第 11 卷 第 3 期
2022 年 5 月
集 成 技 术
JOURNAL OF INTEGRATION TECHNOLOGY
Vol. 11 No. 3
May 2022

引文格式：
王海涛, 李战怀, 张晓, 等. 基于非易失性存储器的存储引擎性能优化 [J]. 集成技术, 2022, 11(3): 56-70.
Wang HT, Li ZH, Zhang X, et al. Performance optimization of storage engine based on non-volatile memory [J]. Journal of Integration Technology, 2022, 11(3): 56-70.

基于非易失性存储器的存储引擎性能优化

王海涛[1,2,3] 李战怀[1,2,3] 张 晓[1,2,3*] 赵晓南[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](空天地海一体化大数据应用技术国家工程实验室 西安 710072)

摘 要 非易失性存储器具有接近内存的读写速度，可利用其替换传统的存储设备，从而提升存储引擎的性能。但是，传统的存储引擎通常使用通用块接口读写数据，导致了较长的 I/O 软件栈，增加了软件层的读写延迟，进而限制了非易失性存储器的性能优势。针对这一问题，该文以 Ceph 大数据存储系统为基础，研究设计了基于非易失性存储器的新型存储引擎 NVMStore，通过内存映射的方式访问存储设备，根据非易失性存储器的字节可寻址和数据持久化特性，优化数据读写流程，从而减小数据写放大以及软件栈的开销。实验结果表明，与使用非易失性存储器的传统存储引擎相比，NVMStore 能够显著提升 Ceph 的小块数据读写性能。

关键词 非易失性存储器；存储引擎；软件栈；性能优化
中图分类号 TP 311 **文献标志码** A doi: 10.12146/j.issn.2095-3135.20210913001

Performance Optimization of Storage Engine Based on Non-Volatile Memory

WANG Haitao[1,2,3] LI Zhanhuai[1,2,3] ZHANG Xiao[1,2,3*] ZHAO Xiaonan[1,2,3]

[1](*School of Computer Science*, *Northwestern Polytechnical University*, *Xi'an* 710129, *China*)
[2](*Key Laboratory of Big Data Storage and Management*, *Ministry of Industry and Information Technology*, *Northwestern Polytechnical University*, *Xi'an* 710129, *China*)
[3](*National Engineering Laboratory for Integrated Aero-Space-Ground-Ocean Big Data Application Technology*, *Xi'an* 710072, *China*)
[*]*Corresponding Author*: *zhangxiao@nwpu.edu.cn*

Abstract Non-volatile memory has a read/write speed that is comparable to dynamic random access

收稿日期：2021-09-13 **修回日期**：2021-10-12
基金项目：国家重点研发计划项目(2018YFB1004401)；国家自然科学基金青年项目(61502392)；北京市自然科学基金-海淀原始创新联合基金项目(L192027)；陕西省重点产业链项目(2021ZDLGY03-02, 2021ZDLGY03-08)；国家自然科学基金重大集成项目(92152301)
作者简介：王海涛，博士研究生，研究方向为存储系统性能优化；李战怀，教授，研究方向为大数据管理技术以及数据库系统；张晓(通讯作者)，副教授，研究方向为云存储系统以及文件系统，E-mail：zhangxiao@nwpu.edu.cn；赵晓南，副教授，研究方向为存储系统以及分布式文件系统。

memory and can be used to replace traditional storage devices to improve the performance of storage engines. However, existing storage engines typically use generic block interfaces to access devices, resulting in a long I/O software stack, increasing read/write latency at software layers, thereby limiting the performance benefits of non-volatile memory. To solve this problem, this paper proposes a new storage engine, named NVMStore, which is based on non-volatile memory and the Ceph big-data storage system platform. NVMStore accesses storage devices through memory mapping and optimizes data read/write processes according to byte-addressability and data persistence characteristics of non-volatile memory, thus reducing the data write amplification and software stack overhead. Experimental results on real non-volatile memory devices show that NVMStore can significantly improve the performance of Ceph when dealing with small block data read/write workloads, compared with traditional storage engines using non-volatile memory.

Keywords　non-volatile memory; storage engine; software stack; performance optimization

Funding　This work is supported by National Key Research and Development Program of China (2018YFB1004401), National Natural Science Foundation of China Youth Program (61502392), Beijing Natural Science Foundation-Haidian District Joint Fund for Original Innovation Project (L192027), Shaanxi Key Research and Development Program (2021ZDLGY03-02, 2021ZDLGY03-08), and Key Integration Program of National Natural Science Foundation of China (92152301)

1　引　言

目前，大数据及其分析成为了科学研究和商业运行的核心[1-5]。然而，在过去十年，互联网平均每秒产生的数据量超过 30 TB，且增长速度持续加快[6-7]，这对大数据存储系统造成了巨大的压力。为了应对该挑战，分布式存储系统将数据平均分布在多个节点上，利用并行的数据访问，扩展存储系统的容量及性能，但数据访问量的持续增加仍导致存储系统的性能压力越来越大。为了进一步提升分布式存储系统的性能，需要减小节点间的数据传输开销，提高节点上的数据存储性能。随着高速网络在存储集群中的应用，数据传输开销可大幅降低，因此，提高节点上的数据存储性能成为关键。

分布式存储系统一般基于通用文件系统（如 Ext4）构建存储引擎，负责节点上的数据存储与管理，其优势是能够节约开发成本，保持较强的通用性。为了保证数据一致性，存储引擎在进行写入操作时，需要先将数据写入预写日志（Write Ahead Log，WAL）文件，并将其持久化到存储设备层，然后再将数据更新到目标文件中。而本地文件系统也会利用自身的日志机制保证数据一致性，因此造成了双重日志（Journaling of Journal）[8]的问题，引起了数据写放大，进而降低了系统的写性能。

高性能非易失性存储器（Non-Volatile Memory，NVM）的出现为存储引擎的性能优化提供了新的机会。NVM 也被称为存储级内存（Storage Class Memory，SCM）或持久性内存（Persistent Memory，PM），其具有类似动态随机存取存储器（Dynamic Random Access Memory，DRAM）的字节寻址能力与类似闪存的数据持久性，有望弥合计算机内外存之间的性能鸿沟[9-11]。近年来，工业界和学术界对多种 NVM 进行研究，包括相变存储器（Phase Change Memory，

PCM)、磁性随机访问存储器(Magnetic Random Access Memory，MRAM)、自旋传递扭矩MRAM(Spin Transfer Torque MRAM，STT-MRAM)和电阻随机访问存储器(Resistive Random Access Memory，ReRAM)等[12]。由于NVM具有极大地提升存储系统性能的潜力，因此目前有大量工作研究如何在现有系统中利用NVM进行性能优化，如结合NVM特性优化传统索引结构[13]，提升物联网终端设备的存储性能[14]；作为分布式文件系统的缓存扩展[15]，以及配合DRAM和固态硬盘(Solid State Disk，SSD)构建混合存储架构，优化系统整体性能[16-18]。

在不改变现有系统的前提下，可将存储引擎底层的存储设备替换为NVM，从而提高系统的读写性能。但是，对于分布式存储系统而言，上层应用的数据需要经过元数据查询、数据分发以及存储引擎的处理才能持久化到底层存储设备，这些软件层次构成了系统整体的I/O软件栈。然而，当上层数据到达存储引擎后，需要先在存储引擎的缓存中进行处理，然后通过文件系统以及块设备接口层等软件栈逐层传输和处理，最后才能到达存储设备。数据在上述I/O路径上的传输和处理会增加延迟，本文称其为I/O软件栈开销。在传统存储引擎中，除I/O软件栈开销外，I/O路径中的数据写放大也会限制NVM设备的性能发挥，因此，简单地替换底层存储设备并不能有效提高系统存储性能。如被业界广泛使用的大数据存储系统Ceph，其存储引擎的I/O软件栈较长，且写入流程会造成较为严重的数据写放大[19-20]，这会进一步增加写操作延迟，限制存储性能的扩展。

