分布式存储系统中支持近数据处理的纠删码技术

李浩然,黄志杰*,史宇龙,赵承佳,赵楠楠,张 晓
(西北工业大学计算机学院,陕西西安 710072)

摘 要: 纠删码技术和近数据处理技术是构建高效的云边端协同数据管理系统的两大基石,前者通过对数据添加编码冗余方式来保障系统的可用性,而后者则通过在存储端处理数据的方式避免大量的网络传输开销. 云边端协同的数据管理系统通常采用成熟的分布式存储系统作为底层存储引擎,然而主流的分布式存储系统中的纠删码实现方式并不能高效地支持近数据处理. 本文提出了一种支持近数据处理的纠删码技术架构,其基本原理是通过对待编码的一组数据进行重新布局,保证语义相关数据被存储在同一个存储设备上,避免执行近数据处理时的跨节点数据传输. 该方案在分布式存储系统 Ceph 上获得实现,并测试典型场景的读写性能. 实验结果表明,在近数据处理场景下和常规数据读取场景下,读取对象的性能分别提升 59.4% 和 10%,对象写入性能则与原版保持一致.

关键词: 纠删码;分布式存储;Ceph;近数据处理;云边端协同数据管理

基金项目: 国家自然科学基金(No.62272394,No.62202382);国家重点研发计划(No.2022YFB2702101);广东省基础与应用基础研究基金(No.2021A1515110080)

中图分类号: TP302 **文献标识码:** A **文章编号:** 0372-2112(2025)02-0344-10

电子学报 URL:http://www.ejournal.org.cn **DOI:**10.12263/DZXB.20240611

An Erasure Coding Technology Supporting Near-Data Processing in Distributed Storage Systems

LI Hao-ran, HUANG Zhi-jie*, SHI Yu-long, ZHAO Cheng-jia, ZHAO Nan-nan, ZHANG Xiao
(*School of Computer Science*, *Northwestern Polytechnical University*, *Xi'an*, *Shaanxi* 710072, *China*)

Abstract: Erasure coding and near-data processing are two cornerstones for building efficient cloud-edge-end collaborative data management systems. The former ensures system availability by adding coding redundancy to the original data, while the latter avoids significant network transmission overhead by processing data at the storage end. Cloud-edge-end collaborative data management systems typically adopt mature distributed storage systems as the underlying storage engine. However, the erasure coding implementation in mainstream distributed storage systems can not efficiently support near-data processing. This paper proposes an erasure coding architecture that supports near-data processing. Its basic principle is to reorganize the data to be encoded to ensure that semantically related data is stored in the same storage device, thereby avoiding cross-node data transmission during near-data processing. The scheme has been implemented on the distributed storage system Ceph, and the read and write performance under typical scenarios are tested. The experimental results show that the performance of reading objects in the near-data processing scenario and the conventional data reading scenario are improved by about 59.4% and 10% respectively, while the object writing performance remains consistent with the original version.

Key words: erasure coding; distributed storage; Ceph; near-data processing; cloud-edge-end collaborative data management

Foundation Item(s): National Natural Science Foundation of China (No. 62272394, No. 62202382); National Key Research and Development Program (No. 2022YFB2702101); Guangdong Basic and Applied Basic Research Foundation (No.2021A1515110080)

收稿日期:2024-07-01;修回日期:2024-10-27;责任编辑:孙瑶
*通讯作者:黄志杰

1 引言

互联网的飞速发展极大加快了信息获取的速度和便捷,同时催生了海量数据的生成和累积,标志着社会进入了大数据时代. 随着云边端技术的兴起和应用,数据存储系统不仅需要处理大规模数据的存储问题,还要应对在复杂和分布式环境中的数据安全性和可靠性挑战. 在云边端架构中,数据的生成、处理和存储分布在云服务器、边缘服务器和终端设备上,系统规模的增大增加了数据丢失的风险,使存储系统中数据的可靠性成为学术界和工业界关注的焦点. 为应对这一挑战,最简单的办法是为每份数据存储多个副本,这会导致存储效率降低,增加系统总体运行成本. 因此,近年云边端协同的海量数据存储系统开始采用纠删码技术[1~3],实现数据的高可用性和低存储成本.

“纠删码”这一概念最早起源于通信系统中的数据传输领域,用来处理数据在信道传输时的可靠性问题. 在进行信号传输时,由于受到多种因素的干扰和损耗,数据信号会存在一定的错误率和丢失率,给数据传输带来很大的困难. 为解决这一问题,纠删码这一技术应运而生. 纠删码技术的基本原理是将需要传输的数据信号进行分段,在每一段数据中加入一定的校验码,这些校验码可以用来检测数据是否出现了错误或丢失. 存储系统本身可以看作是一种特殊信道,纠删码也可以应用到存储系统以对抗硬盘(节点)故障. 在存储系统中纠删码技术的主要原理是首先针对原始数据进行分片,然后基于原始数据分片编码生成冗余数据,最后将原始数据和冗余数据分别写入不同的存储设备. 只要丢失的分片数不大于纠删码的容错度,原始数据便可根据未丢失的那些分片恢复出来,因此,使用纠删码的存储系统可以容忍一定数量的设备故障.

在大数据时代,由于庞大的数据规模,数据处理和数据存储不能得到平衡和充分发展,大量的数据迁移导致系统能耗急剧增加. 在冯·诺依曼架构中,存储与计算的分离造成了存储墙问题,导致现有的系统架构难以应对大数据和人工智能时代的数据爆炸. 数据移动成本的增加使传统数据计算模式受到冲击,研究者们开始尝试将计算单元移动到靠近存储器位置,即近数据处理技术[4]. 近数据处理的核心思想是利用存储器中的处理能力对数据进行简单计算处理后只传输数据处理结果到主机,节约大量的系统资源,具有低延时、高可靠性、高扩展性和低功耗等特点,有广阔的应用前景. 例如,云边端中的边缘计算,将计算任务从中心云下放到网络的边缘,即更靠近数据源或用户的地方. 边缘节点可利用近数据处理技术就近处理用户设备采集的数据,减少上传到数据中心的数据量.

