doi:10.3772/j.issn.1002-0470.2021.07.002

一种面向非易失性内存文件系统的数据读写粒度控制策略①

王 盈② 蒋德钧 熊 劲

(中国科学院计算技术研究所先进计算机系统研究中心 北京 100190)

(中国科学院大学 北京 100049)

摘 要 数据读写是文件系统的重要操作。传统的文件系统基于磁盘设计,数据读写需要旋转磁头,因此数据读写慢。文件系统使用I/O调度层对读写操作进行拆分和合并,减少磁头寻道时间,提高系统性能。新型非易失性内存(NVMM)支持字节寻址和随机访问,文件系统可以直接使用内存指令进行数据操作。因此,现有的NVMM文件系统,如ext4-dax、PMFS和NOVA,不再考虑数据操作粒度的优化,直接按照文件数据的存储粒度(如4 kB)在应用和文件系统之间传输数据。然而,文件系统中的数据读写性能仍然受到操作粒度的影响。本文分析了文件系统在真实NVMM硬件上的性能,发现大粒度操作会降低文件系统性能,并针对数据读写操作粒度提出了优化策略。实验结果表明,本文提出的优化策略使得NVMM文件系统性能提升30.1%。

关键词 非易失性内存(NVMM);文件系统;性能;数据读写粒度

0 引 言

文件数据读写是文件系统的重要操作,应用通过数据写操作将数据存储在文件系统中并随后执行读操作从文件系统中获得数据。当应用程序读取一个文件时,文件系统根据文件名和文件偏移定位目标文件数据,然后将目标数据从文件系统读到应用程序的动态随机存储寄存器(dynamic random access memory, DRAM)缓冲区中,文件读操作完成。与读操作相反,文件写操作将数据从应用程序的DRAM缓冲区写到文件系统中。

对于传统的文件系统,文件数据存储在磁盘中。磁盘为机械设备,数据读取和写入需要旋转磁头,因此数据在磁盘上操作慢,磁盘操作成为影响文件系统性能的主要因素。文件系统根据磁盘的访问特点,使用大粒度顺序读写减少磁盘寻道时间[1-2]。除此之外,文件系统设计了数据输入输出(input/output, I/O)调度层对磁盘读写粒度进行拆分和合并,以减少磁盘寻道时间。

新型非易失性内存(non-volatile memory, NVM)例如相变随机访问存储器(phase change RAM, PCM)[3]、可变电阻式存储器(resistive random-access memory, ReRAM)[4-5]和3D XPoint技术,提供亚微秒级别的数据访问延迟和非易失特性,可以作为内存级存储设备(non-volatile main memory, NVMM)[6-15]存储文件系统数据,提升文件系统的数据读写性能。NVMM支持字节寻址方式,文件系统中的数据可以通过load/store指令直接访问。因此,文件系统可以直接使用内存命令(如memcpy)完成文件系统和应用缓冲区之间数据的传输。

由于NVMM不需要磁盘寻道操作,现有的NVMM文件系统就没有考虑数据读写粒度对读写操作性能的影响。它们不再为文件系统设计I/O调

① 国家重点研发计划(2016YFB1000302),中国科学院战略性先导科技专项(XDB44030200)和北京市自然科学基金-海淀原始创新联合基金(L192038)资助项目。

② 女,1994年生,博士生;研究方向:计算机系统结构;联系人,E-mail: wangying01@ict.ac.cn

(收稿日期:2020-07-27)

度层,直接按照应用需求以及文件系统数据存储粒度在文件系统和应用缓冲区之间传输数据。比如,NOVA 文件系统[12]按照 4 kB 的粒度存储文件数据块,因此 NOVA 按照 4 kB 的粒度传输数据块。然而,由于 NVMM 带宽以及访问粒度的影响,数据读写的性能依然会受到当前读写粒度的影响。

首先,数据在应用内存缓冲区和 NVM 中传输需要经过中央处理器(central processing unit, CPU)缓存。传统的磁盘文件系统由于磁盘设备不支持高并发操作,且设备读写是影响文件系统性能的主要因素。它们多使用大粒度顺序操作来优化数据读写的性能。NVMM 支持高并发操作,NVMM 文件系统使用多种技术提升并发性能[11,12,16]。多线程大粒度读操作使得 CPU 缓存成为数据拷贝时的竞争资源。测试结果显示,CPU 缓存缺失使得文件系统的读性能下降 21.1%。

其次,英特尔公司于 2019 年发布了基于 3DXPoint 技术的真实 NVMM 硬件即傲腾 PMM(Otane DC persistent memory module,Optane PMM)[17]。傲腾 PMM 支持字节寻址,为研究人员提供了真实持久化内存设备。傲腾 PMM 设备的实际性能也成为持久化内存文件系统设计与优化的主要依据。例如,傲腾 PMM 提供接近于 DRAM 的写延迟以及 2~3 倍的读延迟,而傲腾 PMM 的写带宽仅是 DRAM 的 1/8[17]。大量的数据写操作会造成傲腾 PMM 内部产生冲突,降低系统性能。除此之外,傲腾 PMM 的读写粒度为 256 B,并不是真正支持单字节粒度的读写操作。因此大粒度的写操作在傲腾 PMM 上依然存在写放大的问题。测试结果显示,当傲腾 PMM 写冲突严重时,写放大使得文件系统的性能降低 46.0%。

本文评测了现有的 NVMM 文件系统的数据读写操作在真实硬件傲腾 PMM 上的性能,分析产生该性能结果的原因并提出了对数据读写粒度的优化策略。本文的主要贡献如下。