针对上述问题，本文以Ceph为平台，对存储引擎的主要性能问题进行分析，并提出了一种基于NVM的新型存储引擎NVMStore。NVMStore的主要创新点是利用NVM的字节可寻址与数据持久性，通过内存直接访问的方式读写NVM设备，并对数据读写流程进行改进，避免了数据在缓存中的拷贝以及小块数据更新造成的“读-改-写”操作，减小了I/O软件栈开销，从而提升了存储性能。

2 Ceph存储引擎分析

目前，Ceph官方提供了FileStore和BlueStore两种成熟的存储引擎，以及正在开发的一种新存储引擎SeaStore，该引擎主要面向高速的SSD设备以及高速网络，尚未正式发布。现有关于Ceph的研究工作主要是在特定场景下的集群配置参数优化，使用SSD等快速存储设备来提高集群性能[21]，优化集群的分布式锁管理机制[22]以及故障恢复流程[23]，或基于开放通道SSD的特性设计新型存储引擎以提升存储性能[24]，但是尚未出现基于NVM的Ceph存储引擎。对于Ceph存储引擎的性能问题，相关文献已进行较为充分的研究[19,24]，本节在此基础上作简要分析。

对于Ceph的FileStore，每次数据写操作会转换为一次写日志和一次写对象，因此，实际向磁盘写入的数据量至少为用户写入数据的两倍。加之对象数据更新引起的底层文件系统更新，会使实际的写放大更严重，从而造成大幅度的性能退化。此外，FileStore的日志机制基于本地文件系统实现，然而，本地文件系统也具有日志机制，这两层日志功能存在一定程度的重合，加剧了数据写放大。在这种情况下，FileStore的数据写放大(即实际写入底层存储设备的数据量与用户写入操作原始数据量之间的比值)可达14.56，导致了严重的性能退化[19]。BlueStore可有效减小数据写放大，在很大程度上解决FileStore的双重日志问题，因而成为Ceph的默认存储引擎，70%的用户选择在生产环境中使用BlueStore[20]。

如图 1 所示，在用户态下，BlueStore 使用 Linux 异步 I/O 机制直接操作块设备，避免了通用文件系统的复杂软件栈带来的延迟，利用键值存储系统 RocksDB 管理元数据和小块数据，并使用一个简化的用户态文件系统 BlueFS 管理 RocksDB 的数据。BlueFS 使用 Linux 直接 I/O 接口管理块设备，只支持 RocksDB 需要的读写接口，去除了通用文件系统中的复杂功能和属性，进一步减少了文件系统带来的性能开销。

BlueStore 针对不同存储设备设置了不同的空间分配单元(即 bluestore_min_alloc_size 配置项)，对于旋转型存储设备(如 HDD)默认使用 64 KB 的分配单元，以发挥其顺序访问性能优势。对于非旋转型存储设备(如 SDD)，则默认使用 4 KB 的分配单元，主要原因是此类设备的随机访问性能与顺序访问性能的差距较小，设置 4 KB 的分配单元即可降低小块数据更新时的“读-改-写”操作负载，进而提升存储性能。设备上的读写操作通过分配单元进行切分，从而确定需要操作的数据区域。当 BlueStore 接收到用户的写操作请求时，首先根据目标对象的元数据定位目标区域，然后再针对各个区域进行写入操作。在 BlueStore 中，写操作整体上可分为两种：

(1) 新写：当写入数据的目标位置是新分配的空间时，直接在新空间中写入数据(步骤②～③)，然后通过 RocksDB 更新相应的元数据(步骤④)，写操作完成。由于写操作不覆盖已有数据，因此，即使在写入过程中发生崩溃，也不会破坏数据一致性。

(2) 覆盖写：若写入的数据量大于分配单元，则首先将其分割，将达到分配单元的部分按照新写方式处理，未达到分配单元的部分则写入 RocksDB 中(步骤④)，完成用户写操作。最后，RocksDB 通过异步 I/O 的方式把数据写入到目标位置，并更新相应的元数据(步骤⑥～⑦)，由 RocksDB 的事务机制保证数据一致性。

由上述写流程可知，BlueStore 基本解决了由 FileStore 日志造成的数据写放大问题。首先，BlueStore 绕开了通用文件系统，直接管理块设备，因此，不受通用文件系统日志的影响。其次，新写操作不需使用日志，只需一次数据落盘，一次元数据通过 RocksDB 事务提交落盘。最后，小块的覆盖写操作可通过 RocksDB 合并

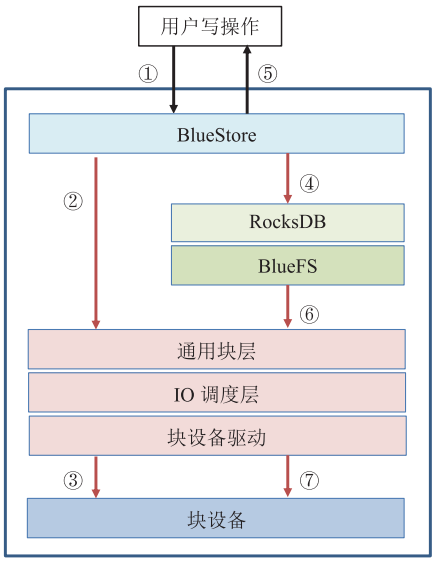


图 1　BlueStore 架构以及写入流程

Fig. 1　The architecture and write process of BlueStore

到日志中进行提交，日志提交完成即可通知用户写完成，再将数据写入到目标位置。经上述优化后，BlueStore 减小了 FileStore 的数据写放大，提高了存储引擎层的性能。

但是，当 BlueStore 使用 RocksDB 处理小块数据（包括元数据）时，RocksDB 的日志机制造成了较为严重的写放大，如进行 4 KB 的随机写时，写放大最高可达 50 倍[19]。此外，BlueStore 与 FileStore 都是通过 Linux 系统的通用块接口来访问存储设备，导致了较长的 I/O 软件栈。特别是对于字节可寻址设备（如 NVM）而言，冗长的 I/O 软件栈可能会成为其性能瓶颈[25]。据英特尔在 2018 年闪存峰会上公布的测试数据，当使用 NVM 作为存储设备时，软件栈造成的延迟可达总延迟的 90%。因此，需进一步研究如何优化存储引擎的架构以及 I/O 软件栈，配合底层存储设备的特性来提升读写性能。

3 新型存储引擎设计与实现

本节提出了一种基于 NVM 的新型存储引擎 NVMStore——通过内存映射而非传统块接口的方式访问 NVM 设备，并根据 NVM 的特性优化数据读写流程，从而减小数据读写放大以及 I/O 软件栈开销，进一步提高读写性能。

