计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.201000149

Ceph 分布式存储系统性能优化技术研究综述

张 晓[1,2,3] 张思蒙[1,2] 石 佳[1,2] 董 聪[1,2] 李战怀[1,2,3]
1 西北工业大学计算机学院 西安 710129
2 西北工业大学大数据存储与管理工业和信息化部重点实验室 西安 710129
3 西北工业大学空天地海一体化大数据应用技术国家工程实验室 西安 710129

摘 要 Ceph是一个统一的分布式存储系统,可同时提供块、文件和对象3种接口的存储服务。与传统的分布式存储系统不同,它采用了无中心节点的元数据管理方式,因此具有良好的扩展性和线性增长的性能。经过十余年的发展,Ceph已被广泛地应用于云计算和大数据存储系统。作为云计算的底层平台,Ceph除了提供虚拟机的存储服务外,还可以直接提供对象存储服务和NAS文件服务。Ceph支撑着云计算系统中多种操作系统和应用的存储需求,它的性能对其上的虚拟机和应用有较大的影响,因此Ceph存储系统的性能优化一直是学术界和工业界的研究热点。文中首先介绍了Ceph的架构和特性;然后针对现有的性能优化技术,从对内部机制进行改进、面向新型硬件和基于应用的优化这3个方面进行了归纳和总结,综述了近年来Ceph存储和优化的相关研究;最后对该领域未来的工作进行了展望,以期为分布式存储系统性能优化的研究者提供有价值的参考。

关键词:Ceph分布式存储系统;性能优化;非易失内存;固态硬盘;统一存储
中图法分类号 TP319

Review on Performance Optimization of Ceph Distributed Storage System

ZHANG Xiao[1,2,3],ZHANG Si-meng[1,2],SHI Jia[1,2],DONG Cong[1,2] and LI Zhan-huai[1,2,3]
1 School of Computer Science,Northwestern Polytechnical University,Xi'an 710129,China
2 MIIT Key Laboratory of Big Data Storage and Management,Northwestern Polytechnical University,Xi'an 710129,China
3 National Engineering Laboratory for Integrated Aero-Space-Ground-Ocean Big Data Application Technology,Northwestern Polytechnical University,Xi'an 710129,China

Abstract Ceph is a unified distributed storage system,which can provide storage services of 3 types of interfaces:block,file and object. Different from the traditional distributed storage system,it adopts the metadata management method without central node,so it has good scalability and linear growth performance. After more than ten years of development,Ceph has been widely used in cloud computing and big data storage systems. As the underlying platform of cloud computing,Ceph not only provides storage service for virtual machines,but also directly provides the object storage service and NAS file service. Ceph supports storage requirements of various operating systems and applications in cloud computing systems. Its performance has a great influence on virtual machines and applications running on it. Therefore,the performance optimization of the Ceph storage system has been a research hotspot in academia and industry. This paper first introduces the architecture and characteristics of Ceph,then summarizes existing performance optimization technologies from 3 aspects,including internal mechanism improvement,new hardware-oriented and application-based optimization and reviews the recent research on Ceph storage and optimization. Finally,it prospects the future work,hoping to provide a valuable reference for researchers in the performance optimization of distributed storage system.

Keywords Ceph distributed storage system,Performance optimization,Non-volatile memory,Solid state disk,Unified storage

随着物联网、大数据和云计算等技术的发展,大量的数据需要被采集、存储和高速处理。分布式存储系统是大数据处理和云计算系统中的重要组成部分,它的性能对上层的数据处理和虚拟化应用等都有很大的影响。传统的分布式存储系统将数据分布在多个存储节点上,并将数据和存储节点的对应关系保存在少数元数据服务器上。客户端通过访问元数据服务器获得数据与存储节点的对应关系后,直接访问存储节点完成读写操作。虽然客户端和存储节点间采用多对多直接

到稿日期:2020-10-16 返修日期:2020-11-26 本文已加入开放科学计划(OSID),请扫描上方二维码获取补充信息。
基金项目:国家重点研发计划(2018YFB1004401);北京市自然科学基金-海淀原始创新联合基金(L192027)
This work was supported by the National Key Research and Development Program(2018YFB1004401) and Beijing Natural Science Foundation(L192027).
通信作者:张晓(zhangxiao@nwpu.edu.cn)

访问的方式可以提供高并发性能，但是元数据的查询与处理成为了限制该性能的瓶颈。同时，全局一致的元数据服务也限制了分布式存储系统的扩展性和可用性。针对这个问题，Weil 提出了 CRUSH(Controlled Scalable Decentralized Placement of Replicated Data)算法，将数据与节点的映射关系通过哈希函数来表示，避免了大量并发地访问元数据服务器的过程。在此基础上设计的 Ceph 分布式文件系统得到了学术界和工业界的广泛支持。

Ceph 具有高可扩展性和高可用性，也提供了丰富的访问接口，因此被广泛应用于大数据处理、高性能计算和云计算场景中。但是，Ceph 内部机制复杂，过多的抽象和封装导致其早期版本的软件开销很大。在过去的十余年中，学术界、工业界和开源社区持续致力于优化其性能，如何充分利用集群的存储和网络性能一直是研究的重点。本文对有关 Ceph 性能优化的研究成果进行了整理和分析，并展望了未来的研究方向。

本文第 1 节介绍了 Ceph 的发展历史、系统架构及核心算法，总结了 Ceph 与其他分布式文件系统的特点和面临的挑战，并给出了本文的研究框架；第 2 节对通用场景的存储后端、数据分发、网络通信等优化技术进行了分析；第 3 节从新型硬件和网络方面，对非易失内存、混合存储及网络传输的优化技术进行了研究；第 4 节从应用的角度，总结了针对不同应用场景的存储优化方法；第 5 节指出 Ceph 存储系统当前面临的挑战，并对未来的研究方向做出了展望。

1 Ceph 分布式存储系统的概述

本节首先对 Ceph 分布式存储系统的发展历程进行了介绍；然后概括了 Ceph 系统架构和关键技术；最后对 Ceph 存储特点和挑战进行了分析，并给出了本文的研究框架，即从内部机制、特定硬件和应用场景这 3 个方面对 Ceph 相关的存储优化技术进行了研究和综述。

1.1 Ceph 存储系统的发展历程

Ceph 项目是加州大学圣克鲁兹(Santa Cruz)分校的 Weil 于 2006 年开发的。当时他发现元数据的查询和维护严重影响了 Lustre 等分布式文件系统的性能和扩展性，因此设计了一种利用算法来确定数据与存储节点对应关系的方法 CRUSH[1]。然后，在此基础上设计并实现了 Ceph 分布式系统的原型，其初始代码有约 4 万行。在 2006 年的 OSDI 学术会议上，Weil 发表了介绍 Ceph 的论文，并在论文中提供了下载链接[2]。2010 年 5 月发布的 Linux 内核 2.6.34 已开始支持 Ceph。Weil 也成立了 IntTank 公司，专注于 Ceph 的开发。2014 年 5 月，该公司被 Red Hat 收购。Ceph 同时支持 3 种存储访问接口，因此被广泛应用于开源私有云计算平台中，为云计算平台提供虚拟机存储和对象访问能力。开源私有云平台 OpenStack 于 2016 年的调查显示，在其部署的生产系统中，近 60%的系统使用了 Ceph 提供的块存储服务，并且该数据已经连续 5 年持续增长[3]。

Ceph 开源后，大量公司和开发者投入精力去开发其新特性和功能。表 1 列出了自开源以来 Ceph 的重要版本更新，它以字母为顺序对重要的发行版本排序。第一个 Ceph 的版本是于 2008 年发布的 0.1 版。直到 2015 年 4 月发布了 0.94.1 版后，社区才采用了新的版本策略。内部版本 x.0.z 表示开发版本，x.1.z 表示内测版，x.2.z 表示发布版本。Ceph 的发布版本一般有两年左右的维护期，目前仍在维护期的版本是 mimic 之后的 3 个版本。

表 1 Ceph 的重要版本更新

Table 1 Descriptions of important versions of Ceph

Num	Name	Version	Initial release	Content
1	Argonaut	0.48	2012.06	osd：能力模型简化
2	Bobtail	0.56	2013.01	支持自动格式化和安装 XFS 和 ext4(除 btrfs 之外)
3	Cuttlefish	0.61	2013.05	添加“noscrub”“nodeepscrub”OSD map 标志；pg 拆分的杂项修复
4	Dumpling	0.67	2013.08	radosgw 的多站点支持；librados 中的对象命名空间
5	Emperor	0.72	2013.11	radosgw 的多数据中心复制、可改进性，以及大量的增量性能和内部分解工作
6	Firefly	0.80	2014.05	支持擦除编码和缓存分层
7	Giant	0.87	2014.10	对 theOSD 和客户端 librados 代码进行了一系列改进；OSD 支持擦除编码方案
8	Hammer	0.94	2015.04	CRUSH 放置算法的改进(straw2 bucket 类型)；使用 SSD 进行缓存处理(读取支持)
9	Infermalis	9.0	2015.11	RADOS I/O 提示(EC 写作性能提高 35%)；为系统日志调用分配 LOG_DEBUG 优先级
10	Jewel	10.0	2016.04	引入 ceph-mgr；RGW 引入了同步模块
11	Kraken	11.0	2017.10	确定了 BlueStore 磁盘格式，对其进行了压力测试；更好地支持纠删码
12	Luminous	12.2.13	2017.10	默认的消息处理从 SimpleMessenger 变成了 AsyncMessenger；引入 bluestore；内置 Dashboard 预览
13	mimic	13.2.8	2018.05	Luminous 中引入的(只读)Ceph 管理器仪表板由 openATTIC Ceph 新管理工具所取代
14	Nautilus	14.2.8	2019.03	支持多个 users/roles；RBD 可以实时迁移镜像
15	Octopus	15.2.0	2020.03	新增部署工具 cephadm；可临时或永久禁用心跳警告；BlueStore 性能优化

