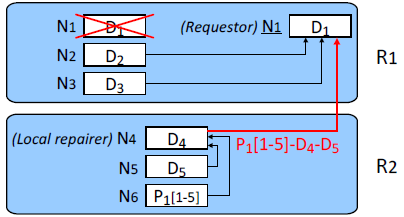
ECWide-C。ECWide-C主要用Java实现，大约有1500个SLoC，而编码模块是用c++实现的，大约有300个SLoC，基于Intel ISA-L[1]。它有一个MasterNode，用于存储元数据并使用Scheduler守护进程组织修复和编码操作，还有多个datanode，用

图1 单节点修复

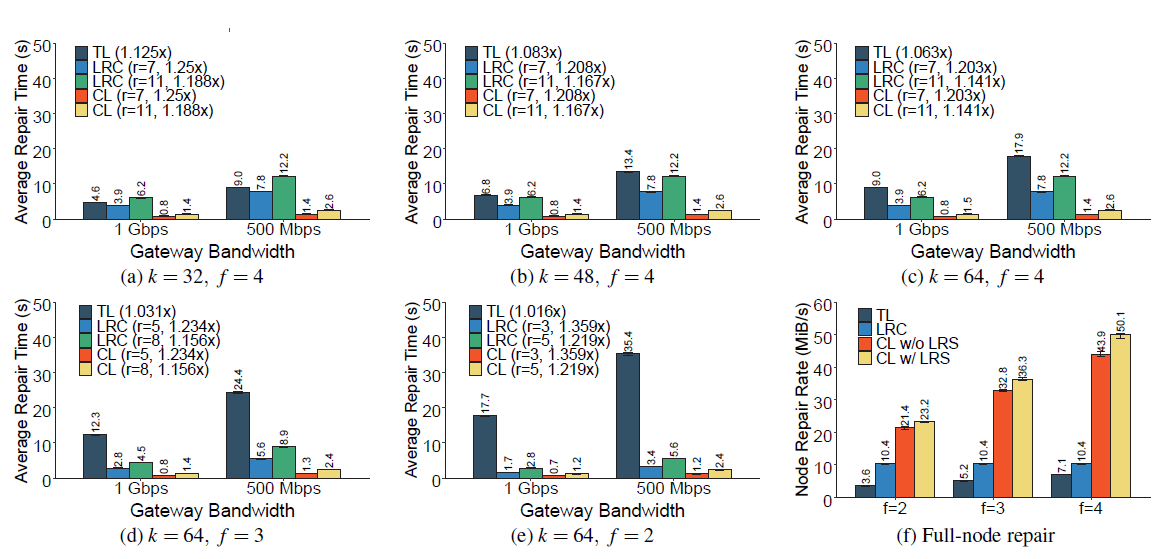


图2 不同情况下的修复时间

于存储数据并执行修复和多节点编码操作。请注意，ECWide-C不考虑更新模块，假设在冷库中更新很少。

ECWide-H。ECWide-H构建在Memcached内存key-value store (v1.4)[2]和libMemcached (v1.0.18)[3]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。注意,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。

（3）性能测试

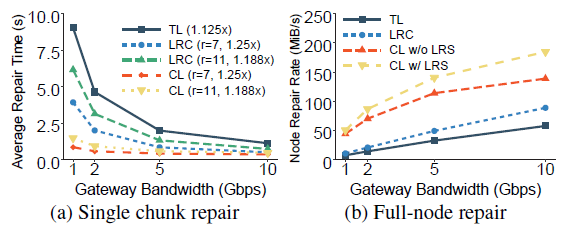
图2(a)-2(e)显示了在网关带宽为1gb/s和500mb /s时，对于不同的k值和f值，LRC、TL和CL的平均单块修复时间。在相同的k、f和网关带宽下，CL总是优于LRC和TL，而具有最小冗余的TL通常表现最差。如图2(c)所示，当网关带宽为1gb /s时，r = 7的CL的单块修复时间为0.8 s，而r = 7的LRC和TL的单块修复时间分别为3.9 s和9.0 s;同样，CL可使LRC和TL的单块修复时间分别减少79.5%和91.1%。在较小的网关带宽下，CL

图3 单块修复和全节点修复的平均时间

比LRC具有更高的增益。例如，在图2(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。

最后，图3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，我们固定k = 64和f = 4。从图3(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3(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。