3.1 NVMStore 架构

图 2 为 NVMStore 的架构以及存储结构。与使用通用块接口的 BlueStore 相比，NVMStore 具有更短的 I/O 路径，并且通过内存直接访问（Direct Access，DAX）的方式可绕过系统缓存直接读写 NVM 设备，利用缓存刷新指令（如 clflush）将 CPU 缓存行中的数据直接持久化到 NVM 中，避免数据在内存中的重复拷贝，从而减小软件栈的开销。DAX 的核心思想是绕过操作系统的页缓存直接访问设备，支持 DAX 的文件系统（如 Ext4）只提供内存映射接口，具体的数据则由 DAX 接口之上的用户程序来管理。用户程序通过定制化的数据结构和读写流程来管理 NVM 的空间和数据，不会由于绕过传统文件系统而影响其功能，而且避免了传统文件系统引入的性能开销，可获得一定的性能提升。

将 NVM 作为底层存储设备的主要挑战是原子粒度的不匹配，即 CPU 和 NVM 之间的数据传输粒度是 CPU 缓存行（通常为 64 字节），而 NVM 设备的读写单元是内存数据位宽（通常为 8 字节）。在 NVM 上，大于 8 字节的数据写入会被分割为多个写入单元，因此，若在数据覆盖写的过程中系统崩溃，则可能造成只有部分新数据写入成功，从而破坏数据一致性。针对该问题，NVMStore 构建了一个事务管理模块，通过事务

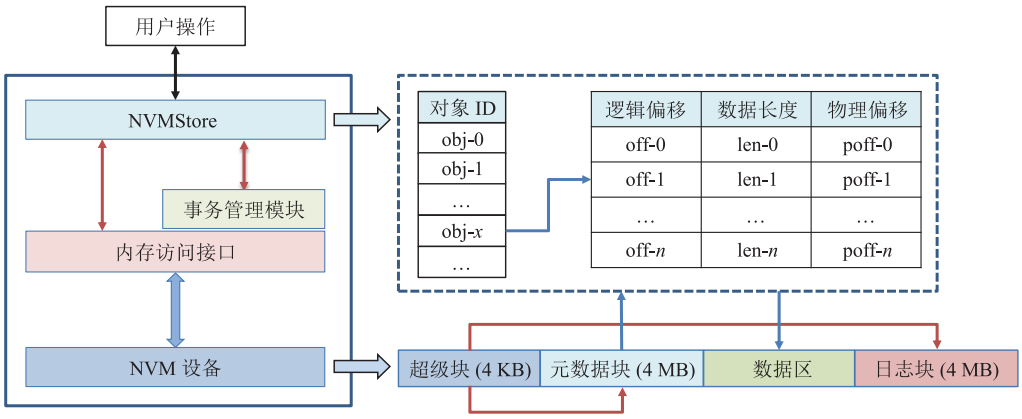


图 2 NVMStore 架构以及存储结构

Fig. 2 The architecture and storage structure of NVMStore

的形式进行覆盖写操作，当系统发生崩溃重启时，通过重放日志中的有效事务记录来恢复数据一致性(详见第 3.3 节)。

如图 2 所示，NVMStore 将其管理的 NVM 存储空间分为超级块(4 KB)、元数据块(4 MB)、数据区以及日志块(4 MB)。其中，超级块位于空间起始位置，用于存放系统标签、元数据块地址以及日志块地址等系统元数据。超级块之后是第一个元数据块，用于记录系统中各个对象的 ID 以及对应的数据区域(即数据在对象中的逻辑偏移、数据长度以及对应到存储设备上的物理偏移等)。元数据块以 4 MB 为单位进行分配，当一个元数据块写满时，可在数据区中申请新块，并在超级块中进行相应的记录。由于元数据块的大小是固定的，因此当新增一个元数据块时，只需在超级块的元数据列表中新增一个 8 字节指针，指针指向新增元数据块的起始地址，该架构利用 NVM 的 8 字节原子性保证了新增元数据块操作的原子性。日志块以 4 MB 为单位进行分配，用于支持事务管理模块。简而言之，事务数据先在写入日志块中追加进行持久化存储，然后再将更新的数据写入目标空间中，从而保证系统的崩溃一致性。当需要新增日志块时，先从数据区分配一块空间，然后在超级块的日志列表中新增一条记录，方法与元数据块相同。不同的是，当日志中的事务成功提交到系统，将其数据持久化到设备后，相应的日志即可标记为废弃(废弃标记为 8 字节，保证原子性)，日志占用的空间即可回收。

NVMStore 不改变现有的存储服务接口，其底层的存储流程对用户是透明的，不会影响上层用户的使用，因此，不需修改用户程序便可使用新的存储引擎来提高存储性能。NVMStore 是基于 Ceph 大数据存储系统设计的存储引擎，因此只支持 Ceph 存储后端需要的功能，不提供服务给通用的文件系统。在实现 Ceph 后端存储功能的前提下，NVMStore 的设计简单，便于实现和维护。

3.2　数据写入流程

NVMStore 读取流程与写入流程类似，但若目标数据在 CPU 缓存中命中，则无须访问 NVM 设备。NVMStore 与 BlueStore 的写操作流程类似，主要区别在于 NVMStore 通过内存接口访问 NVM 设备，且利用 NVM 的字节可寻址特性对数据写入流程进行优化。当用户写操作到达 NVMStore 时，NVMStore 首先根据对象元数据找到与写入数据块对应的存储区域，然后按照存储空间分配单元(默认等于内存页面大小，即 4 KB)切分写操作，再根据写操作的类型执行不同的写入流程，总体上也可分为新写和覆盖写两大类型，具体分析如下：

(1) 新写

对于新写的数据块，直接在设备上分配新存储单元并写入数据，然后利用缓存刷新指令将数据持久化到 NVM 设备，最后通过事务管理模块更新相应的元数据，完成写操作。

BlueStore 使用传统的块接口写入数据，为了与设备上的块单元对齐，需要进行数据补 0 操作，从而造成一定的数据写放大。而 NVMStore 利用 NVM 的字节可寻址特性，无须额外的数据补 0 操作，可直接将小块数据写入新空间。因此，NVMStore 的写入流程更简单，且减小了数据写放大。

BlueStore 通过 RocksDB 写入元数据，而 RocksDB 具有较长的 I/O 软件栈，其数据压缩流程会造成严重的写放大。NVMStore 则使用一个简单的事务管理模块更新元数据，通过内存接口将数据持久化到 NVM 设备，避免了 RocksDB 的多层数据压缩，从而减小了数据写放大，并缩短了 I/O 软件栈。

(2) 覆盖写

如图 3 所示，NVMStore 在写入对象 A 时，首先将写操作按照分配单元进行切分，对达到分配单元的整块覆盖写，将其直接写入新分配的

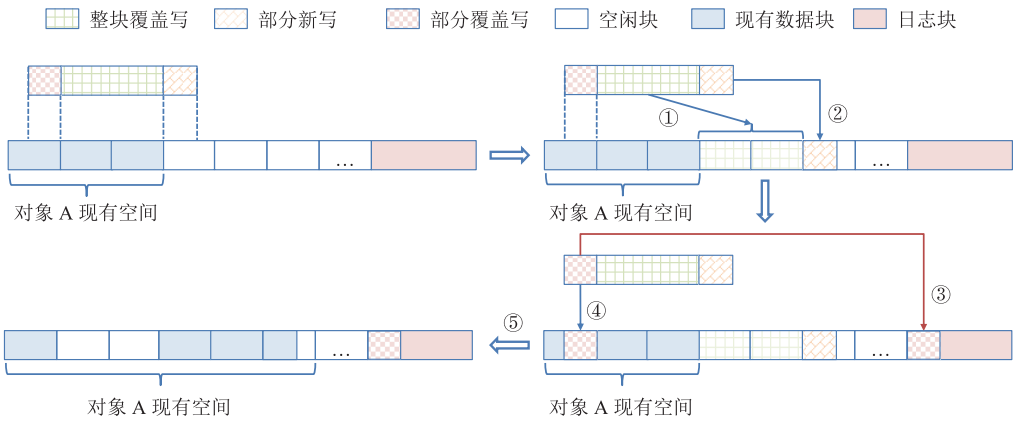


图 3 覆盖写操作示例

Fig. 3 Overwrite example

空间(步骤①)。对不足分配单元的新写(称为部分新写)，则直接分配一个新的单元并写入(步骤②)。对不足一块的覆盖写(称为部分覆盖写)，则先将数据写入日志块，并持久化到 NVM 设备(步骤③)，然后将数据更新到目标区域(步骤④)，最后以事务的方式更新对象 A 的元数据，完成写操作(步骤⑤)。

