doi:10.3969/j.issn.1003－3114.2019.05.002
引用格式:李鑫,孙蓉,刘景伟.分布式存储系统中容错技术综述[J].无线电通信技术,2019,45(5):463－475.
[LI Xin,SUN Rong,LIU Jingwei.Overview of Fault－tolerant Techniques in Distributed Storage Systems [J].Radio Communications Technology,2019,45(5):463－475.]

分布式存储系统中容错技术综述

李　鑫[1],孙　蓉[1,2*],刘景伟[1]
(1.西安电子科技大学 综合业务网理论及关键技术国家重点实验室,陕西 西安 710071;
2.华侨大学 厦门市移动多媒体通信重点实验室,福建 厦门 361021)

摘　要:互联网、5G及其相关产业的飞速发展使我们迈入了大数据时代,存储海量数据将面临着巨大挑战。大规模分布式存储系统以其海量存储能力、高吞吐量、高可用性和低成本的突出优势取代了集中式存储系统成为主流系统。由于分布式存储系统中节点数量庞大,经常会产生各种类型故障,从而导致节点失效情况频发。因此,必须采用容错技术来保证在部分存储节点失效的情况下,数据仍然能够被正常读取和下载,具有容错能力且节约存储资源的分布式存储编码成为大数据时代重点研究的核心技术之一。讨论了大数据背景下存储与可靠性的问题,从而引出数据容错对分布式存储的重要性。阐述了传统的2种数据存储容错技术,即多副本机制和MDS码。重点分析了3种主要的分布式存储编码,即再生码(RGC)、局部可修复码(LRC)和Piggybacking编码的基本原理、优缺点以及发展现状。总结对比了这5种数据容错技术的性能差异。面向数据的容错存储,针对存储中的节点修复问题,为大数据和移动数据的分布式存储编码提供理论基础,为海量数据的高效、可靠存储提供技术支撑。

关键词:分布式存储;MDS码;再生码;局部可修复码;Piggybacking编码

中图分类号:TN911.22　　**文献标志码**:A　　**开放科学标识码(OSID)**:
文章编号:1003－3114(2019)05－0463－13

Overview of Fault-tolerant Techniques in Distributed Storage Systems

LI Xin[1],SUN Rong[1,2*],LIU Jingwei[1]
(1.State Key Laboratory of ISN,Xidian University,Xi'an 710071,China;
2.Xiamen Key Laboratory of Mobile Multimedia Communications,Huaqiao University,Xiamen 361021,China)

Abstract:The rapid development of the Internet,5G and their related industries has led to an explosive growth of data, which poses a huge challenge to the storage of big data.Large scale distributed storage systems have replaced centralized storage system to become the mainstream systems with their outstanding advantages of massive storage capacity, high throughput, high availability and low cost.Due to the large number of nodes in the distributed storage system,various types of failures often occur,resulting in frequent node failures.Therefore,fault－tolerant techniques must be adopted to ensure that data can be read and downloaded normally even if some storage nodes fail.The problems of storage and reliability in the context of big data are discussed,thus leading to the importance of data storage fault tolerance for the distributed storage.Then the traditional data fault－tolerant technologies,i.e.,multi－copy mechanism and MDS codes,are discussed.More importantly,the fundamentals, advantages and disadvantages,development status of three main distributed storage codes,i.e.,regenerating codes (RGC), locally repairable codes (LRC) and Piggybacking codes,are analyzed.Finally,the performance differences between these five data fault－tolerant techniques are summarized and compared.Aiming at the problem of node repair in the storage,the data－oriented fault－tolerant storage provides theoretical bases for distributed storage codes of big data and mobile data,and provides the technical supports for reliable and efficient storage of massive data.

Key words:distributed storage;MDS codes;regenerating codes;locally repairable codes;Piggybacking codes

0　引言

在当前云计算时代,全球流量快速增长,互联网、物联网、移动终端、安全监控和金融等领域的数据量呈现出"井喷式"增长态势。存储在云服务器中数据的增长速率甚至超越了摩尔法则,云存储系统成为云计算的关键组成部分之一。数据的飞速增长对存储系统的性能和扩展性提出了更苛刻的挑战。

收稿日期:2019－04－23
基金项目:国家自然科学基金面上项目(61771364)

传统的存储系统采用集中式的方法存储数据，使得数据的安全性和可靠性均不能保证，不能满足大数据应用的需求。分布式存储系统以其巨大的存储潜力、高可靠性和易扩展性等优点成为大数据存储的关键系统并被推广应用到降低存储负荷、疏通网络拥塞等业务领域上，且其以高吞吐量和高可用性成为云存储中的主要系统。目前 Hadoop 分布式文件系统(Hadoop Distributed File System，HDFS)作为谷歌文件管理系统(GFS)[1]的开源实现已成为最主流的分布式系统，它被应用于许多大型企业，如 Facebook，Yahoo，eBay，Amazon 等。

为了应对海量数据的存储需求，该类存储系统的规模往往非常庞大，一般包含几千到几万个存储节点不等。早在 2014 年，中国互联网百度公司单个集群的节点数量就超过了 10 000[2]。近两年，腾讯云的分布式调度系统 VStation 管理和调度单集群的节点数量可达 100 000。然而数量庞大的节点集群经常会产生如电源损坏、系统维修及网络中断等故障致使节点失效频发。据一些大型分布式存储系统的统计数据表明，平均每天都会有 2% 左右的节点发生故障[3]。在过去一年中，迅速发展的云计算市场不断涌出如谷歌云、亚马逊 AWS、微软 Azure 及阿里云等主流云服务器大型宕机事件[4]。

由此可见，全球的云服务器发生故障导致数据丢失的情况时有发生。显然，能够可靠存储数据并有效修复失效数据的容错技术对分布式存储系统的重要性不言而喻。因此如何及时有效地修复失效节点，确保数据能被正常地读取和下载就显得非常重要。据统计，在一个有 3 000 个节点的 Facebook 集群中，每天通常至少会触发 20 次节点修复机制。具有节点修复能力的数据容错技术是通过引入冗余的方式来保证分布式存储系统的可靠性，存储系统能够容忍一定数量的失效节点，即提高了系统的数据容错能力。

为了保证用户访问数据的可靠性、维持分布式存储系统的容错性，需要及时修复失效节点。良好的容错技术要求存储系统具有低的冗余开销、低的节点修复带宽、低的编译码复杂度等特点。如何降低失效节点的修复带宽、降低磁盘 I/O、降低存储系统编译码的复杂度、提高系统的存储效率成为分布式存储中研究的热门方向，因此，能够有效弥补多副本机制和 MDS 码不足的分布式存储编码应运而生。

1 传统的存储容错

传统最常见也是最早的存储容错是多副本机制和 MDS 码。通常，多副本机制引入冗余简单，但其存储负载很大，存储效率非常低，如 Google 文件系统和 Hadoop 文件系统等。传统的最大距离可分(Maximum Distance Separable，MDS)码结构可以实现分布式存储编码，比如里德－所罗门(Reed－Solomon，RS)码，以及 Google Colossus，HDFS Raid，Microsoft Azure，Ocean Store 等。其存储成本更低，但是其拥有更高的修复成本和访问延迟，而且没有考虑存储开销、磁盘 I/O 开销等，因此并不适合用于大规模分布式存储系统。

1.1 多副本机制

多副本机制是产生冗余最基本的方式。为了弥补数据失效带来的损失，确保存储系统中数据的完整性，将数据的副本存储在多个磁盘中，只要有一个副本可用，就可以容忍数据丢失。如图 1 所示。如果数据以最常见的 3 副本方式存储，那么原始数据会被复制到 3 个磁盘上，因此任何 1 个磁盘故障都可以通过剩余 2 个磁盘中任意 1 个来修复。显然，该 3 副本机制有效的存储空间理论上不超过总存储空间的 1/3，多副本机制显著降低了存储效率。