纠删码技术和近数据处理技术是构建高效的云边端协同数据管理系统的两大基石,前者可有效降低存储开销,后者避免了大量的数据传输开销. 云边端协同的数据存储系统通常采用成熟的分布式存储系统作为底层存储引擎,然而主流的分布式存储系统(如Ceph[5]、DAOS[6]和MinIO[7])中的纠删码实现方式并不能高效地支持近数据处理. 以Ceph为例,底层纠删码机制会对写入的对象进行分片并编码,将数据分片和校验分片分布写入到不同的OSD(Object Storage Daemon),如图1所示. 这种方式会导致原本语义上相关联的数据被分散到不同的存储设备中,当应用层要求OSD对某个对象的数据进行分析处理时,该OSD仍需要从其他OSD读取并传输对象的分片数据,这违背了近数据处理的初衷.

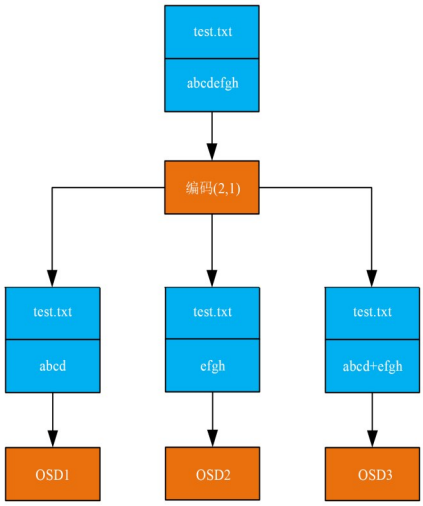


图1　Ceph纠删码编码过程

针对这个问题,本文提出了一种支持近数据处理的纠删码技术架构,并在分布式存储系统Ceph上进行实现,其基本原理如下:

(1)在OSD端以多个对象为一组对数据分布进行重新布局,保证属于同一对象的数据能够完整独立地存储在同一个存储设备上,将位于不同设备上的不同对象分片进行纠删编码.

(2)在Ceph客户端设计并实现了2种接口:put_more和get_shard,保证将多个对象同时写入纠删码存储池的同时,还能读出单个原始对象数据. 保存写入对象的元数据,对上层应用更加友好.

在真实部署的Ceph存储系统上对该方案进行典型场景的性能测试,实验结果表明在近数据处理和常规数据读取场景下,OSD读取对象性能分别提升了59.4%和10%,而对象写入性能与原版保持一致.

2 相关工作

本节对一些与本文主题相关的背景技术进行简要

介绍,主要涉及纠删码、分布式存储系统Ceph以及近数据处理技术的一些相关研究.

2.1 纠删码及其在Ceph中的应用

纠删码的研究起源于20世纪60年代,最早的是Reed和Solomon[8]提出的里得-所罗门(RS)码,这是一类功能强大的通用纠删码,具备强大的纠突发错误能力.1997年Plank等人[9]介绍了基于范德蒙德矩阵的RS码的软件实现,解释了RS码在存储系统中的编解码原理.1995年Blömer等人[10]提出了一种新的改进方案,即采用柯西矩阵取代范德蒙德矩阵作为编码矩阵,大大降低了编解码运算的时间复杂度.近年来比较有名的研究还有应用于微软Azure云存储的本地可修复码(Locally Repairable Codes,LRC)[11],双盘修复效率高的行对角校验码(Row-Diagonal Parity,RDP)码[12]等.对于存储应用而言,还有一项重要的研究是2007年Plank等人[13]给出的主流纠删码开源实现Jerasure库.该库是基于C语言编写的开源库,用于在分布式存储系统中提供数据的冗余和恢复.Jerasure库提供了一系列的函数,用于生成纠删码和重建数据,目前支持多种纠删码方案,例如RS码、柯西RS码、最小密度RAID-6码等.Jerasure库以插件形式集成到Ceph等大型分布式存储系统中,能够帮助Ceph开发者轻松实现纠删码的功能.

除此之外,近些年还出现了一些研究纠删码在Ceph中的应用.最著名的是Butterfly码[14]和Clay码[15],两者都在Ceph存储底层实现了集成应用.前者解决最小存储再生(Minimum Storage Regenerating,MSR)码在现代分布式存储系统中的集成及性能瓶颈问题,能够有效减少纠删码存储机制带来的大量存储开销.后者通过将多个常规的MDS码实例进行巧妙堆叠和关联,构造了一种新的最优修复码,并且在Ceph纠删码机制中进行修改以支持矢量编码.还有一些相关的研究,比如文献[16]分析现有LRC码的性能差异,并在Ceph进行评估实验,寻找不同场景下性能最好的方法.文献[17]提出一种分布式存储系统中的纠删码综合修复方案,在Ceph里对该方案进行实验评估,证明其能够有效减少修复开销.文献[18]考虑了一种新的具有纠删码的缓存框架,称为功能缓存.功能缓存涉及在缓存中使用编码块,使存储节点中的块和缓存组合形成的代码是最大距离可分离的纠删码.文献考虑了最佳的功能缓存放置和来自不同磁盘的文件请求的访问概率,所提出的算法在Ceph中实现,有显著的修复延迟改进.