与 BlueStore 的写入流程相比，NVMStore 的写入流程更为简单。且由于 NVM 具有字节可寻址特性，NVMStore 将数据更新到目标区域时，不必对目标数据块进行“读-改-写”操作，就可直接将数据写入指定位置(如步骤②和④)，因此，能减小数据读写放大，特别是小块数据操作。由于 CPU 访问 NVM 的粒度为缓存行，所以在读取小块数据时，会直接读取数据所在的整个缓存行空间，造成一定程度的数据读放大。但与传统存储设备上的预读操作一样，这种数据预读操作可利用数据访问的空间局部性，提高缓存命中率，减小访问存储设备的频率，通常能够有效提升用户的数据读取性能。为了更好地发挥缓存性能，NVMStore 为对象分配新空间时，会优先选择其现有空间附近的空闲空间，从而提高对象数据访问的空间局部性。

综上所述，与 BlueStore 相比，NVMStore 减小的性能开销主要包含两部分：一是通过 DAX 的方式访问设备，数据直接从 CPU 写入 NVM，避免了缓存中的拷贝开销；二是 NVMStore 通过内存接口读写数据，当写入小块数据时，避免了由块接口的“读-改-写”操作导致的开销。

3.3 崩溃一致性保证

NVMStore 通过事务管理模块保证系统中数据的崩溃一致性。对于整块覆盖写或新写(包括部分新写)操作，NVMStore 直接分配新的空间写入数据，当数据持久化成功后，再通过事务管理模块更新元数据。在此过程中，如果系统崩溃导致数据持久化失败，那么由于原有的数据并没有受到破坏，不影响数据一致性。如果数据持久化成功，但元数据更新失败，那么元数据依然指向原有的数据块(由事务管理模块保证)，也不会破坏数据一致性。

对于元数据的更新以及部分覆盖写操作，事务管理模块利用日志机制保证其数据的一致性。图 4 为事务管理模块的写入流程，如果写操作的数据不大于 8 字节(如设置日志块废弃标记)，那么可直接将其写入目标区域(步骤①)，然后将目标数据持久化到 NVM 设备(步骤⑧)，利用 NVM 设备本身的 8 字节原子性保证写入操作的原子性，只需一次持久化操作。如果写操作的数

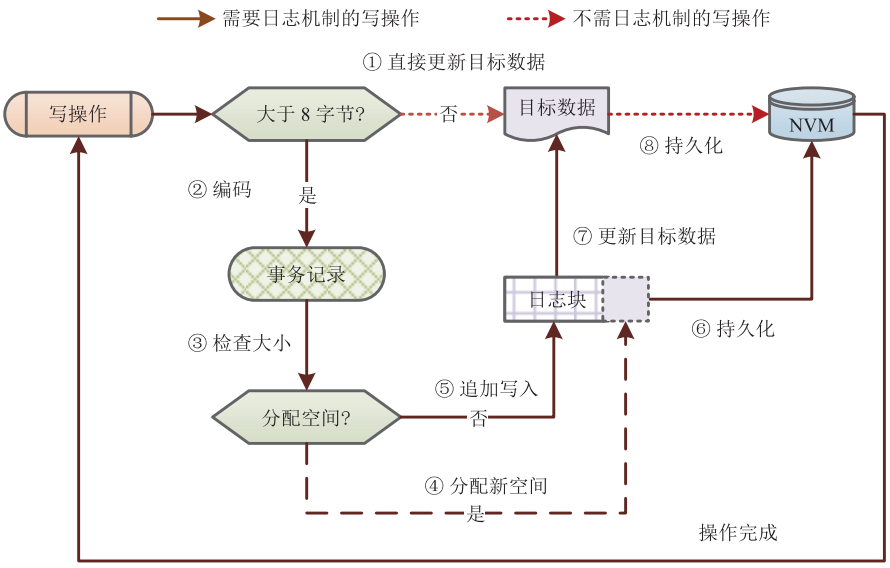


图 4　事务管理模块写入流程

Fig. 4　The write process of transaction management module

据量大于 8 字节，那么 NVM 本身的硬件特性将无法保证写操作的原子性，因此需要通过日志机制来支持数据一致性，需进行两次持久化操作。

当写操作的数据量大于 8 字节时，事务管理模块首先将写操作编码为特定格式的事务记录(步骤②，编码数据包括写操作的目标地址、数据量、数据以及校验码等)，然后判断目前的日志块是否有足够的空闲空间写入当前事务记录(步骤③)。若当前日志块中有足够的空闲空间，则直接写入事务记录并将其持久化(步骤⑤～⑥)；否则分配一个新的日志块(步骤④)，将事务记录追加到日志块中(步骤⑤)，然后将其持久化(步骤⑥)。在此之后，更新目标数据并持久化(步骤⑦～⑧)，完成写入操作。其中，步骤⑥和⑧分别用于持久化目标数据和日志数据，二者利用 NVM 的内存操作接口，通过缓存刷新指令(如 clflush)，从 CPU 缓存中将数据直接写入设备中。

在此过程中，若在写入日志完成前系统崩溃，则认为本次操作失败，由于原有的数据尚未更新，因此不破坏数据一致性。若写入日志成功，但是在更新目标数据时系统崩溃，则在重启时扫描最后更新的日志块，重放其中的有效事务记录，从而恢复数据一致性。综上所述，NVMStore 通过事务管理模块保证了系统的崩溃一致性。

4　实验评估与分析

本文结合英特尔开源的 NVM 开发库 PMDK[1](Persistent Memory Development Kit)，以及 Ceph 的对象存储(Object Store)抽象接口实现 NVMStore。具体实现过程中，Ceph 源码是 14.2.8 版，PMDK 是 1.10 版，NVM 设备是 Intel Optane DC Persistent Memory Module(简称“Optane”)。

Optane 为上层应用提供两种访问模式——内存模式(Memory Mode)和应用直接访问模式(App

注[1]：PMDK 开发库为 NVM 上的空间管理和数据操作提供方便的接口，项目网址：https://github.com/pmem/pmdk/。

Direct Mode)。在内存模式下，CPU 和操作系统忽略 Optane 的数据持久性，而将其视为 DRAM 空间的扩展，从而提高系统的内存容量。在应用直接访问模式下，CPU 和操作系统将 Optane 视为独立的存储设备，用户可跳过 DRAM 缓存，通过内存接口直接访问，在实现数据持久性存储的同时，避免了数据在 DRAM 中的拷贝开销。为了实现数据持久性存储和减小 I/O 软件栈的开销，NVMStore 采用了直接访问模式来读写 Optane。

由于目前成熟的 Ceph 的后端存储引擎只有 FileStore 和 BlueStore，尚未发布基于 NVM 的存储引擎，因此，本实验对使用 NVM 作为存储设备的 FileStore、BlueStore 和 NVMStore 的读写性能进行了对比。