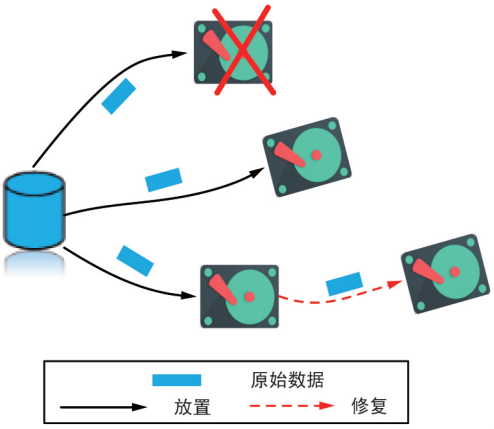

图 1 3 副本机制

1.2 MDS 码

如果系统编码设计的目标是在不牺牲磁盘故障容忍度的前提下最大化存储效率，那么存储系统采用 MDS 码来储存数据，因为它可以为相同的存储开销提供更高的可靠性。也就是说 MDS 码是一种能提供最优储存与可靠性权衡的纠删码。在一个分布式存储方案中，原始文件首先被分成 k 个数据块，接着它们编码生成 n 个编码块并存储在 n 个节点

中。访问任意 $l(l \geqslant k)$ 个节点，通过纠删码的译码就可恢复出原始文件，如果 $l=k$，该码就满足 MDS 性质，最多能够容忍 $(n-k)$ 个节点的失效。如图 2 所示，原始文件被 MDS 码编码生成 5 个数据块并放置到5 个磁盘中。任意某个磁盘损坏，都能通过访问其他 4 个幸存磁盘中的 3 个来修复。

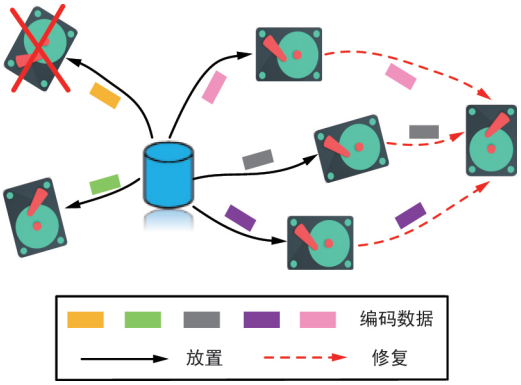

图 2 (5,3)MDS 码修复失效磁盘

采用 MDS 码的分布式存储模型如图 3 所示。将原始文件通过串并变化为长为 m 的 k 块数据(即 $\{d_i\}_{i=1}^{k}$)，分别放入 k 个信息节点中，这些数据块的第 i 位元素组合到一起形成子条带 $a_i(1 \leqslant i \leqslant m)$ 的信息位，每个子条带中的 k 个信息位通过满足编码函数 $\{f_i\}_{i=1}^{k+r}$ 所遵循的 (n,k) MDS 码生成矩阵编码 $n=k+r$ 位。每个独立的子条带构成一个完整的编码信息的集合。所有 m 个子条带上的第 $i(1 \leqslant i \leqslant m)$ 位元素被存储在第 i 个节点中。

MDS 码的 3 个主要缺点阻碍了它在分布式存储系统中的推广。首先，为了读取或写入数据，系统需要对数据进行编译码，由于 CPU 限制会导致高延迟和低吞吐量。根据调研，即使 Facebook 存储系统的集群中用 MDS 码只编码一半数据，修复流量也会使集群中的网络链路接近饱和。其次，MDS 码修复失效节点时必须访问多个编码块，而访问的这些编码块足以得到所有的原始数据，这样的修复代价也太大了。最后，托管在云存储中的应用程序对磁盘 I/O 性能很敏感，且由于大多数数据中心引入链路超额配置；因此带宽始终是数据中心内的有限资源，使得 MDS 码的节点修复在带宽开销和磁盘 I/O 开销方面都非常昂贵。

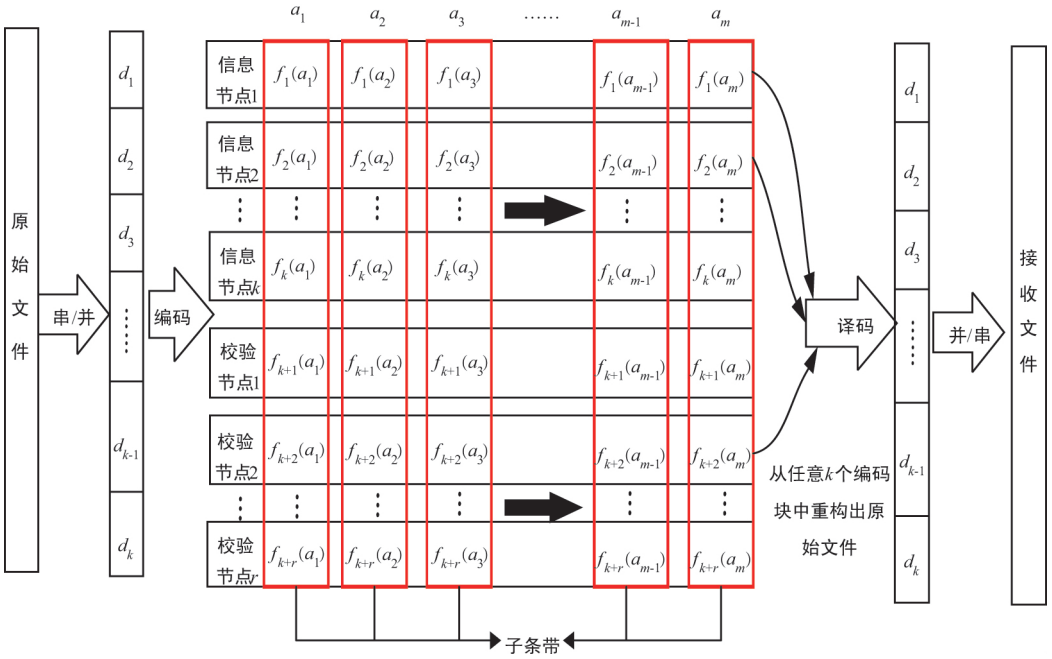

图 3 采用 MDS 编码的分布式储存模型

2 分布式存储编码及性能比较

为了降低修复故障节点的带宽开销，有几种分布式存储编码，即再生码[6](Regenerating Codes，RGC)、局部可修复码[7](Locally Repairable Codes，LRC)和 MDS 阵列码等被提出来，它们都有各自的优缺点。如 RGC 和 LRC 的修复带宽较小，但由于它们在修复过程中有大量的矩阵运算，计算复杂度很高，因此构造过程非常复杂；Piggybacking 编码的研究还处在起步阶段，有很多理论和实际应用还不太完善；MDS 阵列码如 EVENODD，B-code，X-

code，RDP，STAR，Zigzag 码等，它们的编译码过程基本都是构建在低阶域的简单运算之上，复杂度低。但受限于阵列码的构造过程，它们所设计的冗余节点很少，导致其容错能力有限，修复能力一般都比较弱，修复效率较低。目前，在容错技术中存储和修复性能更胜一筹的分布式存储编码主要有 3 类：① 注重存储与修复开销平衡的 RGC；② 面向优化磁盘 I/O 开销的 LRC，③ 不改变系统节点分布结构的 Piggybacking 编码。

2.1 RGC

为了降低 MDS 码的节点修复带宽，Dimakis 等人受网络编码的启发将编码数据的修复建模为信息流图，从而提出了再生码，其约束条件是维持容忍磁盘故障的能力[6]。基于此模型来表征存储与带宽之间的最优权衡，即给定磁盘上存储的数据量大小，可以得到修复过程中需要传输数据量的最优下界。