(1) 评测了 NVMM 文件系统在真实硬件傲腾 PMM 上的读写性能。

(2) 针对现有 NVMM 文件系统的评测结果进行了详细的分析,并针对读写粒度提出建议。

对于读操作,应用程序应该将大粒度操作拆分为小粒度,例如将 1 MB 读操作拆分为 256 kB,以避免大量的 LLC cache 缺失开销。

对于写操作,NVMM 文件系统应该在多线程操作时将写粒度拆分为 256 B,避免引进设备写放大,提升文件系统的性能。

1　背景介绍

本节首先介绍新型非易失性内存,然后介绍现有磁盘文件系统和 NVMM 文件系统数据读写操作方式。

1.1　新型非易失性内存

新型非易失性内存可提供快速的字节寻址访问方式,数据可以直接存储在内存总线上,提高文件数据的读写性能。英特尔公司发布的基于 3D XPoint 技术的 NVMM 设备傲腾 PMM 提供了 NVMM 的真实性能,可以基于该设备对 NVMM 文件系统的性能进行分析。

傲腾 PMM 根据其使用方式可以分为 memory mode 和 APP direct mode。Memory mode 将傲腾 PMM 当做内存设备来使用,数据不需要持久化保存。除此之外,DRAM 被用来作为傲腾 PMM 的缓存以优化性能。App direct mode 提供持久化存储,且不需要 DRAM 来作为缓存,因为文件系统需要保证数据的持久化,文件系统使用傲腾 PMM 的 APP direct mode。

傲腾 PMM 和传统的 DRAM 类似,如图 1 所示,通过内存控制器(integrated memory controller, iMC)连接在系统总线上并和 CPU 进行交互[17]。可以直接使用 load/store 指令访问傲腾 PMM 中的数据。iMC 通过基于事务的双倍速率(transactional double-data-rate, DDR-T)协议以 64 B 的粒度与傲腾 PMM 进行数据传送。傲腾 PMM 的最小读写粒度是 256 B。傲腾 PMM 用一个小的写合并缓冲区(write-combing buffer, XPBuffer)合并小于 256 B 的写操作,该缓冲区可以解决 DDR-T 协议传输粒度和傲腾 PMM 的操作粒度不对等的问题。当向傲腾 PMM 发送小于 256 B 的写操作时,新写的数据需要在 XP-

Buffer 中转换为 256 B 对齐写，再写入到傲腾 PMM 设备中。例如，64 B 的数据写操作需要先从傲腾 PMM 中将对应的 256 B 数据读入到 XPBuffer 中，然后在 XPBuffer 中更新请求的 64 B 数据，最后再将 256 B 数据写入到傲腾 PMM 存储介质中。这一操作会导致写放大，降低傲腾 PMM 的性能。

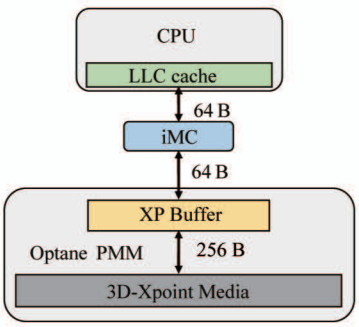


图 1 傲腾 PMM 的架构

文件系统需要保证数据操作的宕机一致性。Store 指令只会将数据写到 CPU 缓存中，不能保证数据写到傲腾 PMM。此时系统如果掉电，就会出现数据丢失。因此文件系统需要将数据从 CPU 缓存写回到傲腾 PMM 中。英特尔指令集架构提供 clflush，clflushopt 和 clwb 指令支持将数据从 CPU 缓存写回到傲腾 PMM 中。除此之外，也可以使用 non-temporal 指令（例如 ntstore）绕过 CPU 缓存直接将数据写入傲腾 PMM 中（直写）。在文件系统写操作中，多使用 non-temporal 指令将文件数据从应用程序缓冲区直写到傲腾 PMM 中。这是因为文件系统需要保证数据的持久化，non-temporal 指令相比于 clwb 等指令在大于 256 B 操作时写延迟更低[17]，且文件系统的数据读写通常大于 256 B 的。

表 1 展示了傲腾 PMM 相比于 DRAM 的延迟和带宽。可以看到，傲腾 PMM 的写延迟和 DRAM 相近。这是因为写操作不需要将数据立即写入到傲腾 PMM 存储介质中，将数据写在 XPBuffer 中就可以保证数据不丢失。英特尔提出的异步内存刷新技术（asynchronous dram refresh，ADR）保证数据在 XPBuffer 中不会掉电丢失[18]。对于读延迟，傲腾 PMM 延迟是 DRAM 的 2 ~ 3 倍。傲腾 PMM 低延迟特性可以提高文件系统的数据读写性能。然而，傲腾 PMM 的带宽低于 DRAM，且写带宽只有2.3 GB/s。在多线程情况下很容易将设备的带宽用满，进而增加设备冲突，影响数据读写的性能。本文分析和讨论了傲腾 PMM 对现有 NVMM 文件系统数据读写性能的影响，以进一步提升文件系统的性能。

表 1 傲腾 PMM 的延迟和带宽[17]

	延迟		带宽	
	读	写	读	写
DRAM	101 ns	86 ns	20 GB/s	20 GB/s
Optane PMM	305 ns	90 ns	6 ~ 7 GB/s	2.3 GB/s

1.2 文件系统读写方式

图 2（a）展示了磁盘文件系统的读操作过程。读操作时，数据通过设备驱动器从磁盘读入高速页缓存（page cache）中，并随后拷贝到应用缓冲区（application buffer，APP buffer）中；写操作时，数据从应用缓存区拷贝到页缓存中，随后再写入到磁盘设备中。由于磁盘为机械设备，数据读取和写入需要旋转磁头，因此数据在磁盘上操作慢，磁盘操作成为影响文件系统性能的主要因素。文件系统根据磁盘的访问特点，使用大粒度顺序读写以减少磁盘寻道时间[1-2]。除此之外，文件系统设计了 I/O 调度层对磁盘读写粒度进行拆分和合并，以减少磁盘寻道时间（如图 2（a）中的 I/O Scheduler layer）。例如，当文件系统需要将页缓存中的数据写到磁盘中时，I/O调度层根据请求数据在磁盘中存储的位置对请求进行拆分和合并，将操作位置近的请求放在一起批量处理，减少磁盘寻道时间，提升文件系统的读写性能。