4.1 实验配置与评估方法

理论上，NVM 具有远超 HDD 的读写速度，特别是在小块数据的随机读写方面，因此利用 NVM 替换存储系统中的 HDD，能够提高存储系统的性能。为了验证该理论，本实验对 HDD 与 NVM 设备的读写性能进行比较，得出二者的性能基准。并将 NVM 替换 HDD 设备后，对 FileStore 和 BlueStore 的性能提升效果进行对比，验证了本文优化方法的有效性。

实验中使用 4 台配置相同的 Linux 服务器搭建实验平台。其中，3 台搭建 Ceph 集群，每个节点配置 1 个 HDD 设备和 1 个 NVM 设备。另外 1 台作为性能测试的客户端，各个节点之间使用万兆以太网连接，具体实验配置如表 1 所示。测试工具是业界广泛使用的 Fio，可直接对块设备进行测试，获得块设备的吞吐量、IOPS 以及读写延迟等性能指标。实验对顺序读、顺序写、随机读和随机写这 4 种基准负载模式进行测试。每次读写的 I/O 块大小分别设置为 1 KB、4 KB、16 KB、64 KB、256 KB、1 MB 和 4 MB。当读写测试时间达到 10 min 或数据量达到 100 GB 时，测试停止，测试 5 次取性能平均值，然后进行对比分析。

表 1 实验配置

Table 1 Experimental configurations

软硬件	配置参数
CPU	2 个 16 核 2.30 GHz Intel(R) Xeon(R) Gold 5218 处理器
内存	80 GB DDR3 内存模块，1 333 MHz
硬盘	西部数据 WD Black 硬盘，1 TB，7 200 转
NVM	Intel Optane DC Persistent Memory Module，128 GB
网卡	Intel Corporation Ethernet Controller 10-Gigabit X540-AT2
Ceph 版本	Ceph v14.2.8 OCTOPUS
测试工具	Fio-3.16
操作系统	64 位 Ubuntu 20.04 LTS，Linux 4.4.0-58-generic 内核

如第 2 节所述，BlueStore 对非旋转型的存储设备默认使用 4 KB 的分配单元，以提升存储性能，NVM 设备也属于非旋转型的存储设备，因此，为更合理地对比性能，本文在进行 NVM 性能测试时，将 BlueStore 的分配单元设置为 4 KB（通过 BlueStore 的日志可验证），与 NVMStore 上的分配单元保持一致。

4.2 块设备性能对比

本节对比了 HDD 与 Optane 作为块设备的数据读写性能，为了达到块设备的性能峰值，实验中的 Fio 测试引擎设置为异步 I/O，队列深度设置为 32，并绕过系统页缓存和块缓存直接对块设备进行读写测试。实验对比的性能指标为吞吐量（MB/s），HDD 的性能数据如表 2 所示，Optane 的性能数据如表 3 所示，Optane 与 HDD 的性能比值如表 4 所示。

由实验数据可知，Optane 设备的吞吐量远高于 HDD，如 Optane 的 1 KB 随机写性能可达 HDD 的 5 243.67 倍，4 MB 顺序写性能可达 HDD 的 5.94 倍（见表 4）。由于 HDD 机械部件导致的延迟，其随机性能远低于顺序性能，特别是当 I/O 块较小时，对比更为明显。而 Optane 设备没有机械部件，其读写性能主要取决于数据的分发是否能充分利用底层存储器件的并发性。如表 3 所

示，虽然 Optane 设备的读性能高于写性能，顺序性能高于随机性能，但是二者之间的性能差距远小于 HDD 的性能差距。此外，Optane 的吞吐量随着 I/O 块的增大而提高，且当 I/O 块大小约为 64 KB 时达到峰值，这主要是由 Optane 底层存储介质的特性及其内部存储单元的分布决定的。

表 2　HDD 吞吐量

Table 2　Throughput of HDD

块大小	不同读写模式的吞吐量(MB/s)			
	顺序读	顺序写	随机读	随机写
1 KB	16.63	11.80	0.12	0.06
4 KB	62.98	51.19	0.47	1.04
16 KB	123.89	116.41	1.87	3.99
64 KB	124.57	118.03	7.12	14.77
256 KB	124.16	118.92	24.21	42.08
1 MB	124.7	118.32	61.28	86.2
4 MB	124.58	118.42	97.81	92.41

表 3　Optane 吞吐量

Table 3　Throughput of Optane

块大小	不同读写模式的吞吐量(MB/s)			
	顺序读	顺序写	随机读	随机写
1 KB	516.66	345.18	342.33	314.62
4 KB	1 209.56	526.34	975.85	524.98
16 KB	1 255.84	686.15	1 184.49	675.34
64 KB	1 292.77	692.29	1 267.61	689.16
256 KB	1 297.30	713.30	1 290.53	712.79
1 MB	1 287.87	704.57	1 282.00	726.64
4 MB	1 294.93	703.17	1 288.00	703.72

表 4　Optane 与 HDD 的性能比值

Table 4　Performance ratio of Optane to HDD

块大小	不同读写模式的性能比值			
	顺序读	顺序写	随机读	随机写
1 KB	31.07	29.25	2 852.75	5 243.67
4 KB	19.21	10.28	2 076.28	504.79
16 KB	10.14	5.89	633.42	169.26
64 KB	10.38	5.87	178.04	46.66
256 KB	10.45	6.00	53.31	16.94
1 MB	10.33	5.96	20.92	8.43
4 MB	10.39	5.94	13.17	7.62

4.3　块接口与内存接口性能对比

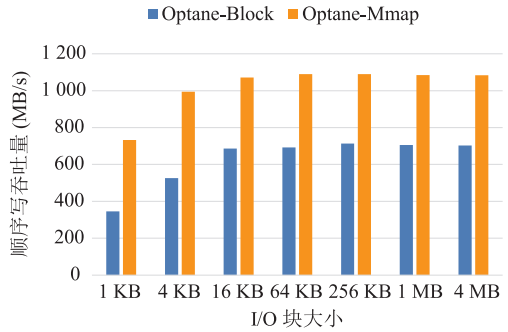
本小节对使用内存映射的方法读写 Optane 时的性能优势进行验证。由于目前 Fio 不支持 Optane 的内存接口测试，因此本小节通过 PMDK 提供的内存读写接口进行了测试，实验结果如图 5 所示，其中，Optane-Block 表示使用传统的块接口读写 Optane，Optane-Mmap 表示使用内存映射的方式读写 Optane。

由图 5 可知，Optane-Mmap 的整体性能高于 Optane-Block，特别是小块数据读写。由图 5(b)可知，当 I/O 块大小为 1 KB 时，Optane-Mmap 的性能约为 Optane-Block 的 2.5 倍。原因是 Optane 具有字节可寻址特性，可通过内存接口直接访问设备，避免了块接口较长的 I/O 软件栈及数据在内存中的不必要拷贝，从而可获得较高的性能。因此，与传统的块接口相比，本文提出的 NVMStore 通过内存接口访问 NVM 设备，理论上具有更优的性能。