2.2 近数据处理技术

近数据处理技术的核心思想是让计算向数据迁移,而非传统的让数据向计算迁移,减少系统中的数据传输开销.由于近数据处理技术的复杂性,编程框架一直是研究重点.2019年加州大学洛杉矶分校首次提出了一种比较完整的近数据处理计算框架INSIDER[19].在文件系统应用方面,2018年威斯康辛大学与华为技术有限公司提出的设备内文件系统DevFS[20],实现了近数据处理的初步需求.SparkNDP[21]构建了以Spark和HDFS[22]为基础的近数据处理框架.此框架在Spark数据源与HDFS之间引入了NDP代理,旨在增强数据处理能力.通过在这原始的数据读请求之上,运用REST API进行一层额外的封装,该框架能够传递下推的算子,实现更高效的数据处理.

Ceph在近数据处理方面主要是通过CLS(Clustered Logic Service)提供了基于插件的扩展机制,允许使用C++和Lua等语言编写自定义插件以扩展对象存储层.这些插件在Ceph OSD内以共享库的形式嵌入,提供在I/O路径中动态访问和操作RADOS对象的能力[23].Malacology[24]是一种基于Ceph CLS为实现近数据处理提供基础的架构,该架构旨在简化对现有存储系统抽象的底层软件栈的重用和扩展,更方便地创建新的应用服务.通过组合这些服务,可以更灵活地适应不断增长的数据处理系统和存储设备种类的需求.与此同时,针对对象存储系统,NearPM[25]介绍了针对存储类应用的近数据处理加速器,专门用于加速持久内存(Persistent Memory,PM)中的崩溃一致性机制.它提出了一种新的排序机制区分持久排序(Partitioned Persist Ordering,PPO),确保CPU和NDP设备之间以及多个NDP设备之间的正确持久排序.

3 技术背景

本节将针对所提出的支持近数据处理的纠删码技术架构所运用的关键技术进行详细的背景介绍,主要包括分布式存储系统Ceph的纠删码原生机制和对象存储架构RADOS.

3.1 Ceph纠删码原始数据布局分析

在分布式存储系统Ceph中,使用纠删码存储数据时,会将属于同一个对象的数据进行分散存储.以纠删码策略数据块数量k=3,校验块数量m=2为例.在一个对象数据写入底层之后,纠删码机制会将对象数据分为若干个大小相同的条带,数据不满1个条带则进行填充.对于每个条带的数据,会被分为3个数据块大小的数据和2个校验块大小的数据,其中块大小相同且固定不变.同一个条带的数据会分发到不同的存储设备(OSD)进行存储,如图2所示,这里假设每个对象数据恰好只占用1个条带.这样虽然降低了存储开销,但由于完整的一份数据被切分到多个存储设备,数据完整性被破坏,无法进行高效的近数据处理.任意一个OSD需要对某个对象进行处理时,都需要从其他OSD获取

对象分片才能获得完整的对象数据,这显然违背了近数据处理技术的初衷.

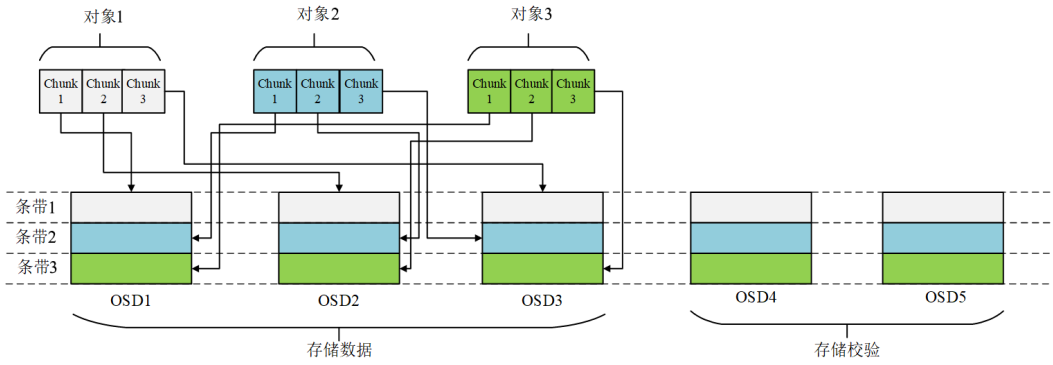


图2　Ceph纠删码原始数据布局

3.2　Ceph对象存储架构RADOS分析

Ceph的对象存储核心架构是RADOS,它完成了一个存储系统的核心功能,包括:Monitor模块为整个存储集群提供全局的配置和系统信息;通过CRUSH算法实现对象的寻址过程;完成对象的读写以及其他数据功能,提供了数据均衡功能、自动恢复功能、克隆和快照功能、对象分层存储的功能等[26]. RADOS作为Ceph对象存储的核心组件,在客户端提供与应用层交互以操作底层对象的接口,如:(a) get,输出对象内容到指定文件;(b) put,写入对象数据到存储底层;(c) create,在一个池中创建对象,池(pool)是存储对象的逻辑分区;(d) rm,删除池中对象.

目前RADOS在客户端提供的接口仅限于单个对象(可以看作1个文件)的操作,无法支持多个对象操作,而提出的支持近数据处理的纠删码架构涉及到多个对象的编码及数据布局,所以需要设计对应的接口对多个对象进行操作.

4　支持近数据处理的纠删码技术架构

本节对提出的支持近数据处理的纠删码技术架构进行阐述,主要包括支持近数据处理的纠删码数据布局方案设计以及RADOS对象处理接口设计(如图3所示).

4.1　支持近数据处理的纠删码数据布局

Ceph采用的原生编码方式并不能很好地适应近数据处理场景的需求,因此,本小节设计了一种新的纠删码数据布局方案,以避免近数据处理场景下的数据传输开销.