1.2 Ceph 的系统架构和关键技术

Ceph 存储系统的设计目标是提供高性能、高可扩展性、高可用的分布式存储服务。它采用 RADOS(Reliable Autonomic Distributed Object Store)在动态变化和异构的存储设备集群上，提供了一种稳定、可扩展、高性能的单一逻辑对象存储接口和能够实现节点自适应和自管理的存储系统[2]。数据的放置采取 CRUSH 算法，客户端根据算法确定对象的位置并直接访问存储节点，不需要访问元数据服务器[1]。CRUSH 算法具有更好的扩展性和性能。本节介绍了 Ceph 的集群架构、数据放置方法以及数据读写路径，并在此基础上分析其性能特点和瓶颈。

1.2.1 集群架构

RADOS 可提供高可靠、高性能和全分布式的对象存储服务。对象的分布可以基于集群中各节点的实时状态，也可

以自定义故障域来调整数据分布。块设备和文件都被抽象包装为对象，对象则是兼具安全和强一致性语义的抽象数据类型，因此 RADOS 可在大规模异构存储集群中实现动态数据与负载均衡。

对象存储设备（Object Storage Device，OSD）是 RADOS 集群的基本存储单元，它的主要功能是存储、备份、恢复数据，并与其他 OSD 之间进行负载均衡和心跳检查等。一块硬盘通常对应一个 OSD，由 OSD 对硬盘存储进行管理，但有时一个分区也可以成为一个 OSD，每个 OSD 皆可提供完备和具有强一致性语义的本地对象存储服务。MDS（Metadata server）是元数据服务器，向外提供 CephFS 在服务时发出的处理元数据的请求，将客户端对文件的请求转换为对对象的请求。RADOS 中可以有多个 MDS 分担元数据查询的工作。

如图 1 所示，一个 RADOS 集群由大量 OSD、0～n 个 MDS 和少数几个 Monitor 组成。

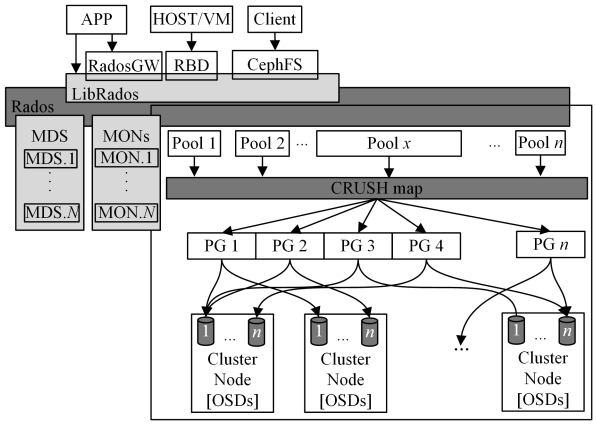


图 1　Ceph 系统的架构

Fig. 1　Architecture of Ceph system

1.2.2　数据放置算法

RADOS 取得高可扩展性的关键在于彻底抛弃了传统存储系统中的中心元数据节点，另辟蹊径地以基于可扩展哈希的受控副本分布算法——CRUSH 来代替。通过 CRUSH 算法，客户端可以计算出所要访问的对象所在的 OSD。与以往的方法相比，CRUSH 的数据管理机制更好，它把工作分配给集群内的所有客户端和 OSD 来处理，因此具有极大的伸缩性。CRUSH 用智能数据复制来确保弹性，更能适应超大规模存储。如图 2 所示，从文件到对象以及 PG（Placement Group）都是逻辑上的映射，从 PG 到 OSD 的映射采用 CRUSH 算法，以保证在增删集群节点时能找到对应的数据位置。

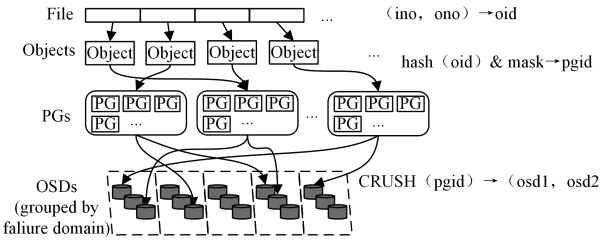


图 2　CRUSH 算法选择数据存放位置

Fig. 2　CRUSH algorithm selects data storage location

根据 Weil 的研究，CRUSH 算法具有相当好的可扩展性，在数千个 OSD 的情况下仍然能保证良好的负载平衡[1]。但这更多的是理论层面上，目前还没有研究人员给出在数 PB 规模的生产环境中的测试结果。

CRUSH 算法是 Ceph 最初的两大创新之一，也是整个 RADOS 的基石。CRUSH 在一致性哈希的基础上很好地考虑了容灾域的隔离，能够实现各类负载的副本放置规则，例如跨机房、机架感知等。同时，CRUSH 算法支持多副本和纠删码这两种数据冗余方式，还提供了 4 种不同类型的 Bucket（Uniform，List，Tree，Straw），充分考虑了实际生产过程中硬件的迭代式部署方式。

CRUSH 虽然提供了快速数据定位方法，但也有一定的缺陷。首先，在选择 OSD 时会出现权重失衡的情况，即低权重的 OSD 虽然在实际中可用，但是与其他复制节点相差较大且需二次哈希；其次，在增删 OSD 时会有数据的额外迁移；最后，完全依赖哈希的随机性可能会导致 OSD 的容量使用率不均衡，在实际环境中出现过超过 40％的差异。因此，从 2017 年发布的 Luminous 版本起，Ceph 提供了被称为 upmap 的新机制，用于手动指定 PG 的分布位置，来达到均衡数据的效果。

综上，CRUSH 算法仍然是目前经过实践检验的最好的数据分布算法之一。

1.2.3　统一访问接口

如图 3 所示，RADOS 提供了分布式对象存储能力，并在此基础上扩展了块存储和文件存储功能。RADOS 中单个对象的大小根据配置文件指定，一般为 4 M。LIBRADOS 提供的库可以访问任意对象的内容。RGW（RADOS Gateway）提供了一个基于 Bucket 的对象存储服务。RGW 提供的服务兼容 AWS（Amazon Web Services）的 S3 以及 Openstack 的 Swift。RGW 的对象可大于 4 M，当其大小超过 4 M 时，RGW 会将该对象分为首对象和多个数据对象。

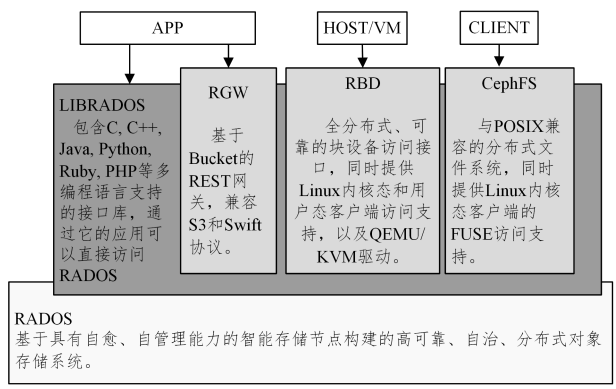


图 3　基于 RADOS 的多种访问接口

Fig. 3　Multiple interfaces based on RADOS

块存储接口提供类似磁盘的连续字节序列的存储能力，这是存储数据应用最广泛的形式。磁盘阵列、存储区域网络、iSCSI 都可以提供块存储功能。Ceph 的块存储利用 RADOS 功能，支持复制、快照、一致性和高可用等特性。块设备是精简配置的，并且可调整大小。Ceph 的 RBD（Rados Block Device）可使用内核模块或 librbd 与 OSD 进行交互。RBD 设备

默认在名为 rbd 的资源池中。每个 rbd 在创建后会生成一个名为 rbdName.rbd 的对象，该对象内保存了这个块设备对应的所有对象的前缀，一般前缀的形式为 rb.0.1014.uniname.seqno，其中 uniname 是 8 位十六进制数，seqno 是该对象对应地址的顺序号。

文件系统接口是通过 CephFS 实现的。CephFS 文件系统中的数据(文件内容)和元数据(目录、文件)都以对象的形式被保存在 OSD 上。客户端可以使用 ceph-fuse 挂载为应用层模式，或者使用内核的 mount-ceph 挂载为内核模式。两种模式都是与 MDS 通信，以获得文件系统的目录结构信息，并访问对应的 OSD。CephFS 的元数据也被保存在对象中，其对象的前缀为 msd_，被保存的内容包括文件系统的索引节点、元数据日志和快照等。MDS 在启动时会读取文件系统元数据对象并将其缓存在内存中，客户端需要与之通信以查询或更新元数据信息。