## Erasure Coding in Windows Azure Storage[10]

（1）设计

文章从Reed-Solomon代码的例子开始解释重建成本的概念。(6,3) Reed- Solomon代码包含6个数据片段和3个奇偶校验片段，其中每个奇偶校验是从所有6个数据片段中计算出来的。当任何一个数据片段不可用时，无论使用哪个数据片段和奇偶校验片段进行重构，都需要6个片段。我们将重构成本定义为重构一个不可用数据片段所需的片段数。这里，重建成本等于6。

LRC的目标是降低重建成本。它通过从数据片段的子集计算一些奇偶来实现这一点。继续使用6个数据片段的示例，LRC生成4个(而不是3个)奇偶校验。前两个奇偶校验(标记为p0和p1)是全局奇偶校验，是从所有数据片段计算得到的。但是，对于其他两个奇偶校验，LRC将数据片段分成两个大小相等的组，并为每个组计算一个本地奇偶校验。为了方便起见，我们将这6个数据片段命名为(x0、x1和x2)和(y0, y1和y2)。然后，从一组(x0, x1和x2)中的3个数据片段计算本地奇偶校验px，从另一组

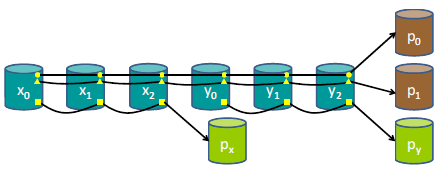


图4 A(6,2,2) LRC 例子

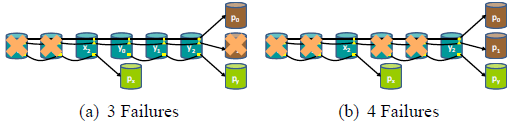


图5 有3个或4个节点失败

(y0,y1和y2)中的3个数据片段计算本地奇偶校验py。

现在，让我们来看看x0的重构过程。与其读取p0 (或p1)和其他5个数据片段(x1, x2, y0, y1, y2)，不如读取px和两个数据片段(x1, x2)来计算x0。很容易验证重构任何单个数据片段只需要3个片段，是Reed-Solomon代码所需数量的一半。

这个LRC例子比Reed-Solomon例子多增加了一个奇偶校验，所以看起来LRC以更高的存储开销为代价降低了重构成本。然而，在实践中，这两个例子在设计空间中实现了完全不同的权衡点。此外，在权衡存储开销和重构成本方面，LRC提供了比Reed-Solomon代码更多的选项。

我们现在正式定义了本地重建代码。A (k, l, r) LRC将k个数据片段分成l组，每组k/l个数据片段。它在每个组内计算一个本地奇偶校验。此外，它从所有数据片段计算全局奇偶。设n为片段的总数(数据+奇偶校验)。那么n = k + l + r，因此，归一化存储开销为n/k = 1 + (l + r)/k。我们示例中的LRC是(6,2,2)LRC，其存储成本为1 + 4/6 = 1.67x，如图4所示。

让我们首先解释最大可收回财产。在(6,2,2)LRC的例子中，它包含4个奇偶校验片段，并且最多可以容忍4个故障。然而，LRC不是最大距离可分离[4]，因此不能容忍任意4失败。例如，假设4个失败是x1, x2, x3和px。这种失败模式是不可解码的，因为只有两个奇偶项(全局奇偶项)可以帮助解码3个缺失的数据片段。另一个本地奇偶校验在本例中是无用的。它是不可能从仅仅两个奇偶校验片段解码3个数据片段，无论编码方程。这些类型的故障模式称为信息-理论上不可解码。

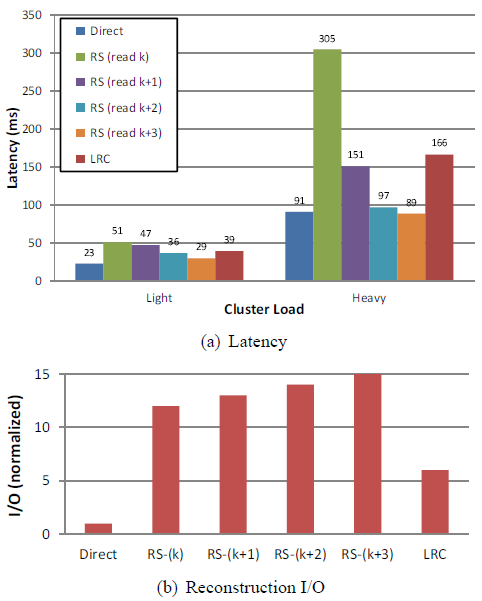
可以重构的故障模式称为信息-理论上可解码的。例如，图5(a)中的3-故障模式和图5(b)中的4-