2.1.1 RGC 原理

在信息流图中，所有的服务器可以分为源节点、存储节点和数据收集器，其中源节点表示数据对象产生的服务器。图 4 为 RGC 信息流图。

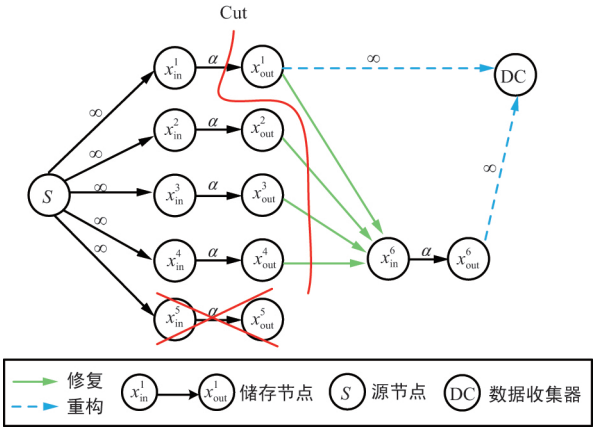

图 4 RGC 信息流图

如图 4 所示，假设再生码是将大小为 M 的原始文件编码（一般为 RS 编码）存储（多播）到 $n(n=6)$ 个节点中，每个节点存储大小为 α。特别是，在信息流图中源节点由一个顶点 S 表示，存储节点由2 个节点 x_{in}^{i}，$x_{out}^{i}(1\leqslant i\leqslant n)$ 表示，边缘的权重对应存储节点中存储的数据量。经过多播后，数据收集器 DC 连接剩余 $(n-1)$ 个存储节点中任意 k 个足以恢复原来的数据对象，这表明 $k\alpha\geqslant M$。当某个节点失效时，重构新的节点是访问幸存 $(n-1)$ 个存储节点中的 $d(d\geqslant k)$ 个，即从每个节点下载大小为 $\beta(\beta\leqslant\alpha)$ 的数据进行修复。因此修复该故障节点所需读取和下载的数据量，也称修复带宽 $\gamma=d\beta$。综上所述，再生码的平均修复带宽 γ 小于文件大小 M。理论上，当连接节点数 $d=n-1$ 时，再生码的修复带宽达到最小值[6]。

MDS 码节点的数据恢复是要新生节点接收来自服务器（数据提供者）的 k 个数据块后，在新生节点上进行编码产生节点新数据。但再生码是在数据提供者和新节点上都能进行编码，即数据提供者（存储节点）有不止一个编码段。当需要修复节点时，提供者先发送自己编码段的线性组合，新节点将接收到的编码段再次编码重构新的节点。如图 5 所示，假定任意 2 个磁盘都足以恢复原始数据的 MDS 码，即 $k=2$，每个磁盘存储一个包含 2 个编码段的编码块。对 MDS 码而言，恢复故障磁盘至少需要给新磁盘发送 2 个编码块（4 个编码段）。但数据经提供者编码后再发送给新磁盘的数据量可减少至 1.5 个编码块（3 个编码段），这样就能减小数据的传输量，降低 25% 的带宽消耗。

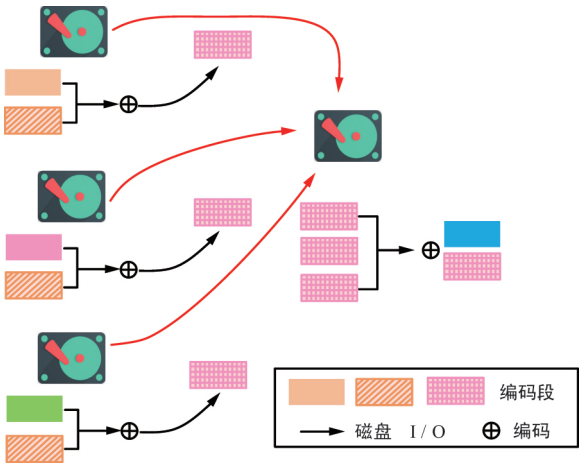

图 5 RGC 基本原理图

Dimakis 等人利用信息流图计算最小割界得到失效节点修复带宽的下限曲线，再生码就是在存储开销 α 和修复带宽 γ 的最优曲线上。该曲线上存在最小存储和最小修复带宽这 2 个极值点。达到存储最小的极值点的编码称为最小存储再生码（Minimum Storage Regenerating Codes，MSR）[7]，达到修复带宽最小极值点的编码称为最小带宽再生码（Minimum Bandwidth Regenerating Codes，MBR）[8]。

MSR 的节点存储大小和节点修复带宽可以表示为：

$$(\alpha_{MSR},\gamma_{MSR})=(\frac{M}{k},\frac{Md}{k(d-k+1)})。$$

MDS 码的存储开销最小，而 MSR 码节点的储存开销就等于传统 MDS 码的节点大小，因此可以被当做 MDS 码。当 $d=n-1$ 时，则上式变为：

$$(\alpha_{MSR},\gamma_{MSR}^{min})=(\frac{M}{k},\frac{M}{k}\cdot\frac{n-1}{n-k})。$$

当 n 趋于无穷大时，其修复带宽等于一个编码块大小，但 MDS 码的修复带宽为 k 倍的编码块大小，因此 MSR 码节省很大的修复带宽。

MBR 的节点存储大小和节点修复开销可以表示为：

$$(\alpha_{MBR},\gamma_{MBR})=(\frac{2Md}{2kd-k^2+k},\frac{2Md}{2kd-k^2+k})，$$

当 $d=n-1$ 时，则上式变为：

$$(\alpha_{MBR}^{min},\gamma_{MBR}^{min})=(\frac{M}{k}\cdot\frac{2n-2}{2n-k-1},\frac{M}{k}\cdot\frac{2n-2}{2n-k-1})。$$

从上式可以看出，MBR 码的节点存储大小和修复开销一致，存储和修复带宽大小比一个编码块稍微大些，因此其修复带宽非常小。

2.1.2 RGC 发展现状

分布式存储网络中一个重要的性能指标是它的安全性。RGC 除了数据存储容错外，还可以应用在信息安全上。它是一种用于提供数据可靠性和可用性的分布式存储网络编码，能实现高效的节点修复和信息安全[9]。其中窃听者可能获得存储节点的数据，也可能访问在修复某些节点期间下载的数据（如果窃听者发现添加到系统中的新节点替换了失效的节点，那么它就可以访问修复期间下载的所有数据，这严重威胁了数据的安全和隐私）。在这种不利环境下，Dimakis 提出了 RGC 的准确构造从而实现了信息保密，即在不向任何窃听者透露任何信息的情况下，可以安全存储并向合法用户提供最大的数据量，同时得到了一般分布式存储系统保密容量的上限，并证明了这个界对于带宽受限的系统来说是紧密的，这在点对点分布式存储系统中是有用的[10]。考虑用分散数据保护一个分布式存储系统的问题，研究了节点间交互在减少总带宽方面的作用[11]。当节点相互独立，交互是没有用的，并且总是存在一个最优的非交互方案。除此之外，RGC 在信息安全保密方面也有研究[12]。

RGC 按照故障节点的修复特点可分为功能性修复和准确性修复。功能性修复是指恢复新节点后使系统仍具有 MDS 性，以维持对故障节点的容忍度，一般的再生码都是功能性修复。准确性修复是指新生节点中的数据与被修复失效节点中的数据相同。因此在前期研究基础上出现了准确 RGC 的概念，当节点失效后能进行准确修复，这意味着系统形式的编码结构可以结合准确重构节点来实现与多副本机制一样低的访问延迟。常用的方法是干扰对齐技术，把不需要的向量对齐到相同的线性子空间以达到消除的目的[13]。

目前大多数关于 RGC 的研究都集中在单一节点的恢复上，但是在大型分布式存储系统中同时看到 2 个或更多的节点故障是很常见的。为了充分利用好修复多个故障节点的契机，一种协同修复机制应运而生，即修复节点之间可以相互共享数据。此类 RGC 被称为合作 RGC[14]，它的出现使得用更低的带宽来修复多个故障节点成为可能。