1.2.4 性能测试与监控方法

Ceph 支持多种存储访问接口，现有的多种性能测试工具都可用于 Ceph 的性能测试，如测试块接口性能的 fio，iometer 等；测试 CephFS 接口的 filebench，fio 等；测试对象接口的 cosbench 等。Ceph 有专用的基准测试集 CBT[4]，其包含 radosbench，librbdfio，kvmrbdfio 和 rbdfio。radosbench 基准测试使用 ceph-common 软件包附带的 rados 二进制文件，通过对象接口来访问 Ceph 集群。剩下的 3 个工具都是测试块存储性能的。librbdfio 基准模块通过用户态 librbd 库来测试 RBD 的块存储性能。kvmrbdfio 基准测试要求在使用 CBT 之前创建虚拟机实例，并挂载 RBD 块设备。rbdfio 基准测试使用内核进行驱动并将其映射到块设备的 RBD 块设备上。Teuthology 是一个 Ceph 自动化测试的框架，可以在指定节点运行测试用例，也可以用于性能的测试[5]。

对 Ceph 系统进行持续的性能监控可以了解集群运行状况，及早发现性能瓶颈。Ceph 提供了命令行接口输出性能相关的统计数据。OSD 以 PG 为单位收集性能数据并定期发给 Monitor 节点。Monitor 节点汇总性能数据并同步至其他 Monitor 节点。我们也提出了一种针对 Ceph 存储系统层进行分层性能监测和采集的框架[6]，以及一种通过 Ceph 电文来分析系统性能和瓶颈的方法[7]。

1.3 Ceph 存储系统的特点和挑战

本节总结了 Ceph 存储系统的特点和面临的挑战。

(1)Ceph 存储系统的优点

1)高性能。针对并发量大的异步 IO 场景，随着集群规模的扩大，Ceph 可提供近线性的性能增长。

2)高可扩展性。Ceph 通过 CRUSH 算法来实现数据寻址。这种方法避免了元数据访问的瓶颈，使集群的存储容量可以轻易扩展至 PB 级，甚至 EB 级。

3)统一存储，适用范围广。Ceph 支持块、文件和对象存储，可满足多种不同的需求。底层的 RADOS 可扩展并支持不同类型的存储服务。

4)支持范围广。自 2012 年起，Linux 内核开始支持 Ceph，目前 Ceph 可以在几乎所有主流的 Linux 发行版和其他类 UNIX 系统上运行。自 2016 年起，Ceph 开始支持 ARM 架构，同时也可适用于移动、低功耗等领域，其应用场景覆盖了当前主流的软硬件平台。

(2)Ceph 面临的挑战

1)Ceph 底层采用定长的对象存储，为了保证对象级别的原子性，底层存储引擎的写放大问题严重影响了性能。

2)Ceph 的数据分布算法 CRUSH 在实际环境中存在一些问题，包括扩容时数据迁移不可控、数据分布不均衡等。这些问题影响了 Ceph 性能的稳定性。

3)Ceph 对新型存储介质的支持较差。在使用高速存储介质时，软件造成的时延比硬件导致的时延高出数十倍。社区也在开发面向新型存储介质的存储引擎。

4)Ceph 的架构复杂，抽象层次多，时延较大。虽然 Ceph 采用面向对象的设计思想，但其代码内对象间的耦合严重，导致不同版本间的接口不兼容。针对不同版本的性能优化技术和方法也互相不兼容。

5)Ceph 是一个通用的分布式存储系统，可应用于云计算、大数据和高性能计算等领域。针对不同的访问负载特征，Ceph 还有较大的性能提升和优化空间。

1.4 研究框架

由上述分析可知，Ceph 在扩展性、高可用和可移植性等方面已经比较成熟。Ceph 本身的抽象层次较多，可适配不同的存储介质及混合存储介质。它提供了通用的数据存储服务，可以在不同的场合使用。当前对 Ceph 性能优化研究的重点是如何在保证高可用和扩展性的前提下，针对不同硬件和应用场景来进行优化，提高数据并发访问性能，并降低访问时延。

作为一个通用的分布式存储系统，Ceph 的优化主要集中在提高数据的持久化速度和提高副本的传输效率这两方面。同时，在特定的使用场景下，针对不同的硬件特性和访问特征，也有研究者对 Ceph 进行了性能优化相关的研究。本文从内部机制、特定硬件和应用场景这 3 个方面对 Ceph 相关的存储性能优化技术进行了总结。

2 Ceph 内部机制的优化

Ceph 是一个通用的分布式文件系统，适用于不同的场景。内部机制的优化对所有的场景都会产生性能的提升，但是优化的难度和复杂度也最高。

2.1 存储引擎的优化

在分布式存储系统中，数据被分散在大量的存储服务器上，大部分分布式存储系统都直接使用本地文件系统来存储数据，如 HDFS(Hadoop Distributed File System)，Lustre 等。高性能、高可靠的分布式存储系统离不开高效、一致、稳定、可靠的本地文件系统。本地文件系统的代码已经过长时间的测试和性能优化，对于数据持久化和空间管理也有相应的方案。文件系统提供了 POSIX(Portable Operating System Interface of UNIX)接口，通过这个接口，分布式文件系统可以切换不同的本地文件系统。

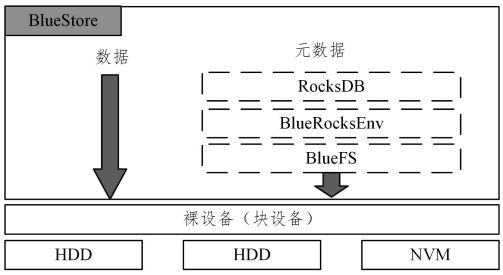
Ceph 早期的版本采用将对象存储在本地文件系统的存

储后端的方式，该方式被称为 FileStore。FileStore 通过 POSIX 接口将对象和对象属性放在本地文件系统上，如 XFS，ext4，btrfs 等。最初，对象属性被存储在 POSIX 扩展文件属性（xattrs）中，但当后来对象属性超出 xattrs 的大小或计数限制时，FileStore 就在 LevelDB 中存放对象属性。

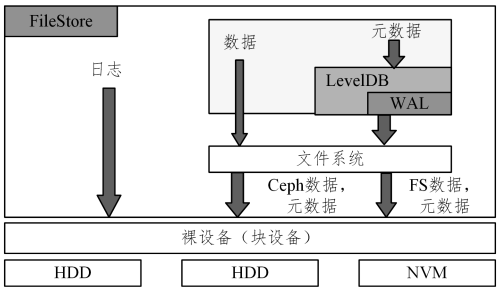
本地文件系统不能很好地适配 Ceph 的对象存储需求的原因主要包括以下几个方面。1）数据和元数据分离不彻底，导致对象寻址慢。FileStore 根据对象前缀将其放置在不同目录下，对象在进行访问时需要多次寻址，且同一目录下的文件也没有排序。2）本地文件系统不支持对象的事务操作。FileStore 为了支持写事务的特性，通过写前日志功能来保证事务的原子性。这导致了数据“双写”的问题，造成了一半磁盘性能的浪费。3）本地文件系统也有日志等保证一致性的操作，这进一步导致了写放大。

使用本地文件系统作为后端存储的弊端导致了 FileStore 的性能很差。我们测试的结果表明，在三副本条件下的块存储服务写性能甚至达不到硬盘自身性能的 1/3[8]。文献[9]分析了不同存储后端的写放大系数，指出 FileStore 在 HDD 和 SDD 环境下，写放大系数分别为 14.558 和 13.750。

针对 FileStore 的缺陷，Ceph 社区于 2015 年重新开发了 BlueStore[10]。两种存储后端的逻辑结构如图 4 所示。BlueStore 通过直接管理裸设备，缩短了 IO 路径。Ceph 社区设计了一个简化的文件系统 BlueFS，该文件系统绕过了本地文件系统层，解决了文件系统层次结构遍历效率低的问题。它通过使用 KV 索引来存储元数据，严格分离元数据和数据，提高了索引效率。这些改进解决了日志“双写”的问题，带来了较大的读写性能提升。



（a）BlueStore 的逻辑结构

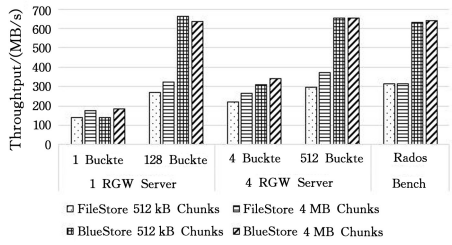


（b）FileStore 的逻辑结构

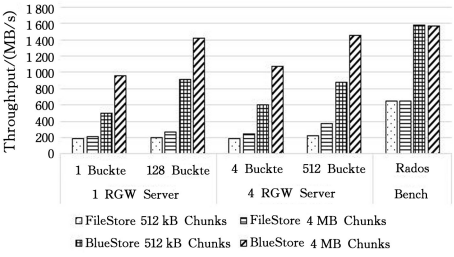
图 4　BlueStore 和 FileStore 的逻辑结构

Fig. 4　Logical structures of BlueStore and FileStore

由图 5 可知，BlueStore 的性能比 FileStore 的性能在三副本情况下提升了 1 倍以上，在使用纠删码的情况下性能提升最高，可达到原本的 3 倍。



（a）三副本场景下的性能对比



（b）4＋2 纠删码场景下的性能对比

图 5　BlueStore 和 FileStore 的性能对比[10]

Fig. 5　Performances of BlueStore and FileStore

虽然 BlueStore 在设计时考虑了与 SSD 及 NVMe SSD 闪存的适配，但其对新硬件或混合存储的支持不佳。除此之外，BlueStore 的设计也存在一些问题，如数据和元数据被存储在不同的位置，元数据结构和 IO 逻辑都较复杂，在 IO 较小的情况下可能存在双写问题，同时元数据占用内存较大[11]。