4.1.1　数据预处理方案设计

Ceph在底层执行纠删码存储策略时采用Primary-Replica模型,基本流程如下:客户端写入对象数据后,根据该对象数据计算得到存储节点(OSD)群,选择1个Primary OSD;Primary OSD对传入的数据进行编码切分等操作,将处理后的数据传入自身内部以及其他OSD进行存储.

Ceph执行纠删码存储策略主体是Primary OSD. 在本文设计的支持近数据处理的纠删码数据布局方案中,由于传入的数据是多个对象数据的集合,即一个数据量较大的对象单位,所以需要在Primary OSD处调用编码方法前在其内部I/O主路径对数据进行预处理. 预处理的主要作用是使写入的数据集合中的数据转换为键值存储的数据,保证每个编码的部分有对应的索引. 且保证需要编码的部分的数据长度与传入的对象数据长度对齐,不对齐的进行补零操作. 具体步骤如算法1

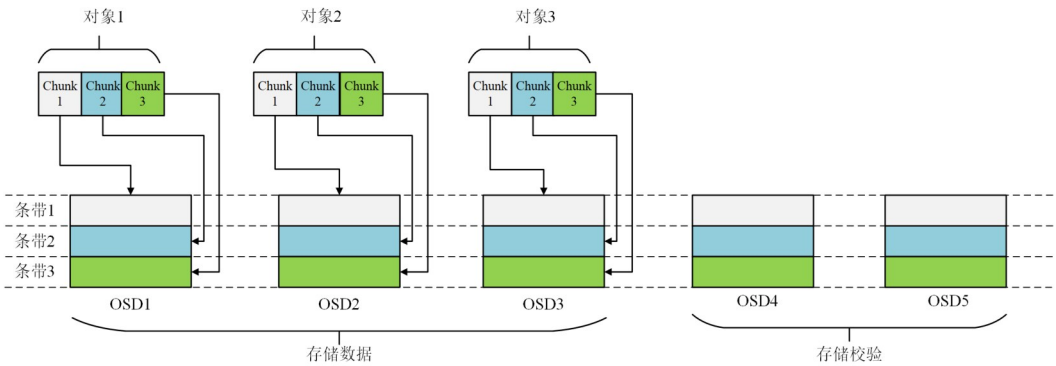


图3　支持近数据处理的纠删码数据布局

所示. 在Ceph客户端传入对象数据集合bl之后,获取纠删码策略所需要的编码块数量k,将bl的总长度按k取余后进行补零操作保证编码数据对齐. 对于数据集合bl,将其划分为k个部分,每部分数据长度为m. 每次按照m为单位调用函数substr对bl进行截断并写入Ceph内部定义的键值数据结构bufferlist类型的newbl中,最后将newbl的数据写入带索引的map类型的数据结构encode中,使写入的数据集合bl每部分的数据都有对应的编号.

算法1 数据预处理算法

输入:对象聚合集合bl

输出:预处理后的数据encode

1. 定义一个新的bufferlist类型的变量newbl
2. k ← get_data_chunk_count() //获取需要编码的数据块数量
3. length ← bl.length % k //按k取余
4. bl.append_zero(length) //补零操作
5. m ← bl.length / k
6. **FOR** each i∈[0, k−1] DO
7. newbl ← substr(bl, m) //每次按m为单位截断bl数据
8. encode[i] ← newbl
9. END FOR

4.1.2 纠删码数据布局方案设计

在上层对数据进行预处理之后,Ceph底层需要调用纠删码机制对处理后的数据进行编码(这一步仍然是在Primary OSD进行). 在调用编码算法比如RS码算法之前,为满足近数据处理的数据完整性需求,设计一个新的数据布局方案. 如图3所示,这里以纠删码存储策略数据块数量k=3,校验块数量m=2为例. 上层传入的实际是多个对象数据的集合,假设条带数量为3,本文设计的方案将单个对象数据按照条带的chunk_size分为k个部分,这里的k为3(假设单个数据对象的大小能被k个chunk_size所容纳). 本文仍然采用Ceph原生的机制,即以条带为单位进行编码. 但每次分别从3个不同的对象数据中取出3个大小相同的块填充条带,保证同一个对象的数据能存储到同一个存储设备中,这样在不破坏Ceph原生的编码机制下,能避免近数据处理场景下的跨OSD数据传输.

在PrimaryOSD的主I/O路径下,对应的算法的主要作用是对传入的对象数据计算其总长度,根据上一层计算的每部分长度按照Ceph条带固定的块大小chunk_size进行切分,写入到条带中进行编码,具体步骤如算法2所示. 首先,计算预处理的数据encode的总长度total_size,通过total_size除以条带宽度得到需要编码的条带数量;其次,遍历每个条带i,以i*chunk_size作为每次编码需要截段的起始位置偏移量offset;然后,在遍历条带的同时遍历encode的k个需要编码的部分,将每个编码部分的数据按照offset为起始位置、chunk_size为大小截段encode数据并写入数据缓存buf中;最后,重复上述过程,直到数据写满为止,调用编码函数进行编码.

算法2 数据布局算法

输入:已预处理的数据集合encode

输出:编码后的数据encoded

1. 定义新的bufferlist类型的变量encoded, buf, 临时缓存tmp
2. total_size ← encode.length //获取需要编码的总数据量
3. FOR each i∈[0, total_size / get_stripe_width()] DO
//遍历每个条带
4. offset ← i * get_chunk_size()
//以单个块大小作为每次截段数据的起始位置
5. FOR each j∈[encode.begin(), encode.end()] DO
//遍历k个编码部分
6. tmp ← j.substr(j.value, offset, get_chunk_size)
//每个编码部分按块大小截段
7. buf ← tmp
8. END FOR
9. encode(buf, encoded) //调用编码函数进行编码
10. END FOR
11. return encoded

4.2 RADOS对象处理接口设计

本文针对Ceph对象存储核心RADOS的客户端设计了2种接口,聚合对象接口put_more和对象部分读取接口get_shard. 本文所设计的支持近数据处理的纠删码技术总架构如图4所示. 该架构通过put_more上传多个对象,写入底层架构编码满足近数据处理条件,通过get_shard接口读取单个对象.