纵然 RGC 达到了存储与带宽开销之间的最优权衡，但因为其需要繁琐的数学参数和复杂的编码理论基础，实现过程很难。目前 RGC 大多在有限域 $GF(2^q)$ 上进行多项式运算。加法在计算机上处理较为简单，然而乘法和除法却相对复杂，更有甚者需要用到离散对数和查表才能处理。这使得 RGC 的编译码复杂度很高，因而很难满足分布式存储系统对计算复杂度的要求。RGC 作为 MDS 码的改进版，拥有良好的理论基础，但是现有的绝大部分 RGC 编译码复杂度很高且码率低，所以迫切地提出高码率且编译码复杂度低的 RGC，这具有积极的现实意义。若同时结合磁盘 I/O、数据存储安全和网络结构等方面构造，能进一步推广 RGC 的应用范围。

2.2 LRC

与通过牺牲磁盘 I/O 开销来节省带宽消耗的 RGC 不同，LRC 具有较少的磁盘访问数量。它是通过强制一个失效节点中的数据只能通过某些特定的存储节点来进行修复，从而减少需要读取和下载的数据量来降低修复带宽。虽然这种方式牺牲了对节点故障的容忍度（LRC 的理论可译性：能够修复信息理论上可译的故障模式，即 LRC 不具有 MDS 性质），以牺牲少量存储开销为代价（增加了额外的局部校验块），但会显著降低磁盘 I/O 开销。众所周知，磁盘 I/O 开销越大，使用寿命就越短。因此 LRC 在数据容错技术中的地位不容小觑。

2.2.1 LRC 的原理

常规的 RGC 修复故障节点时，都需要请求幸

存节点将其数据的线性组合发送给新节点，即 RGC 仅仅针对存储与修复开销，而没能节省磁盘 I/O。在修复故障节点过程中，RGC 最终从幸存节点读取的数据量将远远多于写入新节点的数据量，过多的磁盘 I/O 操作会严重影响数据中心的总体磁盘 I/O 性能，使得磁盘 I/O 开销成为存储系统中故障节点恢复的一个主要性能瓶颈[15]。磁盘 I/O 开销与修复故障节点时访问的幸存节点数目（即局部修复度）成正比，因此访问的幸存节点数越少（局部修复度越小），磁盘 I/O 开销就越少。需要明确的是，写操作发生在新节点上，且写操作的数据量等于一个存储节点的大小。由于写操作作为重构新节点的最后一步是不可或缺的，因此在实际修复过程中更关心的是从幸存节点中读取的数据量。

MDS 码修复一个节点要求读取 k 个编码块。从图 5 可以看到，RGC 中数据提供者的编码操作不能减少读的数据量，只是减少了数据的传输量，因为发送编码后的数据是需要读取节点内整个编码块（所有编码段）之后再进行编码才生成的。因此，有 2 种方法可以降低磁盘 I/O 开销。第 1 种，从幸存节点中选取特定的编码段，而位于同一节点中的其他编码段因为没有参与编码就不用读取；第 2 种，选取少量的特定节点作为数据提供者而不是任意 k 个节点。特别是，LRC 采用的是第 2 种方法。假定任意 3 个节点都足以得到原始数据的 MDS 码，即 $k=3$。修复一个故障节点，MDS 码和再生码需要读取 6 个编码段，而图 6 中却只需要读取 2 个特定的编码段，所以能降低磁盘 I/O 开销。

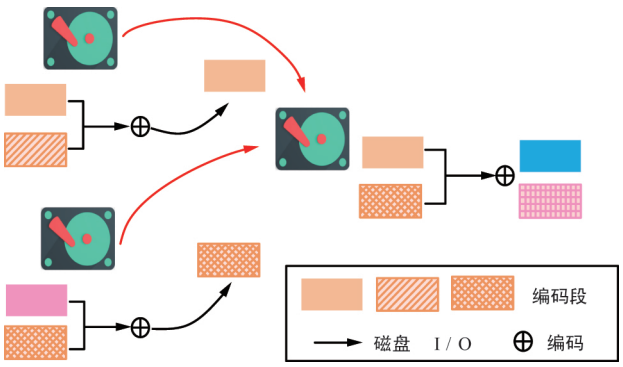

图 6　访问特定编码段（或/和特定节点）来降低磁盘 I/O 开销

LRC 之所以能通过选定特定节点来修复失效节点是因为它以额外的存储开销为代价，任意某个节点都可以通过某些特定的编码块进行修复，减少了修复过程中连接的节点数，也减少了磁盘 I/O 开销。虽然 LRC 具有较好的局部修复特性，但却无法拥有像 MSR 码和 MBR 码等那样较低的存储与修复开销。目前采用 LRC 编码的大型系统主要有 Facebook 的分布式文件系统 HDFS 和微软的 Azure[11]。且 Azure 已广泛应用于微软内部的网络搜索、视频服务、音乐游戏及医疗管理等业务。LRC 的实现方法主要包括分层编码（Hierarchical Codes，HC）和简单再生码（Simple Regenerating Codes，SRC）。HC 是将原始文件进行编码（一般采用 RS 码）之后，进行二次编码得到额外的局部冗余校验块。当需要恢复一个失效节点时，通过访问该故障节点所在分组内的幸存节点和该组内编码产生的局部校验块即可，而并不需要像 MDS 码那样访问更多的幸存节点，因此 HC 可以降低故障节点的修复带宽和磁盘 I/O 操作。Papailiopoulos 等人将可纠正多个错误的 RS 码与简单的异或运算结合，构造出一类 LRC，即 SRC[16]。该方案通过访问少量幸存节点来修复单节点故障，利用特定的放置规则降低了失效节点的修复带宽。

2.2.2　LRC 的发展现状

LRC 除了有数据容错存储的应用外，国际上已经有学者开始尝试利用 LRC 解决分布式存储中的安全问题。2016 年，Kumar 等人为分布式存储设计了一种可修复喷泉码（Repairable Fountain Codes，RFC），在信息安全理论上提出了针对存储节点的窃听模型，应对通过失效节点的修复来窃取其他幸存节点数据的威胁[17]。Rawat 等人通过平衡 LRC 的安全性和修复度，提出了安全性最佳的局部修复度结构的编码方案，可有效抵抗监听[18]。在 2017 年，Kadhe 等人又通过内码和外码的联合设计，构造了一种广泛的安全存储码方案，同时考虑了 MSR 码和 LRC 这 2 种分布式存储编码。

目前，学术上针对单个节点失效 LRC 的研究主要集中在以下的构造方法中：

① 基于二元 LDPC 码的构造：2016 年，Hao 等人阐述了一类有着多修复组的 LRC 与二元 LDPC 码的关系，提出了用正则 LDPC 码去构造 LRC 的方法[19]。2017 年，Su 等人在上述工作基础上基于循环置换矩阵和仿射变换矩阵提出了 2 种二元 LRC 的构造方法，达到了距离限[20]。

② 基于联合信息可用性构造：2017 年，Kim 等人利用联合信息的可用性（任何可译码的纠删符号集可以从任何一组具有小基数的多个不相交符号集

进行修复）提出了 LRC 的 2 个 Alphabet－Dependent 限[21]。

③ 基于图模型的构造：2017 年，Kim 等人还利用具有 r 个顶点和 t 个边的超图构造出了有着（r，t）可用性的二元 LRC[22]。此前，他还提出过 2 个具有联合修复度的 LRC 的独立上限，并利用简单图模型设计出满足该独立上限的 LRC，利用简单图设计出具有最佳码率和联合信息修复度的 LRC[23]。

④ 基于平均修复度的构造：LRC 的平均修复度，影响着修复带宽的大小、磁盘 I/O 的开销。Shahabinejad 等人提出了一个适用于任意 LRC 的平均修复度的下限，并设计出 3 种 LRC。2017 年，他们通过研究信息块的最佳平均修复度，得到了一个更低的下限[24]。

⑤ 基于 Matroid 理论的构造：Westerb 等人给出了拟阵与 LRC 之间的联系，通过线性的或者一般仿射来使用这些拟阵结构，可以构造新的 LRC[25]。Tamo 等人也通过拟阵提出了简单准确的 LRC 构造[26]。

⑥ 基于 Rank－Metric 码的构造：Silberstein 等人基于最大秩距离（Maximum Rank Distance，MRD）Gabidulin 码提出了 LRC 的构造方法[27]，采用串行级联编码的结构，用 MDS 码作为外码，RGC 或 LRC 作为内码，实现容错存储。