NVMM 提供亚微秒级别的数据访问延迟，文件系统可以直接建立在非易失性主内存中提升数据读写的性能。因此，NVMM 文件系统移除了页缓存层和 I/O 调度层，数据直接使用内存命令在文件系统和应用缓冲区之间传输。图 2（b）展示了现有的基于 NVMM 的内核文件系统在执行读操作时的数据操作方法（对于用户态文件系统，数据是从用户态拷贝到用户态，和内核文件系统使用的是相同的数据传输机制）。NVMM 设备被映射到内核的地址空间，文件系统在内核态管理文件数据。当文件数据被读时，文件系统使用内存命令将文件数据从内核

态复制到应用程序提供的用户态 DRAM 缓冲区中。相比于磁盘文件系统,NVMM 文件系统读写文件时不需要在慢速的磁盘设备中读写数据,且不需要寻道操作。因此,现有的 NVMM 文件系统直接按照应用需求以及文件系统数据存储粒度在文件系统和应用缓冲区之间传输数据。

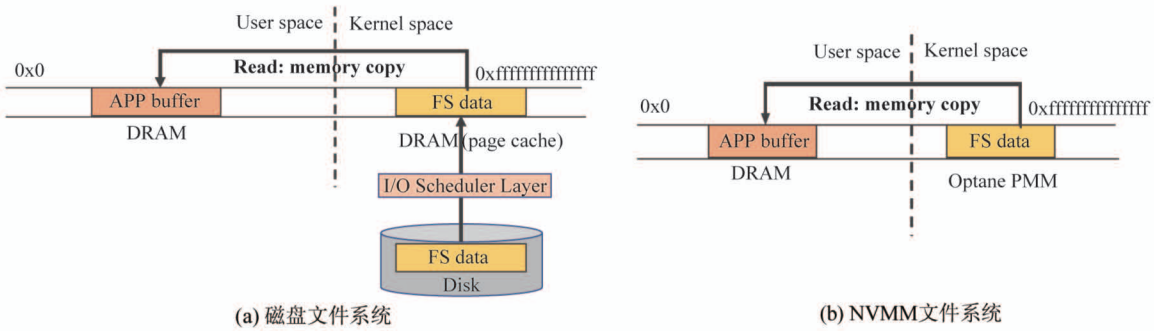


图 2　文件系统数据读写操作方式

2　相关工作

本节主要分为两个部分,现有的文件系统针对 NVMM 设备特性进行的性能优化以及现有的工作针对 NVMM 硬件傲腾 PMM 设备做出的评测分析。

2.1　NVMM 文件系统

随着 NVMM 存储设备的发展,有大量文件系统工作基于 NVMM 设备提升性能。这些工作通过降低软件开销、支持高并发操作来提高文件系统的性能。

现有的 NVMM 文件系统,例如 BPFS[6] 和 NOVA[12],移除页缓存和块设备层来避免冗余的软件栈带来的开销。文献[19-20]建议进一步移除虚拟文件系统以加速 NVMM 文件系统中的元数据访问。文献[9,16,20-21]建议使用用户态文件系统避免系统调用和内核干预的开销。文献[10]在 DRAM 中建立写缓存避免 NVMM 的高写延迟对文件系统性能的影响。除此之外,一些工作通过使用高效的索引结构,例如 Hash 表[10-11] 和基树[12] 来优化文件数据查找操作。这些工作都是针对于 NVMM 支持字节寻址以及低访问延迟所做的优化。对于文件系统中的数据读写,这些文件系统都使用相同的数据读写方法,即按照数据存储的粒度在文件系统和应用缓存区之间进行数据传输。本文根据傲腾 PMM 的访问特点和性能,在现有 NVMM 工作的基础上进一步评测并分析了文件系统数据读写的性能,提出了优化建议。

NVMM 支持高并发访问。现有的 NVMM 文件系统通过划分[11,16] 和预分配[12,21] 资源的方法减少资源的争用,进而提高文件系统的并发性。然而,这些工作没有考虑存储设备带来的限制。单块傲腾 PMM 的写带宽只有约 2 ~ 3 GB/s[17],单个线程就可以将其写带宽用完。此时只在软件层面提高并发操作是不够的,需要结合傲腾 PMM 的特点充分提高文件系统的并发操作和性能。

2.2　基于傲腾 PMM 的评测分析

文献[17]介绍了傲腾 PMM 的内部结构,分析了傲腾 PMM 在各种操作特点下的性能,并且提出了对傲腾 PMM 操作的建议:避免小于 256 B 的访问;对于大粒度写操作使用不过 CPUcache 的直写方法;限制并发线程的个数以及避免远端内存节点访问。除此之外,这些工作对 NVMM 文件系统的性能进行了评测,并根据性能结果得出了一些结论,例如写时复制技术可以保证数据的强一致性但是会增加额外的操作延迟,移除页缓存会降低 NVMM 文件系统的读性能。这些工作有助于发现 NVMM 文件系统的一些问题,并通过设计高效的软件优化文件系统的性能。但是它们没有详细评测和分析文件系统现有的数据读写粒度在傲腾 PMM 中存在的问题。本文评测了现有 NVMM 文件系统的数据读写性能,通过对这些性能结果分析发现了现有 NVMM 文件

系统数据和应用缓存区数据操作粒度的一些问题，针对这些问题给出了一些优化策略。

3 文件系统评测设置

随着 NVMM 的发展，已经有很多工作基于 NVMM 的特性设计文件系统[6-12,16,19-20]。对这些文件系统数据读写粒度进行分析并评测，发现了相同的设计和性能趋势。为了方便阅读，本文只选择一个文件系统（PMFS）进行详细的分析和评测。本文也评测了其他 NVMM 文件系统，它们与 PMFS 有相同的性能趋势。