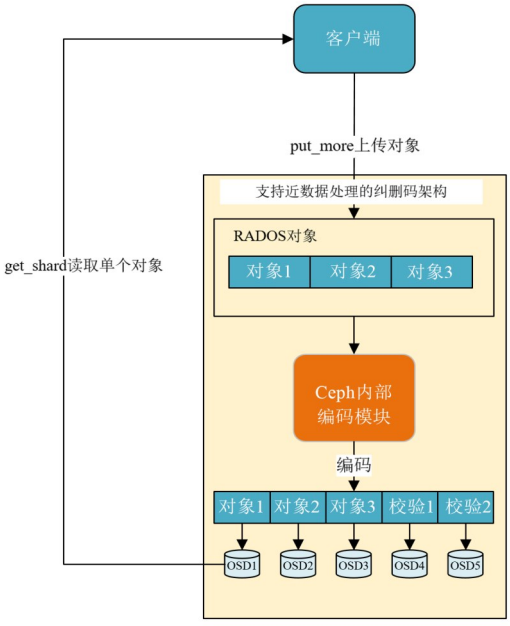


图4 支持近数据处理的纠删码总架构图

4.2.1 对象聚合接口设计

本文在Ceph对象存储核心RADOS写入对象时,新增了对象聚合接口put_more,用来将多个对象合并到一起写入底层.这样做的目的是RADOS不支持多对象写入,只有把多个对象聚合为一个大的对象之后才能调用Ceph底层的写数据接口.在该对象写入之后,本文设计的纠删码数据布局机制会将大对象拆分为原始的多个小对象,保证编码后1个完整的小对象对应存储到一个存储设备上,满足近数据处理场景的要求.

对象聚合算法主要是根据纠删码存储策略中定义好的数据块数量k,写入k个文件对象,聚合到一个大的缓存中传给客户端.客户端接收到应用层传入的数据集合后,直接调用RADOS默认写数据函数即可.具体步骤如算法3所示.应用层传入k个文件对象,依次判断每个文件的内容是否可读,将每个文件依次按照Ceph内部定义的固定对象大小op_size进行读取并拼接到缓存datas中,最后调用RADOS客户端的写数据接口write_full直接写入.

算法3　对象聚合算法

输入:k个文件对象,对象默认大小op_size
输出:对象聚合缓存datas

```
1.定义bufferlist类型变量outdata
2. FOR each i∈[0, k-1] DO //遍历每个文件
3.      fd ← open(file, O_RDONLY)
4.      IF 文件数据为空 THEN
5.          return "error reading input file"
6.      ELSE
7.    outdata.read_fd(fd, op_size)//按op_size为单位读取数据
8.       datas.append(outdata) //数据拼接
9.       END IF
10. END FOR
11. write_full(datas)
```

4.2.2 对象部分读接口设计

由于对象聚合接口写入的对象数据对RADOS底层存储系统而言是一个大的存储对象,所以想要单独读出其中的某部分,即写入的单个对象数据,需要与之匹配的对象部分读接口.本文设计了对象部分读接口get_shard,可以读出对象中指定长度的数据.

首先,需要做数据预处理,将应用层传入的需要读取的长度和读取数据的起始位置偏移量在RADOS客户端处理为Ceph底层能够处理的数据类型;然后,再调用Ceph底层RADOS读数据接口传入处理好的数据进行数据读取,具体步骤见算法4.应用层传入需要读取的对象名称objname、读取长度length和读取的起始位置offset,RADOS客户端将string类型的length和offset转换为long类型的数据,并调用read函数进行数据读取,返回读取结果.

算法4　对象部分读算法

输入:对象名称objname, 需要读取的长度length, 读取的起始位置offset
输出:读取结果ret

```
1.定义指针 ptr ← NULL
2. length ← convert_string_to_long(length, ptr) //数据类型转换
3. IF ptr == NULL || length < 0  THEN
4.          return "invalid value for length"
5.    END IF
6. ptr ← NULL
7. offset ← convert_string_to_long(offset, ptr) //数据类型转换
8. IF ptr == NULL || offset < 0 THEN
9.          return "invalid value for offset"
10. END IF
11.   ret ← read(objname, length, offset)
12. return ret
```

4.3 元数据管理机制设计

由于上文设计的部分读接口get_shard需要应用程序在客户端传入需要读取数据的大小及起始位置偏移量,对于上层应用程序并不友好,需要设计一种元数据管理机制为put_more接口聚合之前的小对象建立索引.本文设计的元数据管理机制主要利用Ceph内部定义的对象扩展属性xattr来实现.xattr以键值对形式存储在RADOS对象存储集群中,用来保存对象的元数据.本文设计的元数据管理方案如图5所示.在设置对象属性时需要一个RADOS对象的id,这里是指多个小对象聚合后的一个大对象.根据此对象的id,获取其每个小对象的编号以及大小,通过对象属性设置接口setxattr写入xattr中,完成元数据的保存.在读取数据时,运用对象属性获取接口getxattr即可读取元数据.