图6 小I/O重建

故障模式都是信息理论上可解码的。对于这两种失效模式，可以构造编码方程，使其等价于使用图5(a)中的3个线性无关方程求解3个未知数，以及使用图5(b)中的4个线性无关方程求解4个未知数。

可以想象，构造一组可以解码特定失效模式的编码方程并不困难。然而，真正的挑战是构建一组编码方程，以达到最大可恢复(MR)属性，或者能够解码所有信息-理论上可解码的失效模式，这正是LRC的确切目标。

（2）测试结果

相比之下，LRC重建速度快。它只需要读取6个片段，达到了166ms的延迟，相当于Reed-Solomon读取13个片段时的151ms。请注意，使用Reed- Solomon进行极端激进的读取(读取全部15个数据)可以获得更低的延迟，但代价是需要更多的I/ O。I/O的相对数量被直接读的数量标准化，如图6(b)所示。快速重构和I/O成本节约是我们选择LRC而不是Reed-Solomon为windows Azure Storage进行擦除编码的原因。

现在我们研究4MB大I/O的重构性能。关键指标是延迟和带宽消耗。我们比较了直接读取和重建与Reed-Solomon和LRC。结果如图7所示。

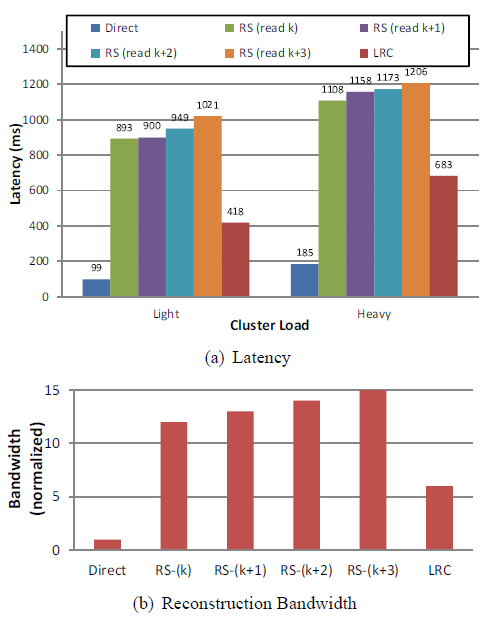
结果与小型I/O情况非常不同。即使在集群负

图7 大I/O重建

载较轻的情况下，考虑到其所消耗的带宽，使用erasure编码的重构已经比直接读取慢得多。与直接读取需要99ms相比，Reed-Solomon用12个片段需要893ms——慢9倍。

在大的I/O中，延迟大多受到网络和磁盘带宽的瓶颈，而这些结果的瓶颈是存储节点上1Gbps的网卡。由于LRC减少了一半的片段数量，其延迟为418ms，与Reed-Solomon相比显著降低。请注意，与小的I/O情况不同，Reed-Solomon使用更多片段的激进读取不会有帮助，反而会降低延迟。当集群负载较重时，观察结果也类似。

由于显著减少了延迟和带宽的节省，这在系统处于高负载或必须从机架故障中恢复时尤为重要，因此我们为Windows Azure存储选择了LRC。

LRU策略将清除其中一个现有条目。通过这个过程，提出的RL辅助GC调度器使用小的Q表缓存，即小的内存资源，进一步减少了长尾延迟。

## Optimal Repair Layering for Erasure-Coded Data Centers: From Theory to Practice[11]

（1）思路动机

实际的数据中心既容易受到独立节点故障的影响，也容易受到相关节点故障的影响[5]:独立节点故障是指每个节点由于单个事件(如磁盘/节点崩溃)而独立故障，而关联节点故障是指多个节点同时由于一个常见的灾难性事件(如停电或常见的交换机故障)而故障。在实践中，机架被视为可能发生相关节点故障的主要故障域。为了在数据中心部署擦除编码，现有的方法大多采用扁平的块放置方式，将条带的每个块放置在不同的机架上。这允许相同数量的节点故障和机架故障，并为独立和相关的节点故障提供最大的容错能力。