本节首先介绍本文的测试平台及测试方法。随后，评测和分析了现有 3 个 NVMM 文件系统（ext4-dax、PMFS 和 NOVA）的性能结果，根据这 3 个文件系统的评测结果，可以看出现有的 NVMM 文件系统有相同的性能趋势。随后，本文详细分析 PMFS 文件系统。

3.1 评测平台

本文使用真实硬件傲腾 PMM 测试 NVMM 文件系统的性能。测试服务器有两个 NUMA 节点。每个 NUMA 节点有一个 Intel Gold 5215 CPU（10 个 CPU 核，LLC cache 大小为 13.8 MB），两块傲腾 PMM 内存条（128 GB ×2）以及 128 GB DRAM 空间。为了避免 NUMA 架构对测试结果的影响，将所有评测运行在 NUMA 节点 0 中。

Linux 系统将傲腾 PMM 作为一段连续的物理内存并创建命名空间进行管理[17]。傲腾 PMM 可以以交错和非交错的方式来组织命名空间。图 3（a）展示了非交错的方式，即一块傲腾 PMM 设备被组织为一个命名空间。交错方式将多块傲腾 PMM 分割成小粒度（如 4 kB）交错映射组织成一个 Namespace，如图 3（b）所示。交错方式使得一段连续的地址空间内有多个傲腾 PMM 设备，有利于提升上层应用的操作带宽。傲腾 PMM 的配置对文件系统是透明的，现有的 NVMM 文件系统只管理一个 Namespace 上的设备。在基本评测中，只使用一块傲腾 PMM 内存条，即图 3（a）中的Namespace 0.0。这样减少评测的变量，有助于对结果进行分析。在后续进一步分析中，使用两块傲腾 PMM 内存条并且使用交错（Interleave）的方式进行配置。

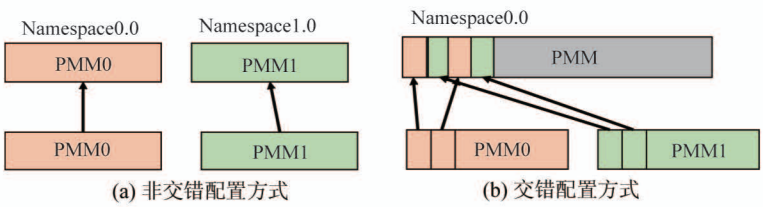


图 3 傲腾 PMM 的配置方法

3.2 评测方法

本文关注于文件系统的整体性能，以带宽作为主要的参考依据。在基本性能评测分析中（第 4 节和第 5 节），使用 fio[22] 测试文件系统在真实硬件傲腾 PMM 上的读写性能。其中每个线程操作自己的私有文件，每个文件大小为 3 GB。多个线程在软件层面上没有争用。以 256 B、1 kB、4 kB、16 kB、64 kB、256 kB、1 MB、4 MB 的粒度进行评测。

在最后验证中，使用 filebench[23] 中 fileserver 来验证对写操作建议的效果。fileserver 是一个文件服务器，支持文件混合读写操作。

3.3 文件系统性能分析

本文测试了内核 NVMM 文件系统 ext4-dax、NOVA[12] 和 PMFS[8]。用户态文件系统，例如 Strata[16] 和 SplitFS[9]，相比于内核文件系统减少了系统调用和内核干预的开销。但是对于数据操作，用户态文件系统使用和内核文件系统相同的方式，实验结果显示它们有相同的性能趋势。因此，本文只分析了内核文件系统。

图 4 显示了 ext4-dax、PMFS 和 NOVA 3 个内核文件系统分别以 256 B、16 kB 和 4 MB 粒度操作文件的读写带宽。对于小粒度读操作（图 4（a）256），

文件系统的带宽会随着线程数的增加而增加。对于大粒度操作,所有文件系统以 4 MB 粒度操作的带宽低于 16 kB 的带宽。由于 ext4-dax 的元数据性能差,如元数据操作需要经过页缓存且元数据索引结构不高效,因此 ext4-dax 的带宽比 PMFS 和 NOVA 低。

对于小粒度写操作(图 4(b) 256),只有 PMFS 的带宽会随着线程数的增加而增加。ext4-dax 的写带宽受到元数据性能的影响。NOVA 使用写时复制技术(copy-on-write, COW)保证数据宕机一致性。任何小于 4 kB 的写操作都需要复制整个 4 kB 的数据块,因此 NOVA 的性能最差。而对于大粒度操作,所有文件系统的性能趋势相似。

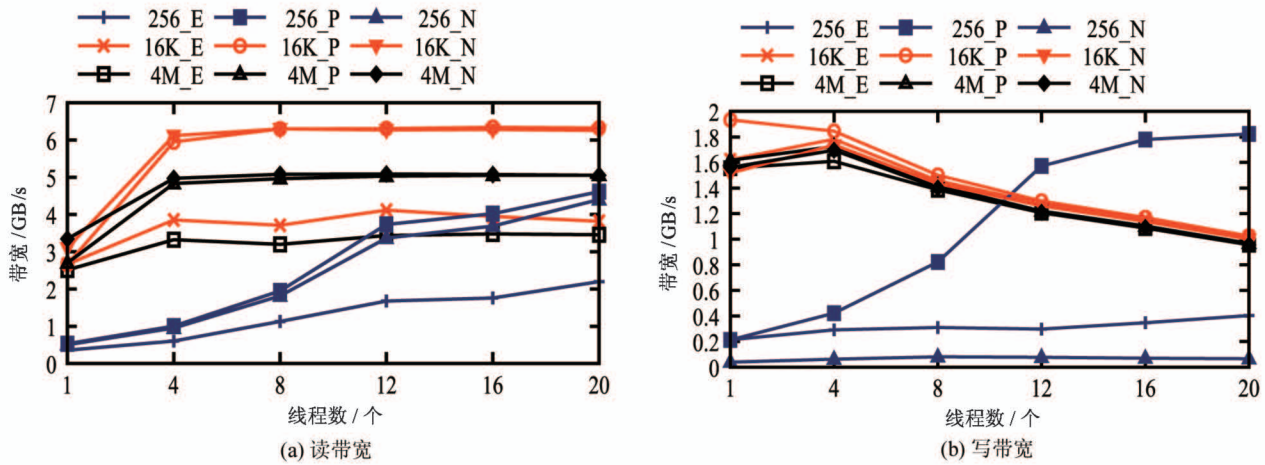


图 4　不同文件系统下的读写带宽("E"代表 ext4-dax,"P"代表 PMFS,"N"代表 NOVA)