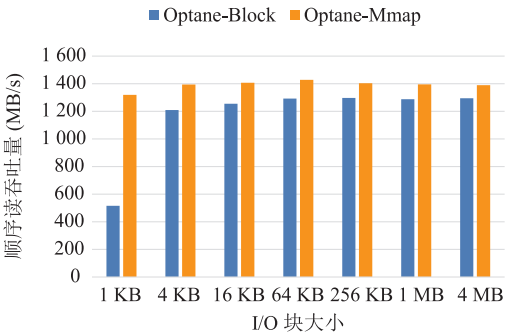
4.4　存储引擎性能提升效果

本节通过实验分析，当使用 Optane 设备替换 HDD 时，不同存储引擎的 Ceph 获得的性能提升效果。由于 Ceph 的块接口的实际应用较为广泛，因此，本节针对 Ceph 的 Rados 块设备(Rados Block Device，RBD)进行测试。测试方法是，在 Ceph 集群中创建一个大小为 100 GB 的 RBD 镜像，然后使用 Fio 的 RBD 引擎对该镜像进行读写性能测试。为更直观地展示性能提升效果，根据存储设备为 HDD 的 FileStore(FileStore-HDD)的吞吐量(MB/s)，将性能数据进行标准化处理。最终对 Optane 的 FileStore(FileStore-Optane)和 BlueStore(BlueStore-Optane)相对于 FileStore-HDD 的性能提升倍数进行对比。图 6 为性能提升效果，FileStore-HDD 的性能值标准化为 1，不在图中展示。

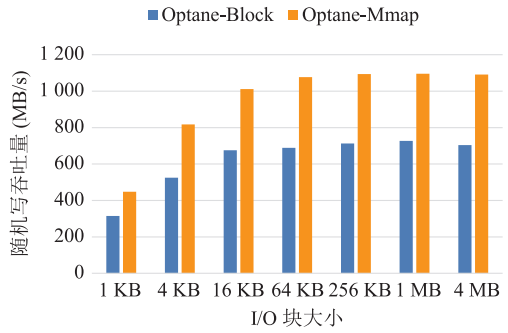
由图 6 可知，小块随机写性能的提升最为明显，且 BlueStore-Optane 的性能更高。当 I/O 块



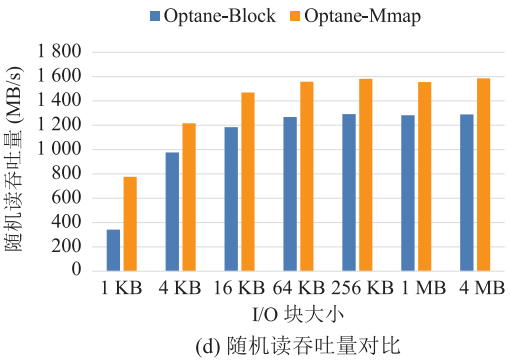
(a) 顺序写吞吐量对比



(b) 顺序读吞吐量对比



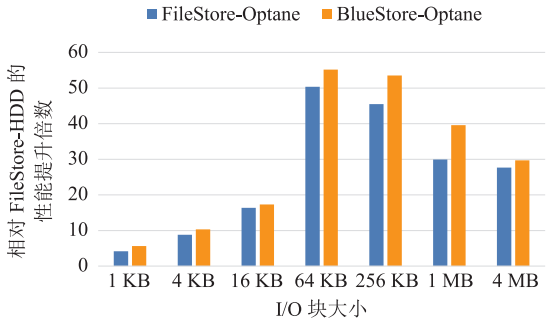
(c) 随机写吞吐量对比



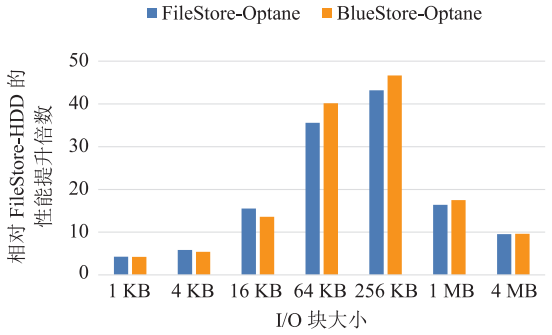
(d) 随机读吞吐量对比

图 5 Optane 块接口与内存接口性能对比

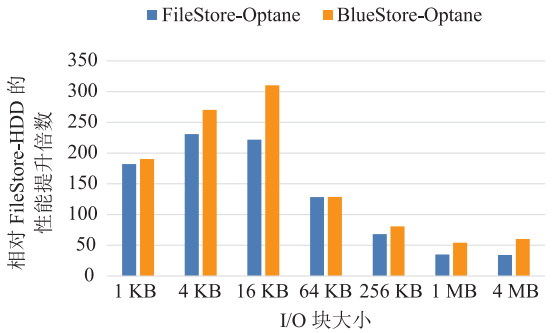
Fig. 5 Performance comparison between Optane's block and memory interface



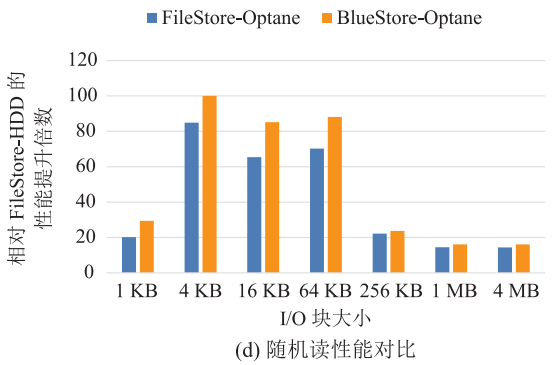
(a) 顺序写性能对比



(b) 顺序读性能对比



(c) 随机写性能对比



(d) 随机读性能对比

图 6 使用 Optane 的存储引擎性能提升效果

Fig. 6 Performance improvement of storage engine using Optane

小于 64 KB 时，与 FileStore-HDD 相比，FileStore-Optane 和 BlueStore-Optane 的随机写性能提升可达 180 倍以上。与 FileStore-Optane 相比，BlueStore-Optane 的性能更高，其原因是 FileStore 中的双重日志问题造成了数据写放大，占用了一部分 Optane 的带宽，限制了 Optane 的性能发挥。而 BlueStore 直接管理块设备，很大程度上减小了数据写放大，且 BlueStore 的 I/O 软件栈较为简单，因此，更容易发挥 Optane 的性能优势。

本实验将不同存储引擎的 Ceph 存储系统的整体性能进行对比，不能简单地将其与第 4.2 节块设备的性能对比结果一一对应。如在第 4.2 节块设备性能对比实验中，当 I/O 块为 64 KB 时，Optane 设备的顺序读性能约为 HDD 的 10 倍(详见表 4)。但是如图 6(b)所示，在同样的负载模式下，Ceph 整体的顺序读性能提高了近 40 倍。其原因是 Ceph 整体上能够将数据均匀地分布在各个存储节点上，当进行大量数据的读写操作时，能够较好地发挥各个存储节点以及存储设备的并发性，实现性能的扩展，尤其是在数据块较大的负载模式下。

然而，直接利用 Optane 替换 HDD，并不能充分发挥 Optane 的性能优势。例如，Optane 的小块随机读性能可达 HDD 的上千倍(详见表 4)，但是如图 6(d)所示，将底层存储设备换为 Optane 后，Ceph 的 1 KB 随机读性能却只提升了 30 倍左右。该情况一方面是由于集群节点之间的数据传输增加了部分延迟；另一方面则是由于 FileStore 与 BlueStore 底层都使用了传统的块接口和异步 I/O 机制，引入了较大的 I/O 软件栈开销，从而限制了 Optane 设备的性能发挥。