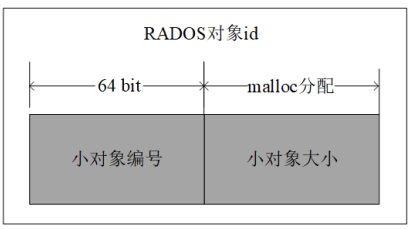


图5　元数据管理设计

元数据保存工作会作为一个子请求封装到写入请求中,再进入PrimaryOSD处理请求后,存储到Ceph内置的键值数据库中,完成对元数据的写入.在设计并实现元数据保存机制后,客户端的读写流程如下:(1)在聚合对象之前,记录每个小对象的编号以及大小,并写入对象扩展属性xattr;(2)使用对象聚合算法聚合小对

象，写入 put_more 处理函数中完成数据写入；(3)读取数据时，读取当前需要读取的小对象和聚合位置在前一位的小对象大小，分别作为需要读取对象的大小 length 和起始位置偏移量 offset；(4)对 length 和 offset 进行处理，调用 get_shard 完成数据读取.

5 性能评估

将支持近数据处理的纠删码数据布局方案及对应的调用接口实现集成到 Ceph 系统中，并在系统中进行对象读写功能的对比测试. 本节主要展示这些实验结果，给出简要分析.

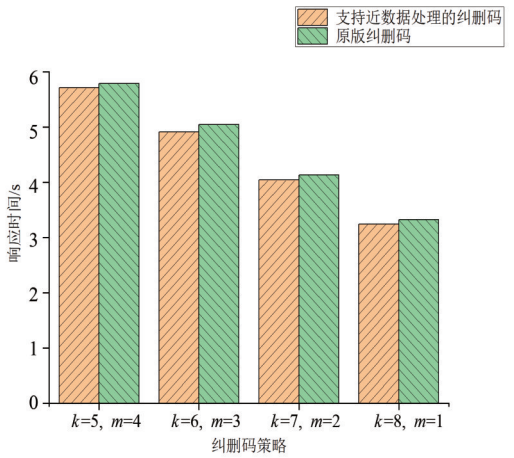
5.1 实验配置

为了对本文所提出的技术架构进行评估，使用了由 9 台 Sugon I620-G20 服务器组成的 Ceph 集群，每台服务器有 1 个专用于操作系统的 HDD，以及 1 个用来安装 Ceph OSD 的 500 GB SSD，这使得集群容量达到 4.5 TB. 每个服务器拥有千兆网络带宽以及 4 GB 的内存容量. 每台服务器都安装了 1 个 OSD 守护进程，为其中 1 台服务器安装了 Monitor 守护进程用来监视集群状态.

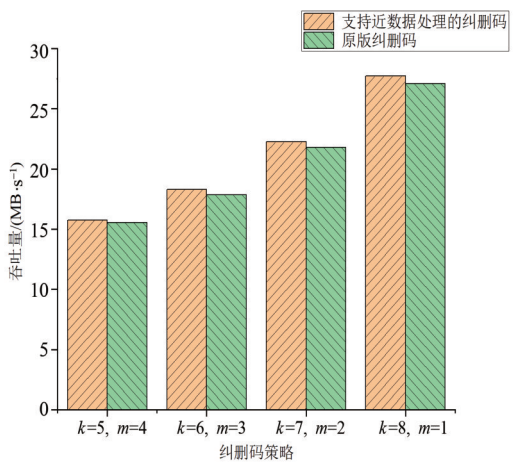
对于纠删码策略，选择最经典的 RS 码，数据块数量 k 的选择范围为 2~8，校验块数量 m 的选择范围为 1~4. 在 Ceph 系统中利用 RADOS 客户端创建存储池，分别对对象的写入以及对象的读取进行性能测试，并与 Ceph 原始的纠删码机制进行比较.

5.2 对象写入性能分析

本节使用编写的 shell 脚本进行对象写入性能的测试. 创建对应纠删码存储策略的存储池 pool，利用脚本通过 put 和 put_more 接口进行写入对象测试，保证每次写入的数据总量为 90 MB(RADOS 客户端限制的最大写入对象大小)，即传输开销也同为 90 MB. 测试结果如图 6 所示，对比了二者的响应时间及吞吐量. 由于是 9 个节点组成的集群，所以 $k+m$ 的总和为 9. 图 6 中可以看出，本文所提出的支持近数据处理的技术架构与 Ceph 原始版本的纠删码在对象写入性能上极为接近，这是因为二者只是对象数据在 OSD 上的布局不同，而写入的总数据量是一样的.



(a) 响应时间



(b) 吞吐量

图 6 对象写入性能

5.3 对象读取性能分析

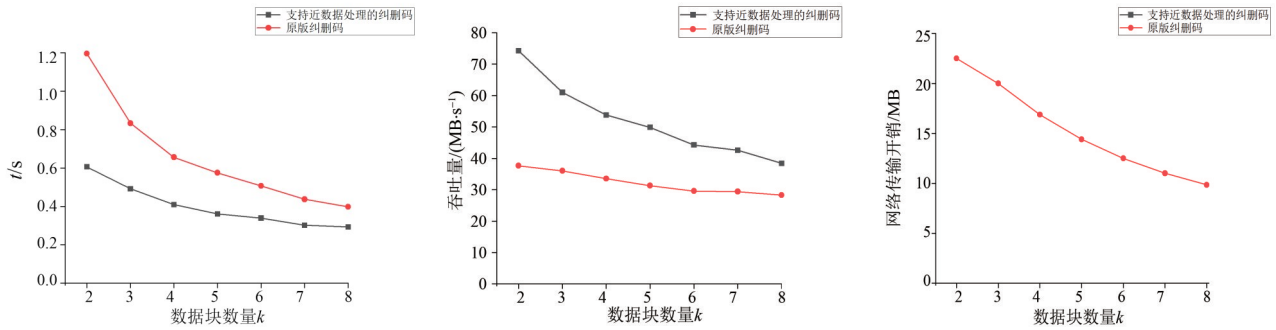
为了验证本文所提方案的效果，本节将对 2 种典型场景进行对象读取性能的测试. 具体的方法是通过 get 和 get_shard 接口进行读取对象，保证读取的数据总量为 90 MB，且每次读取的数据量相同，为 90/k MB. 由于 m 的数量不影响读取性能，所以都采用 k=8 且 m=1 的参数配置进行测试.