闪存盘在写入前需要将原有数据擦除。现有的 NVMe 设备并未记录哪些地址在写入前需要被擦除，因此导致设备内部的垃圾回收效率低下。原则上，异步的垃圾回收可以提高写入效率，若不修改磁盘布局，则垃圾回收的粒度较小，但是实际上该操作在设备和中间层实现的效果并不佳。针对闪存盘的特点，Ceph 社区提出了一种新的磁盘布局方式 SeaStore[12]，该布局在较高层次的驱动上进行垃圾回收。其基本思想是将设备空间分为多个空闲段，每个段的大小为 100 MB 到 10 GB，所有数据顺序地被流式传输到设备的段上，在删除数据时仅做标记不进行垃圾回收，当段中的利用率降低至某个利用率阈值时，会将其中的数据移到另一个段中。清理工作和写入工作混合在一起，以避免写延迟的波动。当数据全部清理完成后就丢弃整个段，设备可以擦除和回收这个段。SeaStore 主要应用于 NVMe 设备，使用 Seastar 框架进行基于未来规划模型（futures programming）的编程方式来实现 run-to-completion 模型，当其与 SPDK 和 DPDK 结合使用时，可以在网络消息层实现读写路径上的零（最小）拷贝。目前该存储引擎还在开发中，性能提升效果尚不明确。

国内的深信服公司设计了一个基于 SPDK 的用户态本地存储引擎——PFStore 来满足高性能分布式存储的需求，对数据使用追加写的方式，将元数据修改增量写入日志，在后期定时刷盘时再把数据写入 RocksDB 中。另外，该引擎启动多个实例来分别管理 SSD 的不同分区，这种方法类似于文献[13]中启动多个 OSD 来提升性能的方法。

近几年，随着 Open channel SSD、3DXPoint、非易失内存、SMR 等新型硬件的成熟和市场化，出现了一些针对特定硬件

的存储后端优化技术，我们将在第 3 节详细介绍面向新型硬件的优化技术。

2.2 纠删码的存储

大规模存储系统中的存储介质故障发生的频率很高，FaceBook 报告称其数据中心的存储设备每天损坏 3%[14]。多副本和纠删码是存储系统中常用的两种容忍介质损坏的方法。Ceph，HDFS 和 GFS 等分布式存储系统默认采用三副本，可以容忍两个设备同时发生故障，但其可用容量仅为物理容量的 1/3，而数据写入量却是原始数据的 3 倍。纠删码技术在部分数据丢失时可以通过结合剩余数据和校验码来恢复丢失数据。它虽然可以用更少的空间来实现与多副本相近的可用性，但是需要更多的计算，且跨节点的数据访问和更新开销较大。

Ceph 的 ECStore 是一个支持纠删码技术的后端存储，它可以配置不同的插件以使用纠删码。当前版本支持的插件包括 Jerasure，ISA-l 和 LRC。Jerasure 是一个开源的纠删码库，是 Ceph 默认的纠删码插件。ISA-I 针对某些定制化指令集的 Intel 平台进行了优化。Ceph 在使用纠删码时，数据写入主 OSD，主 OSD 负责编码并发送数据或校验数据至对应的从 OSD。数据读取时主 OSD 从各个从 OSD 中获取数据并解码，然后将解码后的数据发给客户端。

纠删码增加了额外的 IO 开销和计算开销，即使没有任何故障，针对读取操作，RS(10，4)编码的最大带宽比三副本方式的降低了 33%，且延迟增加了 50%。针对写操作，与三副本相比，RS(10，4)编码的带宽降低了 86%，延时增加了 7.6 倍。而针对 4～16 KB 的请求，三副本方式的吞吐量比 RS(6，3)的吞吐量高 8.6 倍[15]。因此，官方也不推荐在文件或块接口模式下使用纠删码[16]。

2.3 网络通信的优化

在分布式存储系统中，节点间需要通过网络通信来交换状态和数据。Ceph 有 3 种类型的通信模式，分别是 Simple，Async 和 XIO。Simple 线程模式对每个网络连接都创建了两个线程，分别用于接收和发送。Ceph 集群中 OSD、Monitor 节点及客户端之间都需要建立连接。随着集群规模的增长，创建的连接数和线程数会呈指数级增长，需要消耗更多的 CPU 和内存资源。在内存有限的情况下，Simple 模式将导致大量线程的频繁切换以致内存耗尽。Async 模式将连接和线程分开，通过线程池管理来维护线程的使用，用户可设置线程池中线程的数量。这是目前被广泛采用的方式，自 2017 年发布 Kraken 版本后，这已经成为默认的通信模式。XIO 模式使用了开源的网络通信库 accelio 来实现，现今仍处于实验阶段。目前针对网络通信优化的研究都是基于 Async 通信模式实现的。

Async 模式使用线程池，可兼顾资源和性能的平衡，但早期其设计是基于循环的简单调度方案，未考虑传输数据大小和线程负载。这种早期设计会导致工作线程负载不平衡，在高负荷情况下产生一些性能问题。Han 等[17]提出了一种用于 Ceph 文件系统的动态消息感知通信的调度程序，以解决工作线程调度不平衡的问题，从而提高性能。他提出的调度算法根据传入消息的类型来平衡工作线程的工作量，同时避免了工作线程之间不必要的连接转换。例如，一方面，该算法将低优先级消息(例如来自心跳连接的消息)分配给特定线程，以免干扰其他高优先级消息。另一方面，高优先级消息被平均分配给每个工作线程，以平衡线程之间的工作负载。同时其使用遗传算法(GA)来使不必要的连接转换最小化。测试结果表明，在相同的客户端工作负载下，该方法比原始 Async messenger 的性能高出 12.5%，在客户端的随机工作负载下，其性能比原始 Async messenger 高出 24%。

优化 Async messenger 还可以通过将多个工作线程分配给单个连接来处理来自该连接的流量。但是，由于多个线程争用访问连接中的共享资源，这种映射结构会引起与锁定维护有关的另一种开销。文献[18]提出了锁争用感知信使(Async-LCAM)，它为每个连接分配多个工作线程并维护线程的生成。Async-LCAM 更改了线程到连接的映射结构，这样有两个好处。1)由于多个工作线程竞争为所有连接处理消息，它自动解决了由 Async Messenger 引起的负载不平衡问题。这使 Async-LCAM 可以更及时、有效地处理消息，并实现负载平衡。2)映射结构使连接之间的冗余最大化。但是，若将多个线程分配给单个连接，则有可能发生锁争用。也就是说，Async Messenger 不需要为每个连接锁定线程，因为一个连接的消息仅由一个工作线程处理。但是，Async-LCAM 需要为每个连接锁定线程，因为多个工作线程试图处理来自同一个连接的不同消息。值得注意的是，连接是一个消息相关的关键结构，包含从读取操作到调度操作的不同消息，若多线程同时处理同一个连接的不同消息，则共享数据结构可能会发生损坏。因此，为了减小由锁争用引起的开销，Async-LCAM 通过考虑锁争用来将多个线程动态分配给连接。

文献[19]也修改了 Async messenger 使多个工作线程可以处理来自同一连接的消息。但是，该文献没有对锁争用的问题进行改进。出于对平衡工作线程负载的考虑，文献[19]设置了一个平衡算法。该算法首先决定是否可以将工作负载最大的工作线程中的连接重定位到工作负载最小的工作线程上。然后，负载平衡算法检查基于重定位的标准偏差是否小于前一次迭代中的标准偏差。在移动连接时，该算法仅使用 epoll_ctl 函数来添加和删除相应的套接字描述符，避免了总体延迟的增加。另外，该文还应用了多连接的方法，使主 OSD 可以同时向从 OSD 发送复制操作，提高了带宽利用率，并减小了复制操作的开销。实验结果表明，经过优化的 Ceph Messenger 在 4K 消息的随机写入中的性能比原始 Messenger 的性能最高超出 40%。此外，具有优化通信子系统的 Ceph 与原始 Ceph 相比，其性能提高了 13%。文献[19]提出的方法对通信子系统性能的提升较大，但是从 Ceph 对外访问接口的性能方面来看，提升很小。

Ceph 的 Luminous 版本将 Async 网络通信模型作为默认的通信方式。虽然 Async 实现了 IO 的多路复用，使用共享的线程池来实现异步发送和接收任务，但是如何平衡 Async 工作线程的负载也是一个值得关注的问题。表 2 总结了上述 3 种算法的优化方案。

表 2　网络通信调度的优化方案

Table 2　Optimization schemes for network communication scheduling

序号	通信线程数量	通信线程与 IO 处理对应关系	调度策略	优化结果
原始	启动后数量不变	多对一，建立联系后不再变化	循环调度	
1	启动后数量不变	根据消息类型，将低优先级消息分配给特定线程，高优先级消息平均分配给连接池工作线程，动态平衡负载	使用遗传算法，检查工作线程工作量，并定期执行算法以重新平衡	与原始 Ceph 系统相比，性能高出 12.5%[17]
2	追踪固定间隔内的每个连接，动态增删线程	将多个工作线程分配给单个连接以处理来自该连接的流量，使用锁争用感知信使解决冲突	调整 maxevents 参数，使每个工作线程中获取的连接数保持平衡	与原始 Ceph 系统相比，吞吐量和延迟分别提高了 184 和 65%[18]
3	根据负载情况触发平衡算法，动态重定位线程连接	为每个连接分配多个工作线程，但没有处理锁争用问题	在 OSD 之间使用多个连接，两个创建的连接被捆绑并视为一个连接，共享变量	与原始 Ceph 相比，性能提高了 13%[19]