2.3 Piggybacking 编码

相比于 RGC 和 LRC，Piggybacking 编码以其较低的计算复杂度、设计灵活等特点开始在数据容错技术中占有一席之地。编码设计完成后，它不改变原有系统的节点分布结构，不产生额外的存储负载、在修复失效数据时将底层码的译码替代为求解与失效数据相关的 Piggyback 方程。这样不仅显著降低了修复带宽，还有效降低了修复复杂度，为海量数据可靠、高效的存储提供了强有力的保障。

2.3.1 Piggybacking 编码的原理

2013 年 Rashmi 等人首次提出了 Piggybacking 框架的概念，且将该框架下构造的 Piggybacking 编码用来应对存储系统中频发的故障节点[28]。他们解释 Piggybacking 的含义是将 MDS 码作为基本码，将扩展后的 MDS 码中某些条带的数据符号通过精心设计好的 Piggyback 函数嵌入到其他条带中形成 Piggyback 块的一种设计。失效的数据符号可以通过求解 Piggyback 方程来替代传统的 MDS 译码来恢复，这样能有效降低修复带宽。

Piggybacking 编码的框架是在一个任意的、被称为基本码的基础上操作，目前的基本码都采用 MDS 码。Piggybacking 编码保持了基本码的很多属性，比如最小距离和操作域。Piggybacking 框架有一个显著的特点就是在不改变原有存储节点的分布结构下减少了额外的存储开销，即不需要增加除 MDS 编码以外的校验节点，依旧能保持数据重建性。另外，Piggybacking 编码满足 MDS 性质可以实现高码率，拥有少的子条带数和修复节点时更少的平均数据读取量和下载量等特点。另一个特别吸引人的点就是 Piggybacking 编码支持任意码长为 n、信息位长度为 k 的基本码（码参数的选择取决于基本码的需要）。然而有的码如 Rotated－RS 码，当校验节点数 $r\in\{2,3\}$ 和信息节点数 $k\leqslant 36$ 时才存在[29]；EVENODD 码和 RDP 码也只在 $r=2$ 时才可以实现。再加上 Piggybacking 编码设计灵活、复杂度低和易于系统实现等特点，目前已经应用于 Facebook Warehouse Cluster，HDFS 等系统中[30]。通常 Piggybacking 编码采用系统形式，因为系统形式中的信息节点承载了所有未编码的原始数据。因此用户通过直接访问信息节点就可以得到原始数据，而不用访问某些校验节点采用 MDS 译码来恢复原始数据，这样不仅降低了获取原始数据的复杂度，还降低了获取数据的延迟。

Piggybacking 框架如图 7 所示，对每一个条带 $i(2\leqslant i\leqslant m)$，把前例 $\{1,2,\ldots,(i-1)\}$ 的信息符号 $\{a_i\}_{i=1}^{m}$ 作为独立的输入设计成 Piggyback 函数 $g_{i,j}\{i\in(2,\ldots,m),j\in(1,\ldots,n)\}$ 嵌入到 $\{f_i(a_k)\}_{i=1,k=2}^{i=n,k=m}$ 并存储到第 i 列的条带中，这些值称为 Piggyback 块。将条带数为 $\{1,2,\ldots,(i-1)\}$ 的原始符号线性组合作为 Piggyback 函数增加到第 i 个条带上的操作称为“Piggybacking 约束”，保证整个 Piggybacking 框架可译。Piggybacking 框架允许对一行（一个存储节点）中存储的数据符号进行任何可逆的线性变换，很容易重新变化出这些数据符号以满足上述 Piggybacking 约束[29]。Rashmi 等人证明了 Piggybacking 设计没有减少任何一个节点子集中所存储的数据量，也就是在任意可译的节点子集中，原始数据都可以被恢复，即 Piggybacking 编码保持基本码的 MDS 性质。

由图 7 可以发现，Piggybacking 编码的第 1 列与基本码编码的一个实例相同。因此 a_1 可以通过使用这个基本码的译码直接恢复出来，Piggyback

函数 $\{g_{2,i}(a_1)\}_{i=1}^{n}$ 可以在第 2 列中被减掉，则第 2 列中剩下的 $\{f_i(a_2)\}_{i=1}^{n}$，确切而言是另一个基本码，同 a_1 一样可以译出 a_2。继续同样的方法迭代到所有的 $i(2 \leqslant i \leqslant m)$，Piggyback 块被减掉来获得此列的可译基本码，由此恢复出其余的 MDS 实例。由于 Piggybacking 编码的译码性质不受限于所选择的 Piggyback 函数 $g_{i,j}$，因此可以灵活设计 Piggyback 函数，可使 Piggybacking 编码实现期望的目标，如降低修复带宽、降低修复度等。

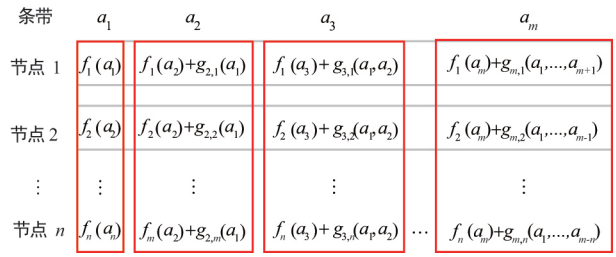

图 7　Piggybacking 框架

2.3.2　Piggybacking 编码的发展现状

作为 Piggybacking 编码的鼻祖，Rashmi 等人提出了 3 种 Piggybacking 设计[29]。第 1 种设计是基于较少的子条带数量，其中最少的子条带数只有 2。这类设计修复信息节点时根据码参数不同可以节省 25%～50%不等的修复带宽，且该设计也具备修复校验节点的能力。经计算表明，当整个 Piggybacking 框架采用的条带(例)数 m 越大时，修复带宽越低，修复效率越高。第 2 种设计是由 $(2r-3)$ 列结构相同的 MDS 码组成，并在最后 $(r-1)$ 列中增加了 Piggyback 方程。该设计要求校验节点数 $r \geqslant 3$，相比第 1 种设计拥有更高的修复效率，平均能有效减少约 50%的修复带宽。这是以牺牲子条带数量为代价：要求最少子条带数为 $(2r-1)$，且该设计不能有效修复校验节点。作者将关注度从修复带宽转移到修复度上，设计了第 3 种 Piggybacking 编码，该码可以有效修复最小修复度可达 $k+1$ 的 MDS 码中的信息节点。

随着 Piggybacking 编码被提出，研究者们正试图寻找构造有效的 Piggybacking 框架来修复失效节点。Yuan 等人提出一种广义的 Piggybacking 编码，并证明了当校验节点数趋于无穷大时，信息节点的修复带宽比率趋近于零，而且非常接近分布式存储编码修复带宽的理论极限(cut－set bound[31])。Kumar 等人针对信息节点的修复设计了一种 Piggybacking 框架，然而该 Piggybacking 结构并没有维持 MDS 性质[32]，且当 B 类校验节点数增加时，信息节点的修复带宽会减少，同时也增加了系统的存储负载。Shangguan 等人提出了一种 Piggybacking 编码用于修复信息节点，他们根据不同 Piggyback 块中嵌入的信息符号数量的不同将信息节点进行划分，由此获得了较低的修复带宽[33]。Yang 等人保证信息节点可修复的同时，为减少校验节点的修复带宽提出了一种 Piggybacking 框架[34]。

2.3.3　Piggybacking 编码的理论研究、构造与应用

Piggybacking 编码的研究主要处在理论和构造阶段，因此从 Piggybacking 码的理论研究、构造和可能应用的这 3 个方向对其研究内容进行阐述，其结构如图 8 所示。