5.3.1 近数据处理场景下的对象读取性能

在近数据处理场景下，OSD 接收到来自应用的对某个对象的处理命令，然后读取对象进行处理并将结果返回. 在这种情况下对象并不需要传输给客户端，将 Ceph 客户端安装在主 OSD 上并调用上述接口模拟近数据处理场景下的对象读取操作，图 7 展示了此场景下本文所提方案与原版纠删码的性能对比. 图中对象读取吞吐量随 k 的增加而减少是因为写入的总数据量 90 MB 不变，每次选择 90/k MB 的数据量进行读取. 从图 7 中可以看出，本文设计的支持近数据处理的纠删码架构相比原始版本而言，读取对象的性能提升明显，约 59.4%。网络传输开销由于不需要传输数据到客户端，所以几乎为 0. 本文的架构保证了对象完整地存储到一个独立的 OSD 中，而原版 Ceph 在读取对象时需要从其他 OSD 收集对象分片. 图 7(b)中吞吐量随数据块数量 k 下降的原因是，读取的数据量下降幅度大于响应时间的下降幅度.

5.3.2 常规场景下的对象读取性能

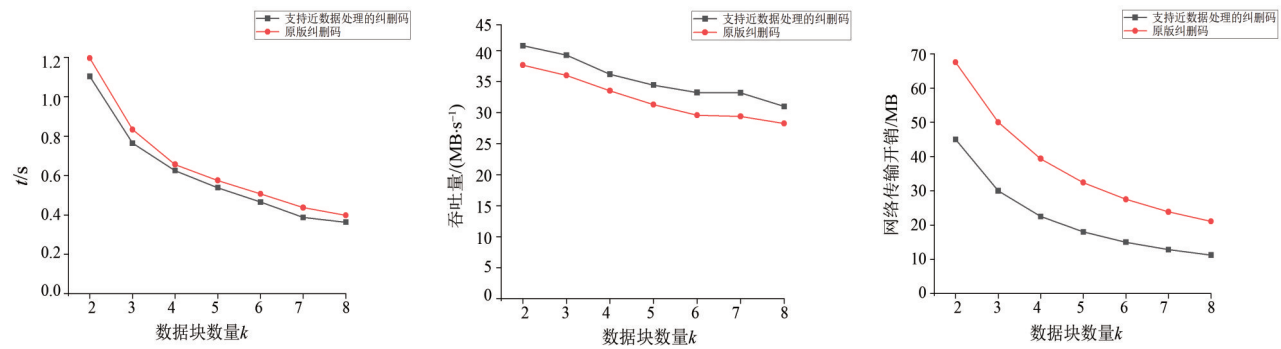
除了近数据处理的相关应用外，一般的读对象请求依然要从主 OSD 把对象数据传输给客户端.



(a) 响应时间　(b) 吞吐量　(c) 网络传输开销

图 7　近数据处理场景下的对象读取性能

本节测试常规场景下,即客户端和主 OSD 位于不同节点时读取对象的性能. 图 8 展示了本文方案与 Ceph 原版纠删码机制在这种情况下的性能对比. 可以看出,本文提出的支持近数据处理的纠删码架构相比原始版本读取对象的性能有 10% 的提升,网络传输开销平均约减少 42.7%. 本文的架构只需要将主 OSD 的对象传给客户端,而原版的主 OSD 在读取对象时仍需要从其他 OSD 收集对象分片.



(a) 响应时间　(b) 吞吐量　(c) 网络传输开销

图 8　常规场景下的对象读取性能

6　结 论

纠删码技术能够以较小的冗余开销保障系统的可用性,而近数据处理技术则可以减少大量的网络传输开销,这两者在云边端协同数据管理中均有应用价值. 然而,现有支持纠删码的主流分布式存储系统无法高效支持近数据处理技术,阻碍了两者在云边端协同数据管理中的结合使用. 本文设计并实现了一种支持近数据处理的纠删码存储架构,包括新的纠删码数据布局机制以及支持该架构的对象处理接口,避免了近数据处理场景下的跨节点数据传输. 实验结果表明,本文所提出的架构在保持数据写入性能不变的前提下,在近数据处理场景和常规数据访问场景下的数据读取性能分别有 59.4% 和 10% 的提升. 研究结果将有助于构建高效可靠的云边端协同数据管理系统.

参考文献

[1] LIU K Y, PENG J, WANG J R, et al. Adaptive and scalable caching with erasure codes in distributed cloud-edge storage systems[J]. IEEE Transactions on Cloud Computing, 2023, 11(2): 1840-1853.

[2] JIN H, LUO R K, HE Q, et al. Cost-effective data placement in edge storage systems with erasure code[J]. IEEE Transactions on Services Computing, 2023, 16(2): 1039-1050.

[3] YANG J, SABINS A, S D Bergeret al. C2DN: How to harness erasure codes at the edge for efficient content delivery[C]//USENIX Symposium on Networked Systems Design and Implementation. Washington: USENIX Association, 2022: 1159-1177.

[4] GAO M Y, AYERS G, KOZYRAKIS C. Practical near-da-

ta processing for In-memory analytics frameworks[C]// 2015 International Conference on Parallel Architecture and Compilation (PACT). Piscataway: IEEE, 2015: 113-124.

[5] WEIL S A, BRANDT S A, MILLER E L, et al. Ceph: A scalable, high-performance distributed file system[C]//Operating Systems Design and Implementation. Seattle: USENIX Association, 2006: 307-320.