RDMA 是一种低延迟、高性能的网络传输协议，已被广泛应用于高性能计算环境中。为了在 Ceph 中利用 RDMA 以实现高速的数据传输，开发社区提出了两种方案[20]。第一种方案是降低 Ceph 对网络层状态的要求，减少 Messenger 需要实现的逻辑。现在的 Xio Messenger 规定的语义和策略过于复杂，使用新的网络协议实现的难度大；减少 Messenger 的逻辑则需要增加上层的逻辑。第二种方案是基于目前的 Async Messenger 的框架，扩展出支持 RDMA 的网络后端而无需关心上层的会话逻辑。国内的 XSKY 公司和 Mellanox 公司合作提出了基于 Async Messenger 的网络通信引擎。这种修改使用 RDMA 的双边通信机制，性能提升有限。并且在当前版本的代码实现中，RDMA 只可用于客户端与服务器之间、服务器与服务器之间的通信，不能在两个网络中同时被应用，这也限制了该方案的应用。

2.4　数据放置方法的优化

经典 Ceph 存储系统在副本模式下选择存储节点时，仅以节点存储容量为唯一选择条件，并没有考虑到网络和节点的负载状况，这影响了系统在网络性能差和节点高负载的情况下的读写性能。为解决这些问题，文献[21]设计了基于软件定义网络技术的 Ceph 存储系统模型和存储节点选择策略，首先利用软件定义网络技术实时获取网络和负载状况，以简化网络配置和减小测量开销，然后通过建立并求解出综合考虑了多种因素的多属性决策数学模型来确定存储节点的位置。在实际环境中对设计的存储节点选择方法进行读写操作的测试，结果表明，与现有的 CRUSH 算法相比，提出的存储节点选择方法可以在保持与原有 Ceph 系统相同的写操作性能的同时，针对 4 KB 对象的 100% 读操作的响应时间比原有的 Ceph 集群的缩短了 10 ms 左右，针对 4 096 KB 对象的 100% 读操作响应时间相对缩短了 120 ms 左右。这种方法需要获取网络的实时性能以用于数据放置策略的调整，引入了网络负载采集的开销，在大规模集群场景下应用受限。并且由于该方法会频繁更新 CRUSH 算法的参数，其读取性能会有所下降。

文献[22]使用基于遗传算法改进的数据放置算法来优化原始 CRUSH 算法在选择存储节点时只考虑节点存储容量的不足，通过对分布式计算框架 MapReduce 的研究，利用混合整数线性规划算法来求解异构环境下 Ceph 的最优数据放置策略。当对象存储设备、集群数量或者任务数量较多时，因为利用混合整数线性规划算法的时间复杂度太高，所以该文还通过遗传算法优化了 CRUSH 算法。实验结果表明，该算法得到的数据放置策略与传统 Ceph 中的策略相比，在性能方面的提升高达 25.6%。

但文献[21]和文献[22]都没有考虑 Ceph 在扩充集群时因数据迁移所带来的开销。因为 CRUSH 是一种去中心的数据分布算法，所以在集群扩展时，数据迁移不可控，进而导致性能下降。文献[23]提出了一种可控的 CRUSH 算法 MAPX 来解决这个问题。MAPX 增加了一个虚拟选择层，通过时间戳来记录集群中各节点加入的时间，将批量加入的节点看作一个新的虚拟层；虚拟层将集群分为了不同的子集，子集在数据均衡和创建时可以选择不同的虚拟层；由于新层不会影响旧层的权重，旧对象在旧层中的放置不会改变，减少了数据迁移，实现了对集群扩容过程的性能控制。文献[23]通过扩展 RBD 元数据结构将 MAPX 应用于 Ceph-RBD，以在扩展层的粒度下维护和检索近似的对象创建时间。实验结果表明，基于 MAPX 的免迁移系统比基于 CRUSH 的原始系统的尾部等待时间高出 4.25 倍。该文提出的方法适用于批量增加大量节点的情况，通过控制层次合并时机来推迟数据迁移造成的性能影响。但是，若多次扩容创建的虚拟层不进行合并，则该方法相当于将 Ceph 集群分割为几个独立的子集群，在性能和可用性方面可能会产生一些问题。

2.5　配置参数性能调优

Ceph 存储系统的可配置参数有 1 500 多个，参数的调整对系统性能有较大的影响。默认配置针对不同的硬件和应用通常不是最优配置。文献[24]通过锁优化和系统参数调优技术使系统的吞吐率提升了 1.6 倍，但其并未讨论修改了哪些配置参数。文献[13]虽然详细介绍了在全闪存环境下需要调整哪些参数（包括内核、文件系统、磁盘缓存、RADOS 和 RBD 等），但是没有给出调整前后的性能对比。文献[25]分析了在云计算环境下使用 Ceph 时不同的存储性能需求，讨论了影响范围不同的各个配置参数，包括集群级别、资源池级别和异构池等。对比测试结果表明，修改一个参数对虚拟机的性能影响可达 44.5% 以上。

Cao 等[26]提出了一种黑盒自动调整存储系统参数的方法，该方法迭代尝试不同的配置，测量系统的性能，并根据之前的信息选择下一次的参数配置。但该方法每次需要运行具有大量输入数据集的应用程序，会耗费大量的时间且会占用大量的系统资源。因此，文献[27]提出了一种基于随机森林（Random Forest，RF）和遗传算法的参数自动调优方法，用于自动调整 Ceph 的参数配置，以优化 Ceph 系统的性能。该方法首先通过随机森林算法来建立性能预测模型，然后将性能

预测模型的输出作为遗传算法的输入，对 Ceph 参数配置方案进行自动迭代优化，以找到最佳参数配置。与需要执行应用程序的方法相比，建立性能模型能够更快地预测应用程序的性能，从而减少系统资源的占用，节省大量测试时间。实验结果表明，与默认的 Ceph 参数配置相比，调优后的参数配置使 Ceph 文件系统的读写性能平均提高了 1.4 倍，且其寻优的耗时远低于文献[26]中的黑盒参数调优方法。

Intel 开发并开源了一个性能优化工具 CeTune[28]，该工具可用于 Ceph 集群的部署、测试、分析和调优。该工具是一个交互性的调优工具，尚不能自动寻找最优配置。

分布式存储系统的性能调优仍是一个具有挑战性的问题。参数组合导致问题解空间大，且参数之间会互相影响。在数据库领域已有一些利用机器学习和决策树的方法进行自动调优的成功案例，而在分布式存储系统领域，相关研究仍处于起步阶段。

3 面向特定硬件环境的优化

随着 3DX point 和非易失内存等技术的发展与成熟，最快的存储介质性能已接近内存性能。使用新型存储器件的系统中软件已成为瓶颈[29]。如图 6 所示，HDD，SATA SSD，NVMe NAND SSD，3D XPoint Storage，3D XPoint Memory 系统中的软件造成的延迟分别是 0，10%，20%，40%和 90%。通过重构软件的体系结构来充分发挥高速存储介质的性能是目前的一个研究热点。

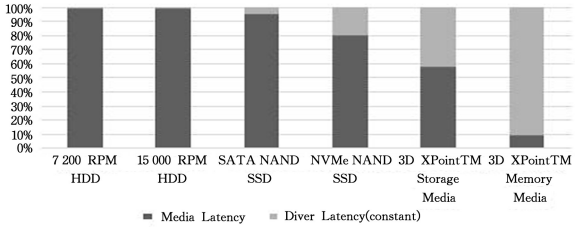


图 6 存储系统中软硬件导致的延迟比重

Fig. 6 Latency caused by hardware and software in storage system

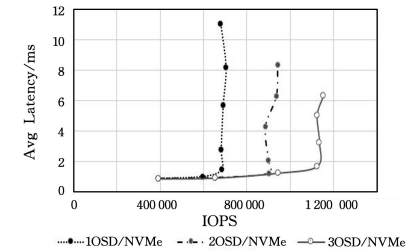
3.1 固态存储

固态存储设备相比磁盘设备，在性能、功耗和机架密度上具有显著的优势。SATA 接口限制了固态存储设备的最大吞吐率。Intel 提出的利用 PCI-E 总线来访问固态存储的 NVMe(Non-Volatile Memory Express)接口方法提供了通用的高速存取方案。

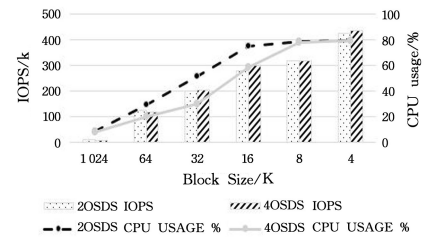
使用 NVMe 的固态存储设备在吞吐量和延迟性能上比传统的磁盘高出 1～2 个数量级，因此在总的 IO 处理时间中，软件造成的延迟占据更大的比例。现有的存储系统为低速硬件设计了合并写、异步写等机制，但是这些机制并不适用于高速存储设备。随着存储设备性能的进一步提升，存储系统软件栈的性能和效率对存储系统的影响越来越大。存储系统因受制于低效冗余的软件栈而不能充分发挥硬件性能。