4.5　NVMStore 性能评估

目前，已有开源的基于 NVM 的键值存储系统(如 pmemkv[2])，但仍未有开源的可用于 Ceph 后端的 NVM 存储引擎。因此，本节对使用 Optane 设备的 NVMStore(表示为 NVMStore-Optane)与 Ceph 现有存储引擎的读写性能进行对比。由于目前 Ceph 默认使用的存储引擎是 BlueStore，且第 4.4 节实验已证明 BlueStore-Optane 的整体性能优于 FileStore-Optane，因此本实验直接对 NVMStore-Optane 与 BlueStore-Optane 的性能进行对比。实验方法为，首先在 Ceph 集群中创建一个 RBD 镜像设备，然后使用 Fio 的 RBD 引擎对该镜像进行读写测试，获得每秒执行 I/O 操作的次数(Input/Output Operations Per Second，IOPS)和吞吐量性能，性能对比结果如图 7 所示。

由图 7 可知，NVMStore-Optane 的性能优于 BlueStore-Optane，且读写负载的 I/O 块越小，NVMStore-Optane 性能优势越大。如图 7(a)所示，当 I/O 块为 1 KB 的顺序写负载时，NVMStore-Optane 的性能约为 BlueStore-Optane 的 2 倍。主要原因是，Optane 读写单块数据的硬件延迟是稳定的，但是当处理小块(小于等于 4 KB 的数据块)I/O 负载时，BlueStore 需要频繁地通过块接口访问 Optane，导致大量的软件栈延迟，且小块操作造成的数据写放大进一步增加了延迟。而 NVMStore 利用内存直接访问 Optane，减小了软件栈引入的延迟，且利用 NVM 的字节可寻址特性减小了数据写放大，因此性能提升较为明显。

为了验证该结论，在 BlueStore 与 NVMStore 的代码中增加日志输出，创建 Ceph RBD 块设备，并针对小块随机写入负载进行测试。测试方法为，使用 Fio 的 RBD 引擎进行 120 s 的 1 KB 随机写入，统计 BlueStore 与 NVMStore 的 I/O 软件栈延迟占比以及数据写放大，测试结果如表 5

注[2]：一个基于 PMDK 开发的，适用于 NVM 的开源键值存储系统，项目网址：https://github.com/pmem/pmemkv.git。

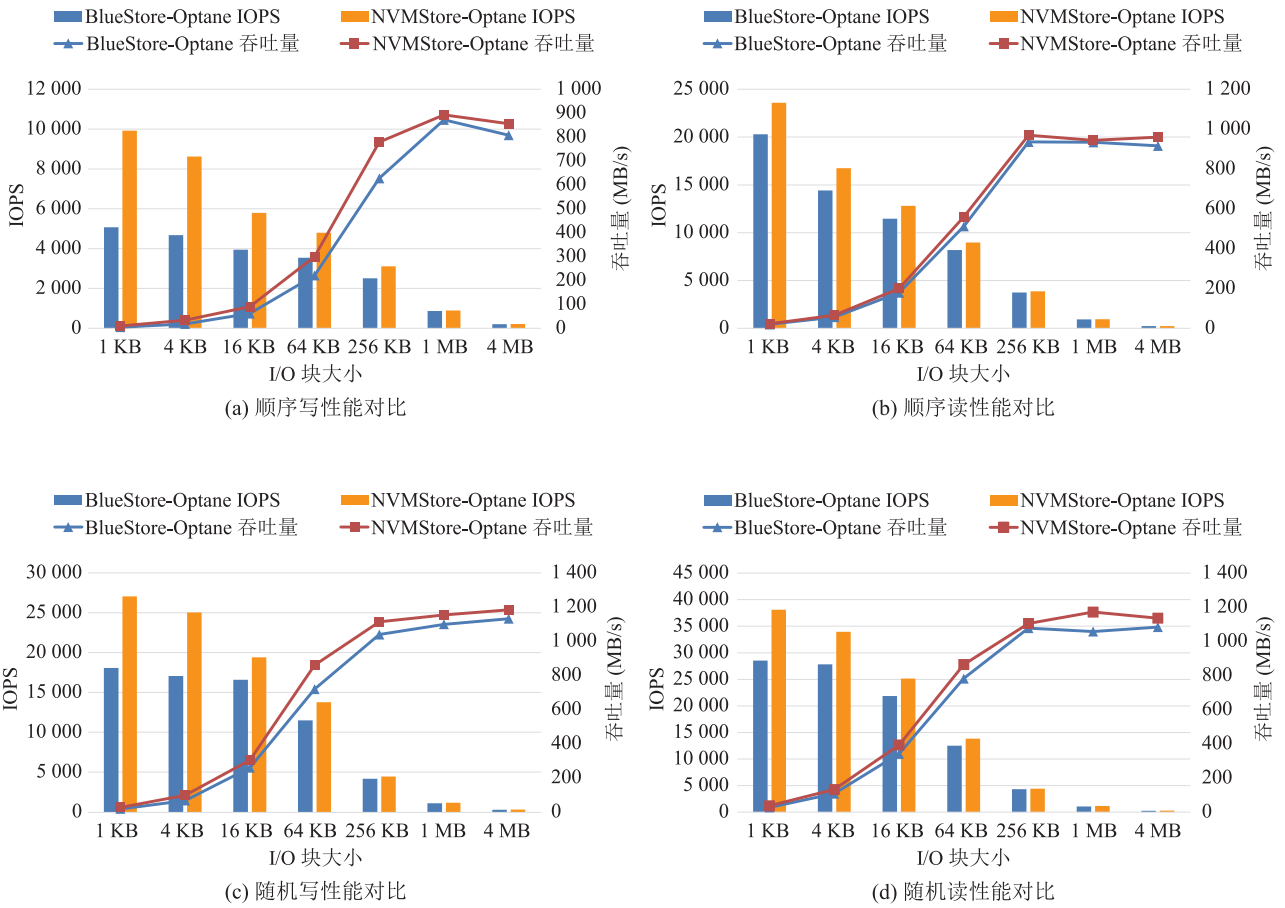


图 7 NVMStore 性能优化效果

Fig. 7 Performance improvement of NVMStore

表 5 I/O 软件栈延迟占比以及数据写放大对比

Table 5 Latency ratio of I/O software stack and data write amplification comparison

存储引擎	Fio 写入时间(s)	I/O 软件栈总延迟	Fio 写入量(MB)	设备写入量(MB)	I/O 软件栈延迟占比(%)	数据写放大
BlueStore-Optane	120	90.92	1 805	18 111	75	10.03
NVMStore-Optane	120	40.86	3 365	7 753	34	2.30

所示。实验结果表明，与 BlueStore 相比，NVMStore 的 I/O 软件栈延迟占比更小，且能够大幅度减小数据写放大，因而在小块写入时其性能提升较为明显。

当 I/O 块较小(不超过 64 KB 的数据块)时，NVMStore 的性能略高于 BlueStore。主要原因是，BlueStore 通过块接口访问设备的频率降低，I/O 软件栈的开销占总延迟的比例降低，进而缩小了与 NVMStore 的性能差距。当 I/O 块较大时(大于 64 KB 的数据块)，NVMStore-Optane 与 BlueStore-Optane 的性能相当。主要是因为，当读写数据量一定时，随着 I/O 操作的粒度增大，调用 I/O 软件栈的频率将大幅降低，从而减小了 I/O 软件栈引入的延迟，使得 Optane 的数据读写延迟占据了性能开销的主要部分。此时，NVMStore 虽然能够减小 I/O 软件栈引入的延迟，但是对整体读写操作的影响较小，从而限制了其性能提升效果。因此，NVMStore 更适

用于大量小块数据读写的应用场景。目前，在 Facebook 等流行的社交网络平台或者虚拟桌面云平台应用场景中，小块数据的随机读写负载非常普遍[19,25-27]，因此，以 NVMStore 为代表的性能优化方向具有重要的现实意义。