（2）实现

我们在Facebook的HDFS[6]上实现DoubleR，该[6]集成了HDFS- raid，在HDFS上支持擦除编码。我们将概述HDFS如何实现擦除编码，然后描述如何在HDFS上实现DoubleR。

我们将解释如何扩展Facebook的HDFS来包含DoubleR，并为我们的设计选择提供理由。

擦除码:我们根据3.3中给出的参数实现了不同的擦除码，包括RS码、MSR码和DRC码。对于MSR码，我们实现了n-k = 2时的Butterfly码[7]和n = 2k时的MISER码[8];这两种代码都是系统代码。对于DRC，我们为(n, k, r)的不同组合实现了两组实际的DRC结构。每个擦除代码在c++中使用Intel的ISA-L[1]实现。

我们主要使用两个ISA-L api: ec init表，它指定编码系数，以及ec encode数据，它指定编码/解码操作。这两个api都根据硬件配置自动优化计算(例如，如果支持，则使用Intel SSE指令)。我们通过Java本地接口(JNI)将每个擦除代码实现与Hadoop连接起来。

条带大小:在原来的HDFS-RAID中，一个块被划分为多个条带进行erasure编码，将位于相同块偏移量的条带编码在一起，形成一个较小的条带。我们的DoubleR实现利用了这一特性，并进一步使用多线程(参见下面的详细信息)来并行化现在跨越多个更小的条带的块的编码/解码。对于再生码和DRC，每个条带都划分为子条带，以便可用节点发送编码后的子条带进行修复;换句话说，再生码中一个已编码的子块和DRC是由一个块的多个已编码的子带组成的。请注意，如果条带太小，将会产生很大的I/O访问开销。我们通过实验研究了带材尺寸的影响。

块放置:DoubleR将属于同一机架中同一条带的多个块分组。我们修改RaidNode来指定如何根据参数(n, k, r)存储每个条带的块。

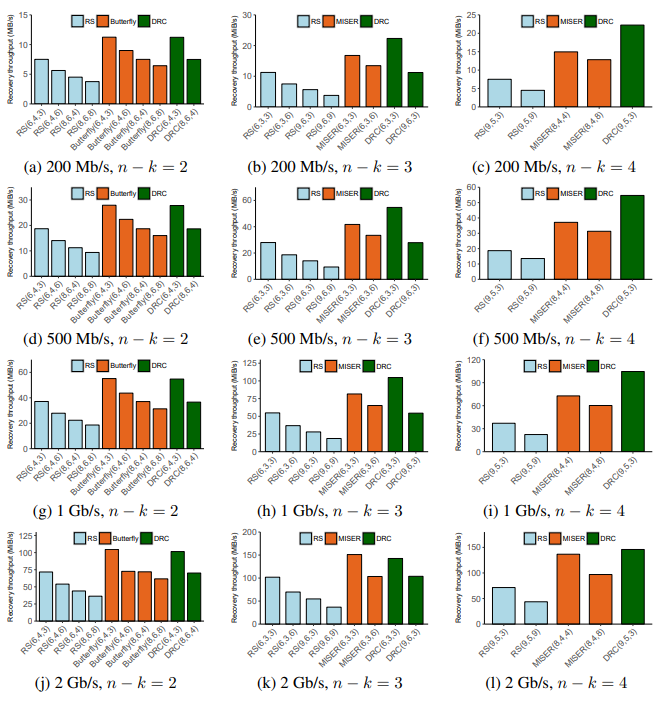
修复操作:DoubleR专注于修复单个故障。它目

图8 不同纠删码在不同情况下节点恢复的性能

前支持两种类型的修复操作:节点恢复和降级读取。节点恢复操作是指对单个故障节点的多个故障块进行修复，每个故障块属于不同的分条。我们修改RaidNode，使其在检测到故障节点时调用DoubleR进行节点恢复。另一方面，降级的读操作会修复单个不可用块。当文件系统客户端无法访问一个块并触发一个块丢失异常时，我们修改文件系统客户端来调用DoubleR来执行降级读取。