文献[29]总结了 Intel 近年来在 Ceph 社区针对 NVMe 的各项优化，包括客户端的缓存和 RBD 优化，利用 DPDK 和 RDMA 优化消息传递，在 OSD 中优化 PG 锁，以及对 RocksDB 的优化等。SPDK 是 Intel 提出的使用 NVMe SSD 作为后端存储的应用软件加速库。该软件库的核心是实现用户态、异步、无锁、轮询方式的 NVMe 驱动。虽然 SPDK 等新型驱动可以将 NVMe SSD 的性能最高提高 6 倍，但是在 Ceph 中直接使用 SPDK 却没有明显的性能提升。其主要原因在于在 BlueStore 的处理中有很多线程协作，线程间的互斥和切换开销较大。

在 Ceph 中，OSD 使用异步 IO 等待 IO 完成，多线程可以充分利用 NVMe SSD 多通道的特点来提升性能。由于一个 OSD 无法充分利用 NVMe SSD 的带宽，研究人员发现将 NVMe SSD 进行分区，然后在其上运行多个 OSD 可显著提高性能。图 6(a)给出了一个 NVMe SSD 分别使用 1 个 OSD、2 个 OSD 和 4 个 OSD 时的性能，可以看到在 1 个 SSD 上运行 4 个 OSD 时的随机读，其 IOPS 增长很快但延迟增长缓慢[13]。但是将 SSD 分区并同时支持多个 OSD 也有一些缺点，如降低了可靠性、小块随机写延迟增大、需要更多内存和 CPU 资源等。另一项对于随机写的测试结果如图 6(b)所示，在 1 个 SSD 上运行 2 个 OSD 时的 IOPS 与运行 4 个 OSD 时的 IOPS 相近，且需要的 CPU 资源更少[30]。使用多个 OSD 的提升效果与 SSD 的性能和 CPU 的性能相关，且需要占用较多的 CPU 和内存资源，对可用性也有影响，因此不适用于大规模的生产环境。



(a)多 OSD 读性能的对比[13]



(b)多 OSD 写性能的对比[30]

图 7 多 OSD 读写性能的对比

Fig. 7 Read and write performance comparison of multiple OSD

除了 NVMe 以外，现在还有一些使用 SATA SSD 来代替 HDD 的策略，但直接替换存储介质的性能提升有限。Ceph 针对 HDD 设计了很多异步和调整写顺序的机制，但这些机制反而降低了 SSD 的性能。当使用 Ceph 集群和全闪存 SSD 进行 4K 随机写入/读取的性能测试时，随机写入性能也只能达到 16K IOPS。当线程数增加到 32 个及以上时，IOPS 几乎保持不变，且延迟急剧增加。随机读取时，当线程数少于 32 时，IOPS 较低，且延迟较高[24]。

为适应 SSD，文献[24]针对 PG 的锁定进行了优化，引入了一个待处理队列，使得不属于同一 PG 的请求将不会被阻塞。另外，因为等待日志记录也造成了一定的延迟，所以还需

要将日志记录从同步更改为异步，使其不再位于关键路径上，以减小日志记录开销。同时，减少事务中的操作次数，使用直写式高速缓存来防止同时请求读取和写入操作的情况，进行轻量级事务，缓解处理写入操作导致的性能下降。测试结果表明，较小的随机写入(4K)工作负载获得了 20 倍的性能提升。在随机读取(4K)的情况下，性能提高了 2 倍。该研究提出的 PG 锁优化仅适用于 Ceph 的早期版本，Luminous 之后的版本中采用的 PG 加锁机制比该文提出的方法更优。另外，该研究提到参数调整对 Ceph 全闪存环境的性能提升较大，但未详细说明参数调整的方法。

Lu 等[31]提出了基于开放通道 SSD(Open-channel SSD)的 OCStore。OCStore 直接在闪存上管理对象，减少了对象存储、文件系统和 FTL 层的冗余功能。它提供的流式事务更新不仅确保了利用非覆盖 flash 写功能的多页原子性，还为独立的 I/O 流提供了隔离，同时支持对不同通道的并行访问。OCStore 还协调不同的通道来实现事务感知调度。评估结果表明，OCStore 的性能比最新的 Bluestore 高出 1.5 倍至 3.0 倍，同时延迟稳定，在高负载情况下还减少了高达 70% 的写流量。该设计需要使用 Open-channel SSD，但市面上支持 Open-channel 接口的商用产品不多。

3.2　非易失内存

存储是数据密集型系统中最慢的组件。尽管基于 NVMe 的固态驱动器提供了更快、更持久的存储，IO 性能已大大提高，但其仍然比系统中的其他组件慢。随着 NVDIMM (Non-Volatile Dual In-line Memory Module)产品的出现，可字节寻址的非易失性存储器将提供与内存相近的 IO 性能。

文献[32]模拟了在 Ceph 系统中使用 NVDIMM 作为底层介质的性能，实验结果表明，针对单个节点，将所有内容映射到 NVDIMM 可以大大提高吞吐量；针对不同的对象大小，吞吐量均可以提高到 100% 以上。当考虑网络时，若持久性内存仅用于某些组件，总体网络使用率则不会发生重大变化。但是，当所有分区都映射到 NVDIMM 时，网络使用率会急剧上升，网络将成为新的瓶颈。另外，使用 NVDIMM 获得高性能 IO 的同时会大大增加 CPU 的负载。

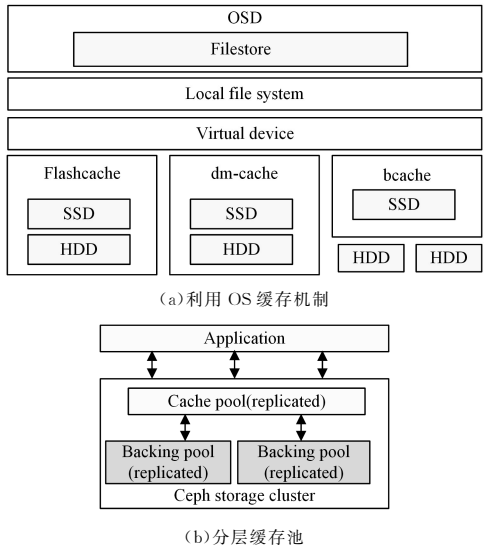
文献[33]分析了 BlueStore 中写操作的关键路径，首先将写前日志(Write Ahead Log, WAL)的相关数据先写入 NVM，然后异步刷新至 SSD。测试结果表明，当线程数为 16 时，该方案性能可达原始 BlueStore 的 4.23 倍。但该方案的实质是将 NVM 作为一个高速缓存，没有设计额外的机制来保证 NVM 和 SSD 的数据一致性，如断电等故障发生时仍然会导致数据丢失。

Intel 将 Client 端的 NVM 作为缓存，提出了 3 个优化方案，大大提升了块存储接口的访问性能[34]。方案一提出由于 Ceph 快照中的父对象是只读的，Client 端将其缓存在本地 SSD 中以提高读取性能。但该方案仅缓存特定的不变对象，不具有通用性。方案二利用 Client 端的 NVM 实现了一个写回缓存，写入 NVM 的数据并将其不定期刷新至 OSD 中。该方案性能提升的效果明显，能将 99.99% 的写入延迟缩短到 1/10 以下。但是因为在客户端发生故障时，保存在 NVM 中的数据没有写入 Ceph 后端的 OSD，所以会造成数据不一致。为了解决这个问题，方案三通过 RDMA 技术为 Client 节点和 OSD 节点的 NVM 空间建立镜像，以避免因 Client 故障导致的数据丢失。复制写日志机制将数据同时持久化至 Client 和 OSD 的 NVM 中，当 Client 不发生故障时，OSD 仅提供镜像空间，无需额外的 CPU 等资源。基于客户端的缓存性能提升明显，但为每个客户端配置 NVM 和 RDMA 的成本较高，且该方案并未充分利用 OSD 端的 NVM 特性。

3.3　混合存储

在 Ceph 集群中可以使用 SSD 作为保存日志或缓存来提高访问性能。

Ceph 支持使用高速存储设备作为缓存来加速 IO 性能。目前有两种不同的缓存实现方式：1)在 OSD 内部使用缓存；2)将高性能节点组成缓存层[35]。根据第一种方案，在使用 FileStore 时，文件系统可以识别并使用异构来存储介质，并将其中的 SSD 作为缓存，其架构如图 7(a)所示。这种方案可以依赖已有的缓存工具(如 dm-cache，bcache，FalshCache)来实现该功能，也可以使用现有的多种缓存控制方法。其中，dm-cache 作为 linux 内核的一部分，采用 device mapper 机制以允许用户建立混合卷；bcache 是 linux 内核块层缓存，使用 SSD 作为 HDD 硬盘的缓存，从而起到加速作用；Flash Cache 可智能缓存最近读取过的用户数据或元数据，从而加快数据访问。



(a)利用 OS 缓存机制

(b)分层缓存池

图 8　两种混合存储缓存的架构

Fig. 8　Caching architecture of two heterogeneous storage

第二种方案的实现方式是将独立的设备或节点组成缓存池，在慢速节点保存冷数据，在高性能节点保存热数据。在这种方案中，高速节点被组织成为缓存层，该缓存层也是一个 RADOS 池，不仅具有持久化的能力，还具有独立的 CRUSH 策略。该层以下是基于 HDD 的 RADOS 池，其可以采用三副本的 Replicate PG 作为后端，也可以采用 Erasure coded 作为后端。