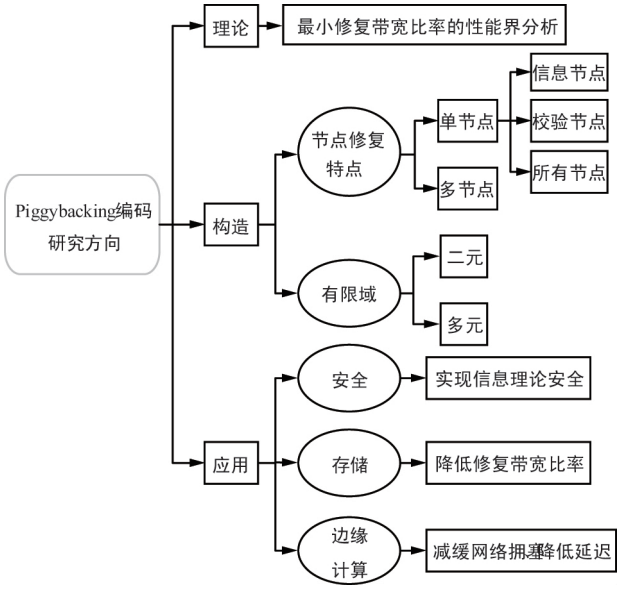

图 8　Piggybacking 编码的研究方向

2.3.3.1　理论研究

Piggybacking 编码通常是以 MDS 码作为基本码，然而 MDS 码的弊端很多，比如：系统中 MDS 码的编译码操作经常会导致高延迟和低吞吐量的情况，MDS 码的修复代价太高，MDS 码修复度较大，对磁盘 I/O 性能不友好。更加遗憾的是，由于 MDS 码的编译码复杂度很高，会导致 Piggybacking 编码的编译码复杂度也不低。因此对于 Piggybacking 编码来说，选择复杂度低的基本码是降低自身复杂度最快、也是最有效的方法。比如可以考虑将复杂度较低的类 LDPC 码来作为基本码。

一种 Piggybacking 编码性能的好坏，不仅在于其具有较低的修复带宽比率，更希望它能有更快的收敛速度，有最小的修复带宽比率下界。因此，需要

对该码的最小修复带宽比率的性能界进行分析。收敛速度越快，说明该码的最小修复带宽比率下降速度越快；最小修复带宽比率能达到的下界越低，说明该码的修复性能越好。

2.3.3.2 构造

理论上基于有限域构造，通过 Piggyback 块构造后产生的 Piggybacking 编码不改变基本码的有限域，也就是说 Piggyback 函数设计得到的 Piggybacking 编码不仅适用于二元域，也同样适用于多元域 F_q（q 为素数幂）。根据面向修复节点特点主要分为单节点修复和多节点修复，具体如下。

(1)单节点修复

在 Piggybacking 编码的构造阶段，也就是设计 Piggybacking 的框架，目前所有研究只是停留在单节点的修复设计上，而且大部分都是针对信息节点的修复，检验节点的非常少，而同时有效修复所有节点的 Piggybacking 码就更少。本小节针对这 3 种编码设计，分析构造它们所采取的构造思路如下：

a.信息节点的修复

一种 Piggybacking 编码能够修复信息节点，是因为 Piggyback 函数中嵌入了信息节点中的原始信息数据，然后将这些 Piggyback 函数嵌入到不同校验节点中的不同子条带(或条带)中。显然，当信息节点失效时，带有信息节点中失效信息数据符号的 Piggyback 函数存在于没有损坏的校验节点中，通过这些函数构造的逆过程即求解 Piggyback 方程，可得到失效数据。但当校验节点失效时，失效的校验符号没有留下任何与之相关的 Piggyback 函数(块)，因此该 Piggybacking 编码就无法通过求解 Piggyback 方程来恢复，只能通过该 Piggybacking 编码基本码的 MDS 性质译出。但这样的修复带宽是大于通过求解 Piggyback 方程来恢复失效数据的修复带宽，且当基本码的编码参数增大(分布式存储系统规模的增大)，通过 MDS 性质译出的修复带宽是远远大于通过求解 Piggyback 方程的。

对于不同的 Piggybacking 编码而言，其最小修复带宽比率(或修复效率)与修复带宽比率的收敛速度均不相同。这与该 Piggybacking 编码的 Piggyback 函数和该 Piggyback 块放置的位置息息相关。一般情况 Piggybacking 编码的修复带宽比率越低，修复效率越高，其 Piggyback 函数构造的复杂度就越高，方法就越巧妙。

通常可以采用整体法和分集法。整体法是将所有信息节点中的信息数据看成一个大的集合，需要构造出一种 Piggyback 函数，将这些数据按照此编码规则统一进行嵌入；分集法是将信息节点进行分组(理论分析，一般均分后的性能是最优的)，针对每个集合需要对应构造出一种 Piggyback 函数，不合集合之间的函数不同，将各自集合中的信息数据采用与之对应的 Piggyback 函数进行嵌入。经分析，分集法的修复带宽比率的收敛速度大于整体法，因此它的修复效率更高，复杂度也较高。

b.校验节点的修复

同理，一种 Piggybacking 编码能够修复校验节点，是因为 Piggyback 函数中嵌入了校验节点中的校验符号数据，将这些 Piggyback 函数嵌入到不同校验节点中的不同条带(或子条带)中。与修复信息节点的 Piggybacking 编码相比不同的是：该 Piggyback 块也需加到校验节点上，而不是信息节点上。通常为了用户方便有效读取和下载数据，分布式存储系统一般是系统节点，也就是当系统中信息节点没有损坏时用户只需读取下载信息节点中自己需要的数据即可，无需进行 MDS 译码操作，这样可以大大降低复杂度和读取延迟。只有当信息节点或校验节点失效时，才会动用嵌在校验节点中的 Piggyback 块进行失效节点的修复。

由于修复校验节点的校验块就放在校验节点上，因此该种 Piggybacking 编码就有了一个与修复信息节点 Piggybacking 编码不同的设计关键：校验节点的修复依赖与它相关的校验节点，因此必须保证每个校验节点的 Piggyback 块(用于修复校验节点)不能嵌入本校验节点的校验符号，假如该校验节点失效后，该节点中的校验符号连同与其唯一相关的 Piggyback 块一起丢失，该符号就无法通过 Piggyback 方程译出。为了提高 Piggyback 方程的可解性，即保证每个数据符号都能被正确恢复出，要求每个失效信息节点里的待嵌入数据符号不能嵌入同一个 Piggyback 块里(避免出现方程式的个数少于未知数个数的情况)，因此这些数据符号需要尽可能地散列，当然这个条件对修复信息节点的设计也需要满足。为了设计修复带宽比率低的用于修复校验节点的 Piggybacking 编码，除了满足上面的设计要求，还在于 Piggyback 函数的巧妙设计。也可采用整体法和分集法，分析比较它们之间的修复效率。

c.所有节点的修复

经过对以上 2 种节点修复方法的简单描述，本

节是对所有节点的修复分析方法进行阐述。该种Piggybacking编码的构造要求是需要同时满足对信息节点和校验的有效修复。用于修复信息节点和校验节点的Piggyback块都要嵌入校验节点中。因此，Piggybacking编码可分为两大类：一类是将修复信息节点的Piggyback块和修复校验节点的Piggyback块混合嵌入到校验节点中；另一类是，将这2种Piggyback块分开嵌入校验节点中。当这2种校验块出现混合嵌入同一个校验节点的同一列时，必须清楚当修复信息节点或校验节点时，用于修复另一种节点的Piggyback块是否会对正在修复节点的修复带宽产生影响，这样是否会引入额外的修复带宽（相比于单一节点修复的Piggybacking编码），以及Piggyback函数的设计复杂度增加的程度。当这2种校验块放置的子条带没有混合时，无形中又增加了Piggybacking编码框架中的校验块的嵌入子条带数，这样是否会影响该Piggybacking编码设计的参数（子条带数）要求。把握好上述2种情况，才能设计出复杂度低、同时有效修复2种节点的Piggybacking编码。

（2）多节点修复

目前所有关于Piggybacking编码的研究，针对失效节点的修复都是单节点修复。但Piggybacking编码的多节点修复可以类似RGC和LRC，且在实际的分布式存储系统中有着重要的现实意义，这对深入研究Piggybacking编码有一定的参考价值。