并行化:DoubleR不像原来的HDFS-RAID那样利用MapReduce进行修复;相反，它的实现体现了广泛的并行化，以加速修复操作，并将瓶颈转移到跨机架转移。首先，我们在节点级使用多线程来并行化磁盘I/O、编码/解码和网络传输操作。我们还生成多个线程来并行修复一个失败块的多个条带。此外，对于节点恢复(包括修复单个节点的多个故障块)，我们为不同的条带分配不同的中继和目标，以利用数据中心的并行性。

导出的api: DoubleR为修复操作导出三个原始api:(1) NodeEncode,存储节点从其本地存储的块中计算编码后的子块;(2)RelayerEncode,中继节点从同一机架中存储节点的编码后的子块中计算重编码后的子块;(3)Decode,目标节点重构失败的块 使用来自同一机架中的节点的块和来自不同机架中的中继的块 对于重新生成代码[9]及其变体[7，8],我们只需要实现NodeEncode和Decode;对于DRC,我们实现了所有这三个api 。

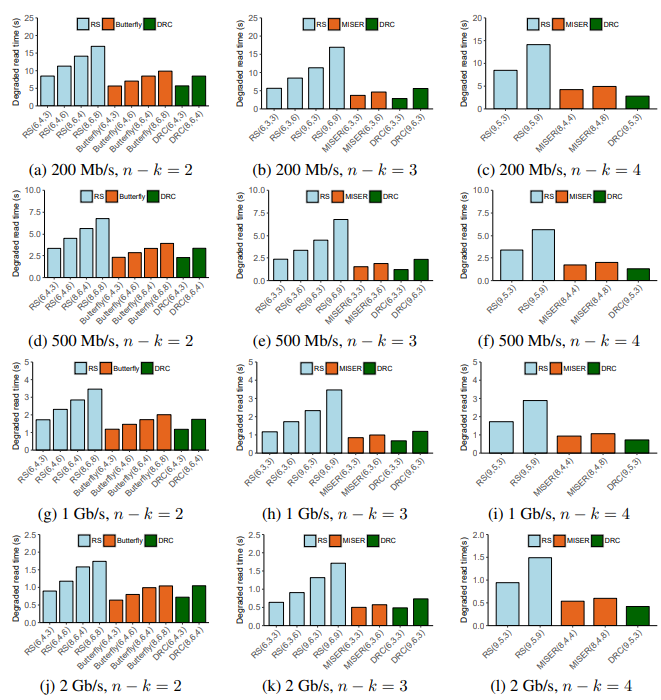


图9不同纠删码在不同情况下读的性能

（3）测试结果

图8显示了不同擦除代码的结果。当网关带宽在200mb /s ~ 1gb /s范围内时，测量的恢复吞吐量结果与图9中的数值结果基本一致，此时修复性能受到网关可用带宽的限制。例如，我们比较RS(9,5,3)和DRC(9,5,3)。从数值结果可以看出，RS(9、5、3)的跨机架修复流量为3个块用于修复单个失效块，而DRC(9、5、3)的跨机架修复流量仅为1个块。从测量结果来看，当网关带宽为200mb /s、500mb /s和1gb /s时，DRC(9，5，3)的恢复吞吐量分别为RS(9，5，3)的2.96、2.92和2.81(分别见图8(c)、图8(f)和图8(i))。总体而言，当网关可用带宽较小时，恢复吞吐量增益更接近理论增益。

图9显示了不同擦除代码的结果。我们看到，DRC通过最小化跨机架修复流量，在降级的读取中也显示了性能的提高。例如，我们比较RS(9,5,3)和DRC(9,5,3)。当网关带宽设置为200mb /s、500mb /s、1gb /s和2gb /s时，DRC(9，5，3)的读时延比RS(9，5，3)的读时延低66.9%、62.3%、58.0%和55.4%。

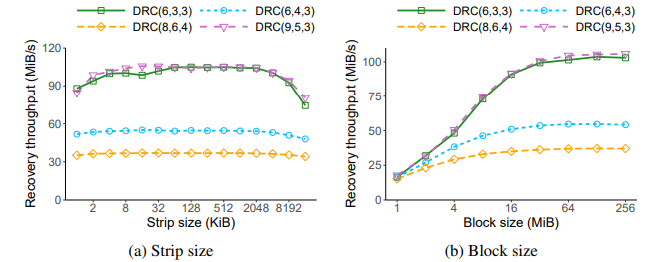
图10(a)首先显示了条带大小为1kib ~ 16mib时的恢复吞吐量，其中块大小固定为64mib。我们看到，当条带大小太小或太大时，会出现性能下降。具体来说，当strip大小小于8kib时，DoubleR需要发出更多的函数调用来访问一个块中的更多strip，开销会更大。当条带大小大于2mib时，同一块数据的多个条带间的并行度不能充分利用。当条带大 v