由图 4 可以看出,除了小粒度写操作,所有文件系统在傲腾 PMM 上有相同的性能趋势,例如 4 MB 读带宽低于 16 kB 的读带宽,多线程以大粒度执行写操作时带宽会随着线程数的增加而降低。在这些文件系统中,PMFS 的性能最优。为了简化本文以方便阅读,在随后的评测分析中,以 PMFS 的结果进行分析。本文也同样验证了 ext4-dax 和 NOVA,得到了和 PMFS 相同的结论。在最后的实验验证中,使用 NOVA 和 PMFS 来验证最终效果。

4　读操作粒度控制策略

4.1　性能分析

图 5(a)显示了文件系统在多线程不同操作粒度下的读带宽。可以看到,当以 256 B 粒度执行读操作时,文件系统的带宽会随着线程的增加而增加,但是达不到设备的最大带宽(6.3 GB/s)。这主要是因为读写粒度太小,文件系统中还包含元数据操作,例如查找文件数据块的位置。当操作粒度小时,元数据操作的占比增加,因此应用测试出来的带宽达不到设备的最大带宽。

大粒度多线程操作时,例如 4 MB,带宽相比于小粒度(4 kB)操作的带宽下降 21.1%。这主要是因为单次读操作的数据量大,造成在访问过程中产生大量的末级缓存(last-level cache, LLC)缺失。在读操作中,线程需要将傲腾 PMM 中的数据读到线程的 DRAM 缓冲区中。这个过程需要先将傲腾 PMM 中的数据加载到对应的 LLC 中(图 6(a)中第 1 步),然后将 LLC 中的数据复制到应用缓冲区对应的 LLC 中(图 6(a)中第 2 步),最后数据写回到应用缓冲区内存中(图 6(a)中第 3 步)。如果按照 4 MB的粒度读文件数据,一个线程就需要 8 MB 的空间,测试机器中 CPU 的 LLC 的大小为 13.8 MB。运行 4 个线程时,所有线程一共需要 32 MB 的空间,该空间超过 LLC 大小。因此在读操作中就会产生大量的 LLCload 缺失和 LLCstore 缺失。硬件预取操作会将傲腾 PMM 的数据预取到 LLC 中,增加了 LLCload 的命中率。LLCstore 缺失是影响性能的主要因素。

图 5(b)展示了文件读操作产生的 LLC-store 缺

失的数量。在评测中，运行相同数量的线程会读取相同的文件数据量，即当运行 20 个线程时，所有粒度的评测都读取了 60 GB 的数据。可以看到当运行 20 个线程时，以 4 MB 的粒度读取文件时产生的 LLC cache store 缺失的数量是以 256 kB 读取文件的 225 倍。LLC cache store 缺失是文件系统在以 4 MB 的粒度读数据时带宽降低的主要原因。