不同Piggybacking编码的设计框架不同，即针对其修复节点的方法不尽相同。因此，不同Piggybacking编码的多节点修复方法也不同。最重要的是，多节点的修复情况相比单节点更加复杂，包括3种情况：① 失效的全部都是信息节点；② 失效的全部都是校验节点；③ 失效的一部分是信息节点，一部分是校验节点。对于同一种Piggybacking编码，其信息节点和校验节点的修复带宽不同，因此以上3种情况的修复带宽都不相同，必须分情况讨论。最终通过概率论与数理统计的方法求得平均多节点修复带宽比率。另外，需注意的是，多节点修复的失效节点数以2起步，小于等于校验节点数r（其基本码MDS码能容忍的最大失效节点数为$n-k=r$）。

多节点的修复过程中会出现一种权衡，失效节点中的失效符号（需要求解未知数的个数）与Piggyback方程个数的关系。显然，Piggyback方程越多，利用线性方程能求解的未知数（失效节点中的符号数）就越多，则通过MDS译码求解剩余未知数减少，总的消耗带宽就小（最差的情况就是所有失效节点全部用MDS译码，即大小等于原始数据大小）。所以希望通过MDS译码求解的符号数应尽量少。但又不能太少，否则Piggyback方程的个数少于剩余需要求解符号数的个数时，又无法求解。因此，必须针对不同的Piggybacking编码，找到它的修复带宽的权衡，才能求得其最小的多节点修复带宽比率。

可以先对该Piggybacking编码的2个失效节点进行分析总结，找出它的权衡，再将这种方法推广到多个修复节点的推算中，难易程度循序渐进，且准确率更高。

2.3.3.3 应用

（1）存储

Piggybacking编码作为分布式存储编码中的新生力量，诞生于大数据、云计算时代。目前它的主要应用是存储业务，并已经应用到了Facebook Warehouse Cluster，HDFS等系统中。

（2）安全

Piggybacking编码与同属于分布式存储编码的RGC和LRC一样，都具有维护信息理论安全的潜力。因此，可以使Piggybacking编码既能成为一种执行高效、复杂度低的节点修复，同时还能实现信息数据的安全性。

考虑一个理论上能应对在存储节点的窃听模型，并通过损坏节点的修复得到其他节点的数据。其中的窃听者可能获得存储节点的数据，也可能访问在修复某些节点期间下载的数据，通过设计安全可靠的Piggyback函数，既能对失效节点进行有效修复，还能使窃听者获得的数据无法解密译出，从而确保信息的保密能力。在不向任何窃听者透露任何信息的情况下，可以安全存储并向合法用户提供最大的数据量。最后通过证明窃听者和原始数据之间的互信息为0，即窃听失败，证明Piggybacking编码的安全模型可行。可以对信息理论安全和Piggyback函数设计的复杂度进行权衡分析，提出理论安全和具有最低Piggyback函数设计的复杂度二者结合的Piggybacking编码方案可抵抗窃听。另外，除了在Piggyback函数的设计（编码）上考虑，还可以结合信息安全方面的知识建立安全模型。

（3）边缘计算

如果在分布式存储系统中完全依赖云中心去修复失效节点，不仅会占用大量的网络带宽，

还会导致整个系统的性能瓶颈变为网络带宽的瓶颈，因为云中心的处理速度要远超网络中数据的传输速度。若节点的修复计算能搬移到边缘节点，且只向云中心提交处理后的结果，不仅能降低不必要的带宽消耗，还能减少修复延迟，从而进一步提高系统性能。另外，就地处理数据，可以大大降低数据在网络传输中被窃取的风险，有效进行隐私保护，保障数据安全。

协同边缘如图 9 所示，它连接了多个在物理位置或网络结构中分散的边缘节点。边缘节点作为一个将终端用户和云中心连接具有数据处理能力的小型数据中心，也是逻辑概念的一部分[35]。实线箭头代表云服务器中心给边缘节点下发任务，虚线箭头表示多个终端用户将自己产生的数据交给具有计算能力的边缘节点进行处理。在网络边缘处，节点不仅可以从云中心请求服务和内容，还可以在本地执行计算卸载、数据存储、缓存和处理以及物联网管理等任务。基于这些能力，需要对边缘节点进行良好设计，保证存储系统中数据的可靠、安全和隐私[36]。综上所述，这些分布式边缘节点为终端用户提供了合作和共享数据的机会。因此，边缘计算具有极大潜力，可以通过与 Piggybacking 编码结合来安全高效地解决分布式存储系统中失效节点的修复难题。

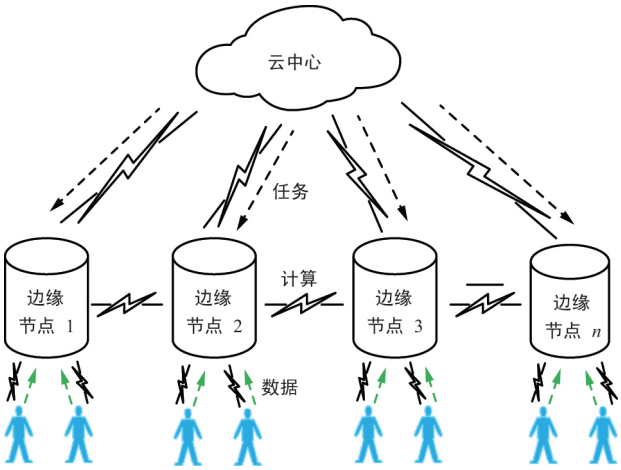

图 9　分布式存储中协同边缘计算范式的结构

2.4　性能比较

表 1 针对文中 5 种存储容错方式的性能进行了简单粗略比较（注：从这几种容错存储的基本原理来考虑，并没有针对具体的容错参数或码参数）。其中“★”的数量表示性能的相对大小程度，即 4 颗星表示最高，3 颗星表示较高，2 颗星表示较低，1 颗星表示最低。例如多副本机制的修复带宽开销最低，用 1 颗星表示；其存储开销最大，用 4 颗星表示。

表 1　几种存储容错方式的性能比较

存储容错	修复带宽销	存储开销	修复度	计算复杂度	容错能力
多副本机制	★	★★★★	★	★	★★★★
MDS 码	★★★★	★	★★★	★★★	★★★
RGC	★★	★	★★★★	★★★★	★★★
LRC	★★★	★★	★★	★★★★	★★
Piggybacking 编码	★★	★	★★★	★★	★★★

3　结束语

本文简单介绍了目前分布式存储系统中常见的容错技术，主要分为 4 部分：① 传统的容错技术即最早的多副本机制和 MDS 码；② 基于网络编码的再生码；③ 着眼于降低磁盘 I/O 的局部可修复码；④ 不改变存储结构的 Piggybacking 编码。传统的容错技术与分布式存储编码的容错相比，其基本性能劣势比较明显，慢慢会被取代。不同的分布式存储编码都有自己优缺点，具体选择哪种容错方式要根据分布式存储系统的需求来决定。

参 考 文 献

[1] Ghemawat S, Gobioff H, Leung S T. The Google file system[J]. ACM SIGOPS Operating Systems Review, 2003, 37(5): 29.

[2] Shvachko K, Kuang H, Radia S, et al. The Hadoop Distributed File System[C] // Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium, 2010: 1－10.

[3] Schroeder B, Gibson G A. Disk Failures in the Real World: What does an MTTF of 1, 000, 000 Hours Mean to You? [C] // In Proc. FAST' 07, USENIX

Conference on File and Storage Technologies (2007), USENIX, 2007: 1－16.

[4] 赵钰莹. 2018 年十大云宕机事故盘点[OB/OL]. https://www.infoq.cn/article/4pSNXHT4PuI4T*L8g1Sk?from=timeline&isappinstalled=0.

[5] Kubiatowicz J, Bindel D, Chen Y, et al. Oceanstore: an Architecture for Global－scale Persistent Storage[J]. ACM SIGARCH Computer Architecture News, 2000, 28: 190－201.