在云计算环境中虚拟机需要不同性能的配置。文献[36]讨论了在 Ceph 集群中配置和使用异构资源池的方法，包括使用不同的硬件介质、底层文件系统和配置等。文献[37]针对混合存储集群提出了一种数据分发和迁移策略，称为偏向对象存储策略(Biased Object Storage Strategy，BOSS)。Wu

等[37]研究了几种企业级工作负载的读/写行为，发现不同对象之间存在较大的读写频率差异。大多数读写请求都集中在少数对象上，且大多数对象都是读密集型或写密集型的。基于这些观察，WU 等[37]提出了将对象动态移到不同存储设备上的策略，包括初始化、分析和迁移这 3 个阶段。BOSS 从访问日志获得数据访问模式，根据负载平衡/性能约束以及系统配置来计算迁移阈值，随着访问模式的变化动态调整阈值、迁移对象并修改对应的元数据，在从对象 id 到 PG 的映射函数中加入 bias factor 参数，使不同访问特征的对象被放到不同的 PG 中。实验结果表明，在典型的负载下，BOSS 可减少 SSD 的写入次数到 64%，平均性能提高 29.51%。该策略对负载稳定的重复任务提升效果明显，但是不适用于在线动态调整数据布局，且文献[23]没有讨论离线负载分析和迁移的开销。

4 面向应用场景的性能优化

Ceph 分布式存储系统的应用场景较多，在云计算平台中主要提供块存储服务；在 HPC 领域主要通过文件来访问接口，现在也有通过对象接口的 HPC 应用；在云存储领域主要通过对象来访问接口。针对不同的应用场景和负载特征，对 Ceph 的 IO 流程进行优化，可以得到更好的性能。

4.1 面向云计算

在云计算 OpenStack 的场景中，Ceph 可以替代 Cinder 组件提供块存储服务。针对块存储服务，保持 Ceph 中以对象为单位的事务一致性是没有必要的。一方面，传统的硬盘等介质并不支持事务操作，因此块访问接口默认不支持事务。另一方面，从应用的角度，同样大小的数据块在读写时，读写范围在对象内的读写操作支持事务，读写范围跨对象的读写操作则不支持事务。基于对象的事务设计无法被封装在块设备服务中。根据这一特点，如图 9 所示，我们在处理 RBD 数据块对象时，不进行 WAL 操作，从而将写的数据量减少一半，写入 IOPS 提升了 3～5 倍，写入吞吐率提升了 4 倍[8]。

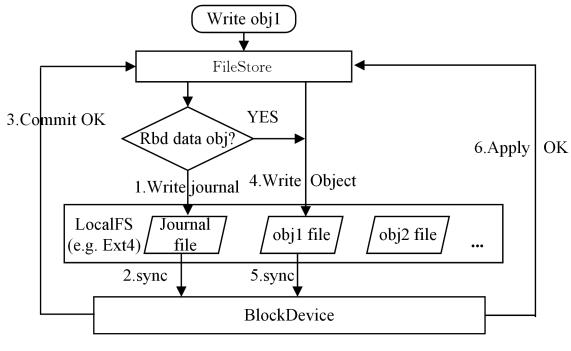


图 9 面向 RBD 数据的写流程优化

Fig. 9 Optimization for RBD writing

4.2 面向高性能计算

随着大数据分析和机器学习的发展，高性能计算(HPC)系统也需要支持多种不同类型的存储服务。除了传统的 Lustre，beeGFS，GPFS 外，CephFS 也成为了 HPC 系统的候选之一[38]。在 2019 年 11 月公布的 IO500 表单中，SUSE 公司使用 CpehFS 获得了第 32 位和第 35 位的成绩[1)]。除了文件接口外，HPC 也开始使用对象来访问接口。文献[39]对比了美国国家能源研究科学计算中心(NERSC)提出的 3 种对象存储系统(Intel DAOS，Ceph RADOS，Openstack swift)的访问性能，结果表明，Ceph 比 DAOS 慢，比 swift 快。

Ceph 并不是为 HPC 环境设计的，因此可以针对 HPC 工作负载进行优化。文献[40]分析了高性能计算中数据密集型数据的访问特征，将 HPC 中数据密集型应用访问的文件分为读密集型(RH)、写密集型(WH)或读写密集型(RW)。该文首次量化了 3 种文件的比例，读密集型的文件占 22%；写密集型文件占 7%，这 7%的文件被不断写入，但未被读取；剩下 71%的 HPC 文件是读写密集型文件。这些文件的读写特征可用于设置文件放置决策，如包括 Burst Buffer 区在内的分层存储放置策略。

文献[41]提出了一种针对 HPC 负载，利用 F2FS 文件系统和混合存储系统特性的性能优化方法 F2FS-split。F2FS 以日志结构写入数据，因此较小的随机写入将合并为较大的顺序写。F2FS 支持多驱动器，可指定不同介质保存不同的内容。F2FS-split 将 F2FS 的元数据区域分配到快速 SSD 中，并将文件数据区域分配到慢速 HDD 中。测试结果表明，在写入负载为主的工作负载中，F2FS-split 的性能分别比 F2FS 和 XFS 高出 39%和 59%。该文还提到修改 Ceph RADOS 对象的大小可以进一步提高读取速度。该方法通过修改底层文件系统来使其更好地适配 Ceph 以获得较好的性能，但该方法涉及的硬件环境和负载都有局限性。

4.3 面向云存储

在 CephFS 中，MDS 处理 Client 的元数据请求，文件系统的元数据也保存在对象中，并保存在多个 OSD 上。尽管元数据仅占数据总量的 10%，但是对元数据的访问数量占系统总访问量的 50%以上。因此，减少对元数据的访问延迟对于提高 CephFS 的整体性能具有重要意义。文献[42]在 MDS 中利用缓存技术来减少元数据的访问延迟。该方案在 MDS 中缓存元数据，并且合并元数据的更新操作以批量写入后端的 OSD 中。为了保证可靠性，该方案还将更新日志备份至其他节点。Zhan 等[42]用 filebench 进行了对比测试，结果表明，该方案对 mkdir 等大量更新元数据的操作有较大的性能提升，对其他性能的提升不大。该方案的缓存方式可提高元数据的写入性能，但是若将写入的内容同步至其他节点，则性能提升幅度有限。另外，针对多 MDS，节点内的缓存会导致不同的 MDS 间的数据不一致。

分布式文件系统在访问小文件时，由于数据量小，元数据访问所占开销比例增加，对小文件访问性能较低。因此，Wang 等[43]提出了一种面向小文件的访问优化方法。该方法通过将小文件的数据存储在元数据区域来修改元数据查询和数据访问协议，在查询元数据时同时推送文件的数据，降低了小文件访问的延迟。其同时设计了数据自动迁移算法，当文件大小超过阈值时，自动将数据迁移至普通存储区域，实现了与原有文件访问方式的兼容。实验结果表明，优化后的

1) http://io500.org

CephFS 的小文件访问性能提升了约 1 倍。文献[44]提出了一种优化大文件写入性能的方法，该方法通过创建多个线程将文件进行拆分，同时调用 LIBRADOS 的功能将数据写入 Ceph。此方法需要修改客户端程序，对原有的数据读写和组织方式都有较大的影响。

5　未来展望

针对前面提到的不同的性能优化方法，本节从 Ceph 内部机制优化、基于新型硬件和面向不同负载优化这 3 个方面对性能优化问题的未来研究方向进行了展望。

5.1　Ceph 内部机制的优化

Ceph 发展至今，其规模和复杂性不断增大。数据分发、元数据管理和对象一致性保证等方面的逻辑复杂，目前的多线程和加锁机制效率较低。采用新的内存分配机制和高效 kv 数据来管理子系统可能对性能有所提升。目前内置的性能采集机制不完善，采集内容和方法、性能数据分析这两方面都有改进的空间。

5.2　基于新型特定硬件的优化

随着新型存储介质的发展，NVM 和 3DX point SSD 等介质的访问性能比传统 HDD 提升了 2～4 个数量级。存储系统优化需通过结合硬件特征来重构存储系统的体系结构，删除冗余抽象和功能，并重新分配软硬件的功能。存储软件中针对传统硬件的优化方法可能会降低性能，如异步写、随机写合并等。SSD 的文件转换层的垃圾回收、空间映射与管理功能也可移至软件实现。多种不同性能的介质也将长期共存于存储系统中，根据数据冷热和介质特性自适应实现高性价比的数据分布也是一个研究方向。Ceph 虽然已支持 RDMA 协议，但是双边操作方式性能的提升有限，设计新的通信机制、采用单边操作可进一步提高性能。

5.3　面向应用场景的自适应优化

Ceph 应用场景众多，不同应用的访问特征和数据分布差异较大。在不同的负载情况下，基于人工制定的存储优化方案不能满足要求。根据不同应用负载的自适应优化技术也是一个挑战，其研究内容包括利用访问标签来实现不同应用的性能服务质量保证以及性能隔离、利用机器学习技术自动寻找最优配置，以及动态的数据预取和迁移。

结束语　本文介绍了 Ceph 分布式存储系统的发展历史和特点，并从内部优化机制、面向硬件设备和应用场景这 3 个方面梳理了现有的性能优化方法。作为一个开源的统一存储系统，Ceph 提供了高性能、高可扩展性和高可用的块、文件和对象存取功能。Ceph 的特点使之被广泛应用于云计算、高性能计算和大数据处理领域。目前，Ceph 的性能优化研究工作虽然有一定的进展，但仍有很多问题尚未完全解决。

参 考 文 献

[1] WEIL S, BRANDT S, MILLER E, et al. CRUSH: Controlled, scalable, decentralized placement of replicated data[C]// Proceedings of the 2006 ACM/IEEE Conference on Supercomputing. SC, 2006: 122.