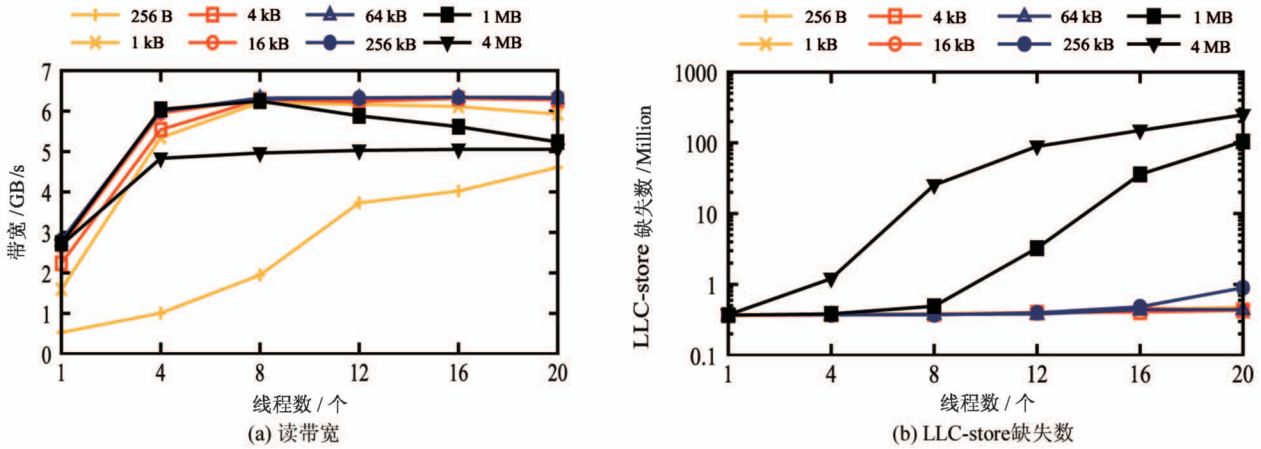


图 5　PMFS 在不同操作粒度下的读带宽和 LLC cache store 缺失数

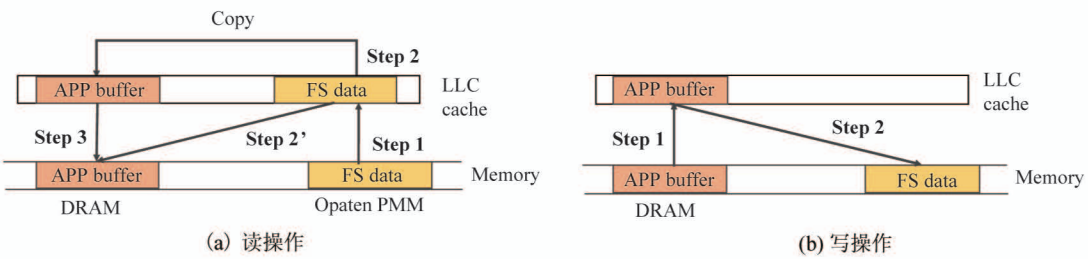


图 6　读写操作时文件系统数据拷贝操作过程

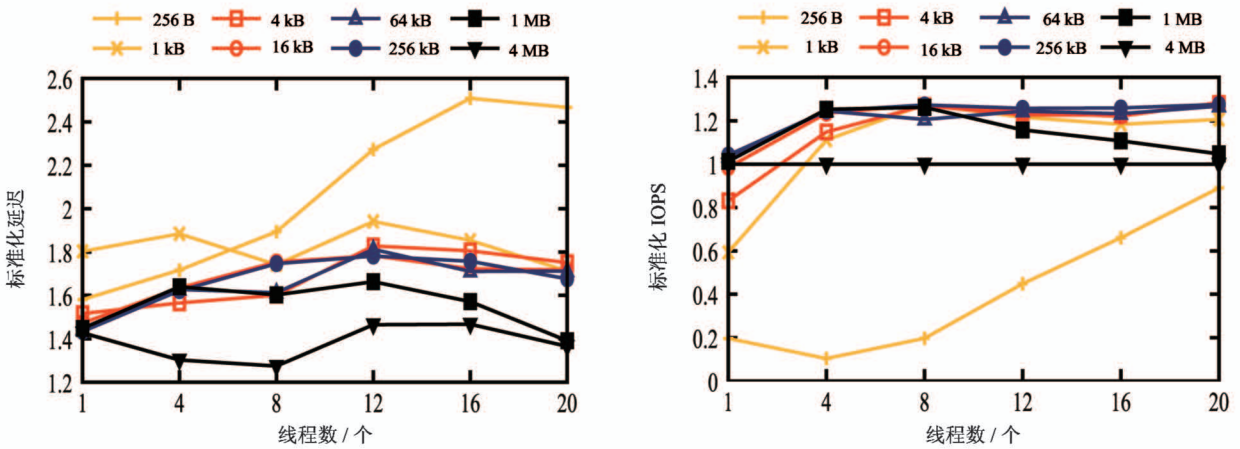
4.2　读粒度控制策略设计

直接将读取的数据越过 LLC 写到应用程序的 DRAM 缓冲区中(non-temporal 指令，如 ntstore)可以避免将新读取的数据写到 LLC 中，进而减少了 LLCstore 缺失。如图 6(a)所示，在将文件系统的数据读入到 LLC 中后，直接将数据写入到应用缓冲区内存中(图 6(a)中第 2 步)，避免一次 LLC 中的数据拷贝，降低了对 LLC 空间的需求。然而，直接将数据写入到内存缓冲区中会增加读数据的延迟。图 7(a)显示了直接将数据写到 DRAM 上的延迟比上数据写到 CPU 缓存中的结果。可以看出，在以 256 kB 的粒度读数据时，相比于将数据写到 LLC 中，直接将数据写到 DRAM 中使得延迟增加 80%，带宽降低 86%。在大粒度操作时(4 MB)，延迟也可以增加 40%。因此，直接将数据越过 LLC 写到 DRAM 中虽然减少了 LLC 缺失次数，却降低了系统整体性能。因此不建议将读取的数据直接写到应用缓冲区中。

控制应用线程的运行个数可以降低线程需要的 LLC 空间，进而减少了 LLCstore 缺失，增加了系统的带宽。然而，这种方法对系统的整体性能不友好。测试结果显示，在以 1 MB 的粒度读文件时，运行 4 个线程相比于运行 16 个线程的带宽提升了14.3%，但是 IOPS 却降低了 91.0%。这是因为应用线程除了做数据 I/O 之外还有其他操作，例如查找文件、进行计算等。直接限制应用线程的个数降低了线程在这些操作上的并发性，进而降低了系统整体吞吐率。由图 5(a)可以看到，4 个线程就可以将傲腾 PMM 的读带宽用满。在系统中只运行 4 个线程会限制系统整体的吞吐率。因此，不建议限制应用线程的运

行个数。

综上分析,在文件系统中没有比较好的方法解决大粒度读操作时傲腾 PMM 使用带宽下降的问题。因此,应用程序不应该对文件系统执行大粒度的读操作,例如 1 MB 和 4 MB 的读操作。大粒度操作导致了大量的 LLC 缺失,使得本文不能充分使用傲腾 PMM 的带宽。除此之外,相对于传统文件系统,基于傲腾 PMM 的文件系统不需要执行磁盘寻道,且元数据开销降低。例如,执行 4 kB 读操作时元数据开销(如查找要读取文件的位置)在 PMFS 中仅占 9.8%。这使得大粒度读文件的优势不再明显。因此可以得出,在 NVMM 文件系统中,应用应避免执行大粒度文件读操作。



(a) 直写操作归一化非直写操作　　(b) IOPS归一化 4 MB操作粒度

图 7　PMFS 在不同操作粒度下的归一化的读延迟和 IOPS

4.3　策略验证

图 7(b)显示了各种操作粒度下读取文件时的 IOPS 相比于以 4 MB 的粒度读取文件时的 IOPS 的结果,将所有结果都转换为 4 MB 操作粒度。例如以 256 kB 执行读操作时,图 7(b)中 256 kB 显示的 IOPS 是执行 256 kB 读操作的 IOPS 的 1/16。这是因为 256 kB 只有 4 MB 的 1/16 大。由图 7(b)可以看到,将大粒度拆分成小粒度(256 kB 和 64 kB)之后,系统总的 IOPS 相比于按照 4 MB 的粒度访问 IOPS 可以增加 20%。因此本文建议应用程序避免使用大粒度的文件读操作,例如使用 256 kB 的粒度执行文件读操作。