图10条带大小、块大小对节点恢复性能的影响

小介于两者之间时，恢复吞吐量最大。

图10(b)显示了块大小为1mib ~ 256mib (strip大小固定为256kib)时的恢复吞吐量。当块大小较小时，恢复吞吐量较小，因为块访问开销较大。恢复吞吐量随着块大小的增加而增加，当块大小大于等于64mib时达到最大值。

# 论文总结

在本文中，我们对云存储系统编码技术的发展进行了概述。云环境下的商用硬件的特性以及海量的存储设备给云存储系统的设计带来了挑战。通过引入擦除编码从再生码到本地修复代码,我们见证了这一趋势的研究云存储的擦除码,设计目标逐渐从数据完整性转移资源开销,和带宽资源其他稀缺资源的云存储系统,例如计算和磁盘I/O开销。

即使有了最近的进展，仍有一些有待研究的问题。在再生码的研究中，还有一些参数的选择，这些参数的存在和再生码的构造至今都是未知的。此外，为精确的流水线修复重新生成代码也是一个有待解决的问题。对于局部可修码，修复度与存储开销之间的权衡尚未明确。此外，还有一些其他的实际考虑因素可以与编码技术一起讨论，例如云中多个数据中心的地理性质。考虑到云存储系统可以在全球范围内扩展到多个数据中心，我们应该仔细讨论带宽、计算以及相应的地理异构性。

参 考 文 献

[1] Intel ISA-L. <https://github.com/intel/isa-l>.

[2] libMemcached. [https://libmemcached.org/lib Memcached.html](https://libmemcached.org/lib%20Memcached.html), Retrieved in Jan 2021.

[3]Memcached. https://memcached.org, Retrieved in Jan 2021. S. Balakrishnan, R. Black, A. Donnelly, P. England, A. Glass, D. Harper, S. Legtchenko, A. Ogus, E. Peterson, and A. Rowstron. Pelican: A building block for exascale cold data storage. In Proc. of USENIX OSDI,2014.

[4]F. J. MacWilliams and N. J. A. Sloane, “The Theory of Error Correcting Codes,” Amsterdam: North-Holland, 1977.

[5]Facebook’s Hadoop 20. <https://github.com/facebookarchive/hadoop-20>, Retrieved 2017

[6] L. Pamies-Juarez, F. Blagojevic, R. Mateescu, C. Gyuot, E. E. Gad, and Z. Bandic. Opening the ´ Chrysalis: On the Real Repair Performance of MSR Codes. In Proceedings of the 14th Usenix Conference on File and Storage Technologies (FAST’16), pages 81--94, Feb 2016.

[7]N. B. Shah, K. V. Rashmi, P. V. Kumar, and K. Ramchandran. Interference Alignment in Regenerating Codes for Distributed Storage: Necessity and Code Constructions. IEEE Transactions on Info. Theory, 58(4):2134--2158, Sep 2012.

[8]A.G. Dimakis, P. B. Godfrey, Y. Wu, M. Wainwright, and K. Ramchandran. Network Coding for Distributed Storage Systems. IEEE Transactions on Information Theory, 56(9):4539--4551, Sep 2010.

[9] Yuchong Hu, Liangfeng Cheng, and Qiaori Yao, Patrick P. C. Lee Weichun Wang and Wei Chen,.Exploiting Combined Locality for Wide-StripeErasure Coding in Distributed Storage. In Proc. of USENIX FAST, Feb 2021

[10] C. Huang, H. Simitci, Y. Xu, A. Ogus, B. Calder, P. Gopalan, J. Li, and S. Yekhanin. Erasure coding in Windows Azure Storage. In Proc. of USENIX ATC,Jun 2012.

[11] Y. Hu, X. Li, M. Zhang, P. P. C. Lee, X. Zhang, P. Zhou, and D. Feng. Optimal repair layering for erasure-coded data centers: From theory to practice. ACM Trans. On Storage, 13(4):33, 2017.