[2] WEIL S, BRANDT S, MILLER E, et al. Ceph: A scalable, high-performance distributed file system[C]// 7th USENIX Symposium on Operating Systems Design and Implementation (OSDI). 2006: 307-320.

[3] OPENSTACK ORG. 2015: Openstack user survey [EB/OL]. https://www.openstack.org/analytics.

[4] INTEL. Ceph Benchmark Tools [EB/OL]. https://github.com/ceph/cbt.

[5] CEPHCOMMUNITY. Teuthology[EB/OL]. https://github.com/ceph/teuthology.

[6] WAN H T, LI Z H, ZHANG X. A Layered Perflormance Monitoring and Gathering Method of Cloud Storage[J]. Joumal of Northwestem Polytechnical University, 2016, 34(3): 529-535.

[7] ZHANG X, KONG L, ZHU S, et al. FSObserver: A Performance Measurement and Monitoring Tool for Distributed Storage Systems[C]// IFIP International Conference on Network and Parallel Computing. Springer, Cham, 2018: 142-147.

[8] ZHANG X, WANG Y Q, WANG Q, et al. A New Approach to Double I/O Performance for Ceph Distributed File System in Cloud Computing[C]// 2019 2nd International Conference on Data Intelligence and Security (ICDIS). IEEE, 2019: 68-75.

[9] LEE D, JEONG K, HAN S, et al. Understanding Write Behaviors of Storage Backends in Ceph Object Store[C]// IEEE Conference on Mass Storage Systems and Technologies. IEEE, 2017, 10.

[10] WEIL S. Bluestore: A New Storage Backend For Ceph [EB/OL]. https://www.slideshare.net/sageweil1/bluestore-a-new-storage-backend-for-ceph-one-year-in.

[11] AGHAYEV A, WEIL S, KUCHNIK M, et al. File systems unfit as distributed storage backends: lessons from 10 years of Ceph evolution[C]// ACM SIGOPS 27th Symposium on Operating Systems Principles. ACM, 2019: 353-369.

[12] CEPH DOCUMENTATION. Seastore [EB/OL]. https://docs.ceph.com/docs/master/dev/seastore/.

[13] CEPH COMMUNITY. Tuning for All Flash Deployments [EB/OL]. https://tracker.ceph.com/projects/ceph/wiki/Tuning_for_All_Flash_Deployments#Tuning-for-All-Flash-Deployments.

[14] SATHIAMOORTHY M, ASTERIS M, PAPAILIOPOULOS D, et al. XORing Elephants: Novel Erasure Codes for Big Data [C]// 39th International Conference on Very Large Data Bases (VLDB). VLDB Endowment, 2013: 325-336.

[15] SUNGJOON K, ZHANG J, MIRYEONG K, et al. Understanding System Characteristics of Online Erasure Coding on Scalable, Distributed and Large-Scale SSD Array Systems[C]// 2017 IEEE International Symposium on Workload Characterization (IISWC). IEEE, 2017: 76-86.

[16] ZHOU Y. Ceph Erasure Coding Introduction [EB/OL]. https://software.intel.com/content/www/us/en/develop/blogs/ceph-erasure-coding-introduction.html.

[17] HAN Y, PARK S, LEE K. A dynamic message-Aware communication scheduler for Ceph storage system[C]// Proceedings-IEEE 1st International Workshops on Foundations and Applications of Self-Systems. IEEE, 2016: 60-65.

[18] BODON J, AWAIS K, SUNGYONG P. Async-LCAM: a lock

contention aware messenger for Ceph distributed storage system [J]. Cluster Computing, 2018, 22(2): 1386-7857.
[19] SONG U, JEONG B, PARK S, et al. Performance Optimization of Communication Subsystem in Scale-Out Distributed Storage [C]// 2017 IEEE 2nd International Workshops on Foundations and Applications of Self Systems (FASW). IEEE, 2017: 263-268.
[20] GITHUB. msg/async: ibverbs/rdma support [EB/OL]. https://github.com/ceph/ceph/pull/11531.
[21] WANG Y, YE M, HE Q, et al. A New Node Selecting Approach in Ceph Storage System Based on Software Defined Network and Multi-attributes Decision-making Model[J]. Chinese Journal of Computers, 2019, 42(2): 95-110.
[22] SHA H M, LIANG Y, JIANG W, et al. Optimizing Data Placement of MapReduce on Ceph-Based Framework under Load-Balancing Constraint[C]// 2016 IEEE 22nd International Conference on Parallel and Distributed Systems (ICPADS). IEEE, 2016: 585-592.
[23] WANG L, ZHANG Y M, XU J W, et al. MAPX: Controlled Data Migration in the Expansion of Decentralized Object-Based Storage Systems[C]// 18th USENIX Conference on File and Storage Technologies. FAST 20, 2020: 1-12.
[24] OH M, EOM J, YOON J, et al. Performance Optimization for All Flash Scale-Out Storage[C]// IEEE International Conference on Cluster Computing. IEEE, 2016: 316-325.
[25] MEYER S, MORRISON J P. Impact of Single Parameter Changes on Ceph Cloud Storage Performance[J]. Scalable Computing: Practice and Experience, 2016, 17(4): 285-298.
[26] CAO Z, TARASOV V, TIWARI S. Towards better understanding of black-box auto-tuning: a comparative analysis for storage systems[C]// Proceedings of the 2018 Annual USENIX Technical Conference. Berkeley. USENIX Association, 2018: 893-907.
[27] CHEN Y, MAO Y C. Automatic tuning of Ceph parameters based on random forest and genetic algorithm[J]. Journal of Computer Applications, 2020, 40(2): 347-351.
[28] INTEL. CeTunetools[EB/OL]. https://github.com/intel/CeTune.
[29] Flash Memory Summit 2018: Ceph Optimizations for NVMe [EB/OL]. https://www.flashmemorysummit.com/English/Collaterals/Proceedings/2018/20180808_FTEC-202-1_Ye.pdf.
[30] CEPH COMMUNITY. Bluestore Advanced Performance Investigation[EB/OL]. https://ceph.io/community/part-4-rhcs-3-2-bluestore-advanced-performance-investigation/.
[31] LU Y, ZHANG J, YANG Z, et al. OCStore: Accelerating Distributed Object Storage with Open-Channel SSDs[C]// 2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS). IEEE, 2019: 271-281.
[32] PYDIPATY R, GEORGE J, SAHA A, et al. The Effect of Non Volatile Memory on a Distributed Storage System[C]// IEEE International Conference on High Performance Computing Data and Analytics. IEEE, 2017: 11-17.
[33] JIN Z S. Optimization of Distributed Storage on Commodity SSD using NVDIMM[D]. Seoul: Graduate School of Seoul University, 2017.
[34] PETERSON S. Using persistent memory and RDMA for Ceph client write-back caching[C]// Storage Developer Conference. SNIA, 2019: 24-27.
[35] WEIL S. Erasure Coding And Cache Tiering[EB/OL]. https://www.slideshare.net/sageweil1/20150222-scale-sdc-tiering-and-ec.
[36] STEFAN M, JOHN P M. Supporting Heterogeneous Pools in a Single Ceph Storage Cluster[C]// International Symposium on Symbolic & Numeric Algorithms for Scientific Computing. IEEE, 2016: 352-359.
[37] WU L, ZHUGE Q, SHA H M, et al. BOSS: An Efficient Data Distribution Strategy for Object Storage Systems with Hybrid-Devices[J]. IEEE Access, 2017, 5(1): 23979-23993.
[38] LÜTTGAU J, KUHN M, DUWE K, et al. Survey of storage systems for high performance computing[J]. Supercomputing Frontiers and Innovations, 2018, 5(1): 2313-8734.
[39] LIU J, KOZIOL Q, BUTLER G F, et al. Evaluation of HPC Application I/O on Object Storage Systems[C]// IEEE/ACM International Workshop on Parallel Data Storage & Data Intensive Scalable Computing Systems. IEEE, 2018: 24-34.
[40] PATEL T, BYNA S, LOCKWOOD G K, et al. Uncovering Access, Reuse, and Sharing Characteristics of I/O-Intensive Files on Large-Scale Production HPC Systems[C]// 18th Conference on File and Storage Technologies. Association, 2020: 91-101.
[41] JEONG K, DUFFY C, KIM J, et al. Optimizing the Ceph Distributed File System for High Performance Computing[C]// 2019 27th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP). IEEE, 2019: 446-451.
[42] ZHAN L, FANG X, LI D, et al. The research and implementation of metadata cache backup technology based on CEPH file system[C]// International Conference on Cloud Computing. IEEE, 2016: 72-77.
[43] WANG L, WEN Y C. Optimization on Small File Performance for CephFS Distributed File System[EB/OL]. https://github.com/ceph/ceph/commit/f8316f1a1a9ecdaebd870ad85159d71ba3429950.
[44] ZHAN K, XU L, YUAN Z, et al. Performance Optimization of Large Files Writes to Ceph Based on Multiple Pipelines Algorithm[C]// 2018 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications (ISPA/IUCC/BDCloud/SocialCom/SustainCom). IEEE, 2018: 525-532.

ZHANG Xiao, born in 1978, Ph.D, is a member of China Computer Federation. His main research interests include storage systems, computer networks and distributed file systems.