5　写操作粒度控制策略

5.1　性能分析

与读操作不同,NVMM 文件系统的数据写操作不会导致大量的 LLCstore 缺失。这是因为文件系统的数据写大都是绕过 CPU 缓存直接写到 NVMM 介质中。如图 6(b)所示,将应用缓冲区的数据读到 CPUcache 之后(第 1 步),直接将数据写入到傲腾 PMM 内存中(第 2 步)。虽然直写操作会增加写操作的延迟,如图 7(a)所示,但是文件系统需要将新写的数据持久化到 NVMM 中以保证文件数据的宕机一致性。如果不用直写操作,就需要使用 clflush,clflushopt 和 clwb 指令将 CPU 缓存中的数据刷回到 NVMM 中。这些指令的性能在操作大于256 B数据时比直写操作差[24],因此 NVMM 文件系统使用直写的方式将文件数据写回到 NVMM 中。直写方式使得 LLC 缺失不再是影响写操作性能的主要因素。

图 8(a)展示了文件系统在多线程不同操作粒度下的写带宽。可以看到,在小粒度操作时(256 B),带宽会随着线程数的增加而增加。合适的操作粒度(16 kB 及 4 kB)可以使得单线程写操作接近设备的最高带宽。

随着线程数的增加,应用的带宽降低。而且操作粒度越大带宽降低越多。这主要是因为傲腾 PMM以256B的粒度进行设备的写操作,且新写

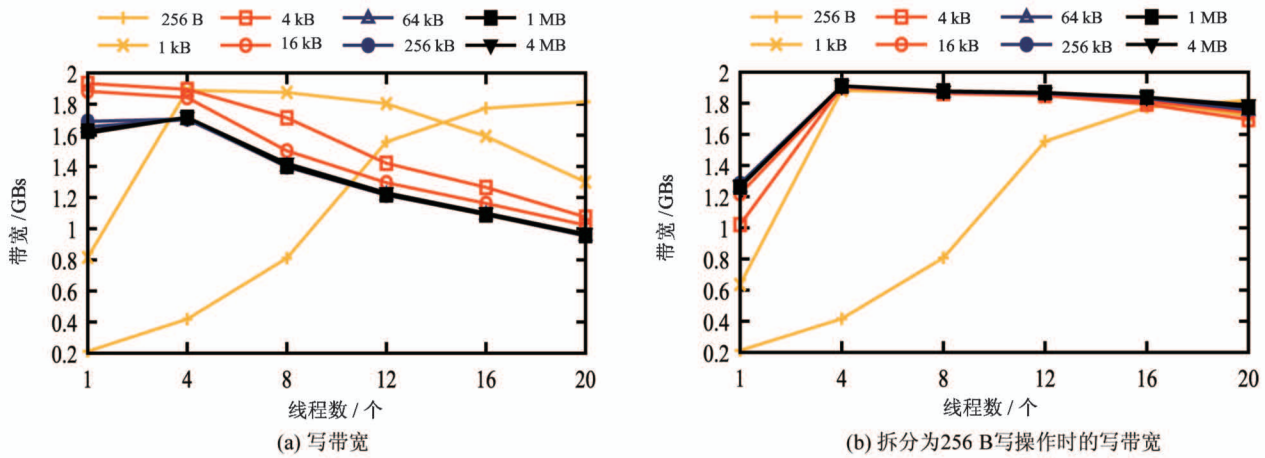


图 8　NVMM 文件系统在不同操作粒度下的写带宽

的数据先被存储在傲腾 PMM 内部 XPBuffer 中(如图 1 所示)。数据由 XPBuffer 写入到傲腾 PMM 介质中的延迟远高于数据写入 XPBuffer 的延迟,这导致 XPBuffer 空间的使用存在大量冲突。除此之外,CPU 乱序执行使得写操作没有充分使用 XPBuffer 的局部性,导致了设备内部的写放大,进而影响了系统的带宽。例如,一次 256 B 的写操作向傲腾 PMM 发送 4 次 64 B 的写操作,XPBuffer 冲突可能会导致每 64 B 的数据都向傲腾 PMM 存储介质执行了一次 256 B 的写操作。因此 256 B 的数据总共执行了 4 次写,导致了 3 倍写放大。测试结果显示,用 20 个线程执行 4 MB 的写操作时,傲腾 PMM 产生了 1 倍的写放大。

5.2　写粒度控制策略设计

为了降低因为 XPbuffer 冲突导致的设备写放大操作,可以对单次写拷贝的数据进行拆分,减小文件系统单次数据写的粒度以减少 CPU 乱序执行,避免傲腾 PMM 中的写放大问题。以写 4 kB 的数据为例,文件系统可以先向傲腾 PMM 发送写 256 B 的请求,等到当前请求将数据写到 XPBuffer 中后再发送下一个 256 B 写请求,以此类推。这种方法可以减小 XPBuffer 冲突,进而减少傲腾 PMM 设备写放大操作。图 8(b)中将所有的写操作拆分成 256 B 写操作的带宽(对于 64 B 写操作,仍然按照 64 B 的粒度向傲腾 PMM 写数据)。可以看到,当运行多个线程时,拆分后的带宽相比于没有拆分时的带宽有所提升。特别是运行 20 个线程时,拆分操作使得带宽增加 80%。这是因为依次向傲腾 PMM 发送 256 B 的写操作可以减少 XPBuffer 的冲突,使得 256 B 的写数据一次写入傲腾 PMM 的存储介质中,减少了写放大操作,进而提高了性能。