[6] LIANG Z, LOMBARDI J, CHAARAWI M, et al. DAOS: A scale-out high performance storage stack for storage class memory[M]//Lecture Notes in Computer Science. Cham: Springer International Publishing, 2020: 40-54.

[7] ADAMS I F, AGRAWAL N, MESNIER M P. Enabling near-data processing in distributed object storage systems[C]//Proceedings of the 13th ACM Workshop on Hot Topics in Storage and File Systems. New York: ACM, 2021: 28-34.

[8] 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.

[9] PLANK J S. A tutorial on Reed-Solomon coding for fault-tolerance in RAID-like systems[J]. Software: Practice and Experience, 1997, 27(9): 995-1012.

[10] BLÖMER J, KALFANE M, KARP R M, et al. An XOR-Based Erasure-Resilient Coding Scheme[R]. Berkeley: International Computer Science Institute, 1995.

[11] HUANG C, SIMITCI H, XU Y K, et al. Erasure coding in windows Azure storage[C]//2012 USENIX Annual Technical Conference. Boston: USENIX Association, 2012:15-26.

[12] CORBETT P, ENGLISH B, GOEL A, et al. Row-diagonal parity for double disk failure correction[C]// 3rd USENIX Conference on File and Storage. Berkeley: USENIX Association, 2004: 1-14.

[13] PLANK J S, SIMMERMAN S, SCHUMAN C D. Jerasure: A Library in C/C++ Facilitating Erasure Coding for Storage Applications[M]. Knoxville: University of Tennessee, 2008.

[14] PAMIES-JUAREZ L, BLAGOJEVIC F, MATEESCU R, et al. Opening the chrysalis: On the real repair performance of MSR codes[C]//14th USENIX Conference on File and Storage Technologies. Santa Clara: USENIX Association, 2016: 81-94。

[15] VAJHA M, RAMKUMAR V PURANIK B, et al. Clay codes: Moulding MDS codes to yield an MSR code[C]// 16th USENIX Conference on File and Storage Technologies. Oakland: USENIX Association, 2018: 139-154.

[16] KOLOSOV O, YADGAR G, LIRAM M, et al. On fault tolerance, locality, and optimality in locally repairable codes[J]. ACM Transactions on Storage, 2020, 16(2): 1-32.

[17] CHEN J M, LI Z P, FANG G, et al. A comprehensive repair scheme for distributed storage systems[J]. Computer Networks, 2023, 235: 109954.

[18] AGGARWAL V, CHEN Y F R, LAN T, et al. Sprout: A functional caching approach to minimize service latency in erasure-coded storage[C]//2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS). Piscataway: IEEE, 2016: 753-754.

[19] RUAN Z Y, HE T, CONG J. INSIDER: Designing in-storage computing system for emerging high-performance drive[C]//USENIX Annual Technical Conference. Washington: USENIX Association, 2019: 379-394.

[20] KANNAN S, ARPACI-DUSSEAU A C, ARPACI-DUSSEAU R H, et al. Designing a true direct-access file system with DevFS[C]//16th USENIX Conference on File and Storage Technologies. Oakland: USENIX Association, 2018: 241-256.

[21] RACHURI S P, GANTASALA A, EMANUEL P, et al. Optimizing near-data processing for spark[C]//2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS). Piscataway: IEEE, 2022: 636-646.

[22] SHVACHKO K, KUANG H R, RADIA S, et al. The hadoop distributed file system[C]//2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST). Piscataway: IEEE, 2010: 1-10.

[23] CHAKRABORTY J, JIMENEZ I, RODRIGUEZ S A, et al. Skyhook: towards an arrow-native storage system[C]// 2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid). Piscataway: IEEE, 2022: 81-88.

[24] SEVILLA M A, WATKINS N, JIMENEZ I, et al. Malacology: A programmable storage system[C]//Proceedings of the Twelfth European Conference on Computer Systems. New York: ACM, 2017: 175-190.

[25] SENEVIRATNE Y, SEEMAKHUPT K, LIU S H, et al. NearPM: A near-data processing system for storage-class applications[C]//Proceedings of the Eighteenth. New York: ACM, 2023: 751-767.

[26] JOHANES J, JOHARI M F, KHALID M, et al. Comparison of various virtual machine disk images performance on glusterFS and ceph rados block devices[C]//3th International Conference on Informatics & Applications. Malaysia: SDIWC, 2014: 1-7.

作者简介

李浩然　男,2000年6月出生,云南省昭通人. 西北工业大学计算机学院硕士研究生. 主要研究方向为纠删码和分布式存储.
E-mail: 1148436381@mail.nwpu.edu.cn

黄志杰　男,1983年11月出生,福建省泉州人. 西北工业大学计算机学院副教授,硕士生导师. 主要研究方向为编码理论、存储系统、区块链等.
E-mail: jayzy.huang@nwpu.edu.cn

史宇龙　男,2000年10月出生,陕西省咸阳人. 西北工业大学计算机学院硕士研究生. 主要研究方向为纠删码、区块链存储.
E-mail: yulongshi@mail.nwpu.edu.cn

赵承佳　男,2001年4月出生,贵州省遵义人. 西北工业大学计算机学院硕士研究生. 主要研究方向为纠删码技术和分布式存储系统.
E-mail: 1034886053@qq.com

赵楠楠　女,1987年10月出生,山东省肥城人. 西北工业大学计算机学院副教授,硕士生导师. 研究方向为云存储、分布式系统、容器虚拟化等.
E-mail: nannanzhao@nwpu.edu.cn

张　晓　男,1978年2月出生,河南省新乡人. 西北工业大学计算机学院教授,博士生导师. 研究方向为分布式存储系统、云计算与云存储系统、系统评测与仿真等.
E-mail: zhangxiao@nwpu.edu.cn