[6] Dimakis A G, Godfrey P B, Wu Y, et al. Network Coding for Distributed Storage Systems [J]. IEEE Transactions on Information Theory, 2010, 56(9): 4539－4551.

[7] Rashmi K V, Shah N B, Kumar P V. Optimal Exact－regenerating Codes for Distributed Storage at the MSR and MBR Points Via a Product－matrix Construction [J]. IEEE Transactions on Information Theory, 2011, 57(8): 5227－5239.

[8] Mahdaviani K, Mohajer S, Khisti A. Product Matrix Minimum Storage Regenerating Codes with Flexible Number of Helpers [C] // Information Theory Workshop (ITW), 2017 IEEE, 2017: 41－45.

[9] Shah N B, Rashmi K, Kumar P V. Information－theoretically Secure Regenerating Codes for Distributed Storage[C] // Global Telecommunications Conference (GLOBECOM 2011), 2011 IEEE, 2011: 1－5.

[10] Pawar S, El Rouayheb S, Ramchandran K. On Secure Distributed Data Storage under Repair Dynamics[C] // Information Theory Proceedings (ISIT), 2010 IEEE International Symposium, 2010: 2543－2547.

[11] El Rouayheb S, Prabhakaran V, Ramchandran K. Secure Distributive Storage of Decentralized Source Data: Can Interaction help? [C] // Information Theory Proceedings (ISIT), 2010 IEEE International Symposium, 2010: 1953－1957.

[12] Kadhe S, Sprintson A. Universally Weakly Secure Coset Coding Schemes for Minimum Storage Regenerating (MSR) Codes [C] // Communication, Control, and Computing (Allerton), 2017 55th Annual Allerton Conference, 2017: 25－30.

[13] Suh C, Ramchandran K. On the Existence of Optimal Exact－repair MDS Codes for Distributed Storage[J]. arXiv Preprint arXiv: 1004.4663, 2010.

[14] Kermarrec A M, Straub G, Scouarnec N. Repairing Multiple Failures with Coordinated and Adaptive Regenerating Codes[C] // Network Coding (NetCod), 2011 International Symposium, 2011: 1－6.

[15] Zhou T, Li H, Zhu B, et al. STORE: Data Recovery with Approximate Minimum Network Bandwidth and Disk I/O in Distributed Storage Systems[C] // Big Data (Big Data), 2014 IEEE International Conference, 2014: 33－38.

[16] Papailiopoulos D S, Luo J, Dimakis A G, et al. Simple Regenerating Codes: Network Coding for Cloud Storage [C] // INFOCOM, 2012 Proceedings IEEE, 2012: 2801－2805.

[17] Kumar S, Rosnes E, Amat A G. Secure Repairable Fountain Codes [J]. IEEE Communications Letters, 2016, 20(8): 1491－1494.

[18] Rawat A S, Koyluoglu O O, Silberstein N, et al. Optimal Locally Repairable and Secure Codes for Distributed Storage Systems[J]. IEEE Transactions on Information Theory, 2014, 60(1): 212－236.

[19] Hao J, Xia S T. Constructions of Optimal Binary Locally Repairable Codes with Multiple Repair Groups [J]. IEEE Communications Letters, 2016, 20(6): 1060－1063.

[20] Su Y S. On Constructions of a Class of Binary Locally Repairable Codes with Multiple Repair Groups [J]. IEEE Access, 2017, 5: 3524－3528.

[21] Kim J H, Song H Y. Alphabet－dependent Bounds for Locally Repairable Codes with Joint Information Availability[J]. IEEE Communications Letters, 2017, 21(8): 1687－1690.

[22] Kim J H, Song H Y. Hypergraph－based Binary Locally Repairable Codes with Availability [J]. IEEE Communications Letters, 2017, 21(11): 2332－2335.

[23] Kim J H, Nam M Y, Song H Y. Optimal Binary Locally Repairable Codes with Joint Information Locality[C] // Information Theory Workshop－Fall (ITW), 2015 IEEE, 2015: 54－58.

[24] Shahabinejad M, Khabbazian M, Ardakani M. Locally Repairable Codes with the Optimum Average Information Locality[C] // Information Theory (ISIT), 2017 IEEE International Symposium, 2017: 181－185.

[25] Westerbäck T, Ernvall T, Hollanti C. Almost Affine Locally Repairable Codes and Matroid Theory[C] // Information Theory Workshop (ITW), 2014 IEEE, 2014: 621－625.

[26] Tamo I, Papailiopoulos D S, Dimakis A G. Optimal Locally Repairable Codes and Connections to Matroid Theory[J]. IEEE Transactions on Information Theory, 2016, 62(12): 6661－6671.

[27] Rawat A S, Silberstein N, Koyluoglu O O, et al. Optimal Locally Repairable Codes with Local Minimum Storage Regeneration Via Rank－metric Codes[C] // Information Theory and Applications Workshop (ITA), 2013: 1－8.

[28] Rashmi K, Shah N B, Ramchandran K. A Piggybacking Design Framework for Read-and Download-efficient Distributed Storage Codes[C] // Information Theory Proceedings (ISIT), 2013 IEEE International Symposium, 2013: 331-335.

[29] Khan O, Burns R C, Plank J S, et al. Rethinking Erasure Codes for Cloud File Systems: Minimizing I/O for Recovery and Degraded Reads. [C] // FAST, 2012: 20.

[30] Rashmi K V, Shah N B, Gu D, et al. A Solution to the Network Challenges of Data Recovery in Erasure-coded Distributed Storage Systems: A Study on the Facebook Warehouse Cluster[C] // Proc. 5th USENIX Conf. HotStorage, 2013: 8.

[31] Yuan S, Huang Q. Generalized Piggybacking Codes for Distributed Storage Systems[C] // Global Communications Conference (GLOBECOM), 2016 IEEE, 2016: 1-6.

[32] Kumar S, Amat A G I, Andriyanova I, et al. A Family of Erasure Correcting Codes with Low Repair Bandwidth and Low Repair Complexity[C] // Global Communications Conference (GLOBECOM), 2015 IEEE, 2015: 1-6.

[33] Shangguan C, Ge G. A New Piggybacking Design for Systematic MDS Storage Codes [J]. arXiv preprint arXiv: 1610.08223, 2016.

[34] Yang B, Tang X, Li J. A Systematic Piggybacking design for Minimum Storage Regenerating Codes[J]. IEEE Transactions on Information Theory, 2015, 61 (11): 5779-5786.

[35] Bonomi F, Milito R, Zhu J, et al. Fog Computing and its Role in the Internet of Things[C] // Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing, 2012: 13-16.

[36] Shi W, Cao J, Zhang Q, et al. Edge Computing: Vision and Challenges[J]. IEEE Internet of Things Journal, 2016, 3(5): 637-646.

作者简介：

李　鑫(1994—)，**男，硕士，毕业于西安电子科技大学，现就职于中兴通讯。主要研究方向：分布式存储编码、5G、信息传输与编码、边缘计算。**

(* 通信作者)孙　蓉(1976—)，**女，2008 年 1 月在西安电子科技大学获得通信与信息系统博士学位，西安电子科技大学副教授，硕士生导师，IEEE 会员，中国电子学会信息论分会会员。主要研究方向：信息论、无线传输、信道编码与调制、分布式编码与存储等。近年来主持参与国家自然科学基金、国家 973 计划、国家 863 计划等多个项目，在国内外权威学术期刊及国际会议上发表论文 30 余篇，授权发明专利 10 余项。**

刘景伟(1978—)，**2007 年 6 月在西安电子科技大学获得通信与信息系统博士学位，西安电子科技大学副教授，博士生导师，IEEE 会员，中国密码协会会员，信息安全高级测评师，陕西省区块链与安全计算重点实验室测评部负责人。主要研究方向：区块链安全、物联网(IoT)安全、云计算、隐私保护、无线/移动网络安全、信息安全。近年来主持国家自然科学基金、国家科技重大专项子课题、陕西省重点研发计划等多个项目，出版学术专著 1 部，在国内外权威学术期刊及国际会议上发表论文 50 余篇，授权发明专利 11 项。**