如图 7 所示，当 Optane 为底层存储设备时，Ceph RBD 设备的随机读写性能高于顺序读写操作性能。主要是因为，在进行大量数据读写时，客户端在 RBD 设备上的顺序操作，经数据的切分与分发，到达各个存储节点后，转化为各个存储引擎上的随机操作。与 HDD 等慢速设备相比，Optane 设备上的随机操作性能与顺序操作性能差异较小，因此，存储引擎处理随机操作不会造成严重的性能退化。相反，大量随机读写操作更容易均匀地分布到各个存储节点的多个对象中，有助于发挥存储节点以及 Optane 设备底层存储单元的并发性，最终达到更高的读写性能。

5　结　论

大数据存储系统的存储引擎对系统的整体性能至关重要。虽然使用高性能的 NVM 设备替换传统存储设备可以提高存储性能，但是并不能充分发挥 NVM 设备的性能优势。主要原因是传统的存储引擎使用块接口来访问 NVM 设备，多层次的 I/O 软件栈开销以及数据写放大造成的延迟，限制了 NVM 性能优势的发挥，进而影响了存储引擎的性能扩展。针对该问题，本文对大数据存储系统 Ceph 的存储引擎及其 I/O 软件栈进行了分析，提出了一种基于 NVM 的存储引擎 NVMStore，其利用内存直接映射的方式访问 NVM，并根据 NVM 的特性对存储引擎的数据读写流程进行优化，减小了 I/O 软件栈的开销以及数据读写放大，进一步提升了存储引擎的性能。最后在 NVM 设备上进行实验，将 NVMStore 与现有的 Ceph 默认存储引擎 BlueStore 的基准读写性能进行对比，实验结果表明，NVMStore 能够显著提高存储系统的小块数据读写性能。在实际应用场景中，由于小块数据读写负载较为普遍，因此，NVMStore 的性能优化方向具有重要的现实意义。

参 考 文 献

[1] Sagiroglu S, Sinanc D. Big data: areview [C] // Proceedings of the 2013 International Conference on Collaboration Technologies and Systems, 2013: 42-47.

[2] Gantz J, Reinsel D. Extracting value from chaos [J]. IDC Iview, 2011, 1142: 1-12.

[3] Gautam A, Chatterjee I. Big data and cloud computing: acritical review [J]. International Journal of Operations Research and Information Systems, 2020, 11(3): 19-38.

[4] Karapiperis C, Chasapi A, Angelis L, et al. The coming of age for big data in systems radiobiology, an engineering perspective [J]. Big Data, 2020, 9(1): 63-71.

[5] Nobanee H. A bibliometric review of big data in finance [J]. Big Data, 2021, 9(2): 73-78.

[6] Warren J, Marz N. Big Data: principles and best practices of scalable realtime data systems [M]. Greenwich: Manning Publications Co., 2015: 1-425.

[7] Hashem IAT, Yaqoob I, Anuar NB, et al. The rise of “big data” on cloud computing: review and open research issues [J]. Information Systems, 2015, 47: 98-115.

[8] Shen K, Park S, Zhu M. Journaling of journal is (almost) free [C] // Proceedings of the 12th USENIX Conference on File and Storage Technologies, 2014: 287-293.

[9] Burr GW, Kurdi BN, Scott JC, et al. Overview of candidate device technologies for storage-class memory [J]. IBM Journal of Research and Development, 2010, 52(4.5): 449-464.

[10] Caulfield AM, Coburn J, Mollov TI, et al. Understanding the impact of emerging non-

volatile memories on high-performance, IO-Intensive computing [C] // Proceedings of the High Performance Computing, Networking, Storage and Analysis, 2010: 1-11.

[11] 夏飞, 蒋德钧, 熊劲. 影响非易失性内存系统性能的因素分析 [J]. 计算机研究与发展, 2014, 51(S1): 25-31.
Xia F, Jiang DJ, Xiong J. Evaluating and analyzing the performance of nonvolatile memory system [J]. Journal of Computer Research and Development, 2014, 51(S1): 25-31.

[12] Mittal S, Vetter JS. A survey of software techniques for using non-volatile memories for storage and main memory systems [J]. IEEE Transactions on Parallel and Distributed Systems, 2016, 27(5): 1537-1550.

[13] Lee SK, Mohan J, Kashyap S, et al. Recipe: converting concurrent DRAM indexes to persistent-memory indexes [C] // Proceedings of the 27th ACM Symposium on Operating Systems Principles, 2019: 462-477.

[14] Liu Y, Zhao S, Chen WH, et al. NVM storage in IoT devices: opportunities and challenges [J]. Computer Systems Science and Engineering, 2021, 38(3): 393-409.

[15] Cheng W, Li CY, Zeng LF, et al. NVMM-oriented hierarchical persistent client caching for Lustre [J]. ACM Transactions on Storage, 2021, 17(1): 1-22.

[16] Lasch R, Schulze R, Legler T, et al. Workload-driven placement of column-store data structures on DRAM and NVM [C] // Proceedings of the 17th International Conference on Management of Data, 2021: 1-8.

[17] Roy T, Kant K. Enhancing endurance of SSD based high-performance storage systems using emerging NVM technologies [C] // Proceedings of the 2020 IEEE International Parallel and Distributed Processing Symposium Workshops, 2020: 1070-1079.

[18] Kornilios K, Ioannou N, Koltsidas I. Reaping the performance of fast NVM storage with uDepot [C] // Proceedings of the 17th USENIX Conference on File and Storage Technologies, 2019: 1-15.

[19] Lee DY, Jeong K, Han SH, et al. Understanding write behaviors of storage backends in Ceph object store [C] // Proceedings of the 2017 IEEE International Conference on Massive Storage Systems and Technology, 2017: 1-10.

[20] Aghayev A, Weil S, Kuchnik M. File systems unfit as distributed storage backends: lessons from 10 years of Ceph evolution [C] // Proceedings of the 27th ACM Symposium on Operating Systems Principles, 2019: 353-369.

[21] Oh M, Eom J, Yoon J, et al. Performance optimization for all flash scale-out storage [C] // Proceedings of the 2016 IEEE International Conference on Cluster Computing, 2016: 316-325.

[22] Jeong B, Khan A, Park S. Async-LCAM: a lock contention aware messenger for Ceph distributed storage system [J]. Cluster Computing, 2019, 22(2): 373-384.

[23] Kong LW, Moreno O. Characterization and prediction of performance loss and MTTR during fault recovery on scale-out storage using DOE & RSM: acase study with Ceph [J]. IEEE Transactions on Cloud Computing, 2021, 9(2): 492-503.

[24] Lu YY, Zhang JC, Yang Z, et al. OCStore: accelerating distributed object storage with open-channel SSDs [C] // Proceedings of the 39th IEEE International Conference on Distributed Computing Systems, 2019: 271-281.

[25] Kannan S, Bhat N, Gavrilovska A, et al. Re-designing LSMs for nonvolatile memory with NoveLSM [C] // Proceedings of the 2018 USENIX Annual Technical Conference, 2018: 993-1005.

[26] Atikoglu B, Xu YH, Frachtenberg E, et al. Workload analysis of a large-scale key-value store [C] // Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE Joint International Conference on Measurement and Modeling of Computer Systems, 2012: 53-64.

[27] Yang JC, Yue Y, Rashmi KV. A large scale analysis of hundreds of in-memory cache clusters at Twitter [C] // Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation, 2020: 191-208.