然而,当运行单个线程时,将操作拆分成 256 B 的写操作使得写带宽平均降低 20%。这主要是因为单线程时傲腾 PMM 没有产生严重的冲突,拆分操作在一定程度上阻止了 CPU 乱序执行,增加了操作延迟,也降低了应用的带宽。

为了进一步分析拆分操作对文件系统性能的影响,对拆分粒度在不同线程和不同操作粒度下的结果进行了分析。图 9 展示了在单块傲腾 PMM 下的带宽结果。其中图 9(a)、(b)和(c)分别展示了以 4 kB、16 kB 和 64 kB 粒度执行写的带宽。“PMFS”代表对文件系统写操作粒度没有进行拆分;“256”,“512”和“1 kB”分别展示了以 256 B,512 B和 1 kB 粒度对写操作粒度进行拆分的结果。可以看到,当操作线程个数大于 1 时,以 256 B 的粒度进行拆分就可以获得最高的带宽。这是因为单个写线程就可以跑满傲腾 PMM 的写带宽,多线程下将写操作拆分为 256 B 避免了因为 XPBuffer 冲突导致的写放大问题。

对于单线程操作,4 kB 写操作时不进行拆分带宽最高,这是因为写操作没有导致严重 XPBuffer 冲突。而当操作粒度大于 16 kB 的时候,拆分为 512 B 粒度最优。512 B 粒度的拆分可以减少 XPBuffer 的冲突,并且相比于 256 B 降低了写延迟。

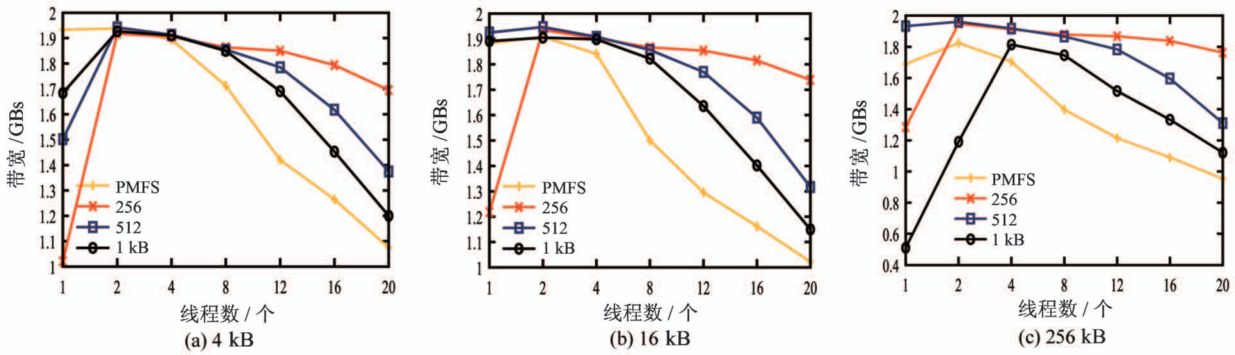


图 9　单块傲腾 PMM 设备的带宽

为了更充分地分析拆分操作对文件系统写性能的影响,使用两块傲腾 PMM 内存条以交错方式配置来测试写带宽。图 10 展示了两块傲腾 PMM 设备以交错模式时的带宽。可以看出,与单块傲腾 PMM 相同,线程数大于 2 个时拆分为 256 B 时最优。而线程数小于等于 2 个时,小于 16 kB 时写不拆分最优,否则拆分为 512 B 最优。由此可以看出,是否进行拆分和线程个数、傲腾 PMM 设备数以及操作粒度有关,可以得出以下结论。

(1) 当写线程个数超过傲腾 PMM 的个数时,拆分为 256 B 写时可以充分利用傲腾 PMM 的带宽。

(2) 当写线程数小于等于傲腾 PMM 的个数时,小于 16 kB 写操作时不进行拆分,否则按照 512 B 的粒度进行拆分写操作。

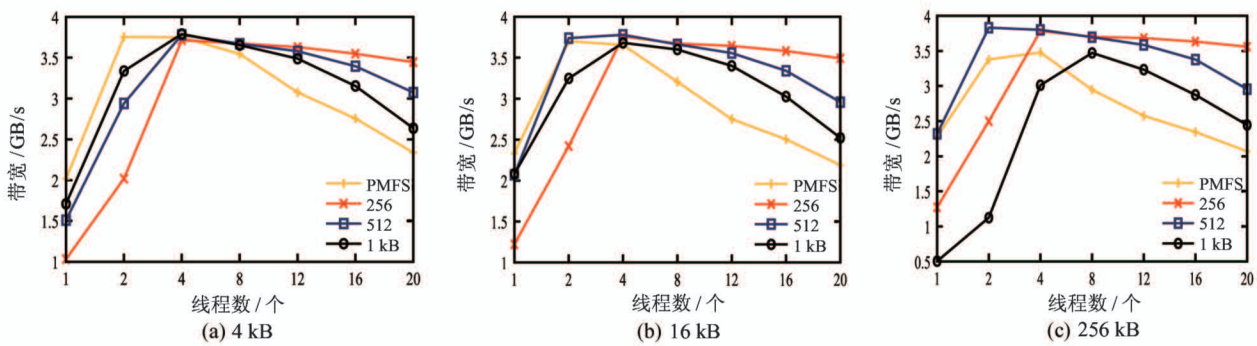


图 10　单块傲腾 PMM 设备的带宽

5.3　策略验证

按照第 5.2 节中的结论,在文件系统中增加了一个计数器,用来统计当前文件系统中写线程的个数。当文件系统向傲腾 PMM 写数据时,根据正在向傲腾 PMM 写数据的线程个数和线程操作粒度对写操作按照 5.2 节中的结论进行拆分。图 11 分别展示了在交错和非交错模式下对写操作进行拆分后的带宽相比于没有拆分的带宽的结果,可以看到,拆分操作使得写带宽提升 80%。同时,在一些小粒度多线程操作的情况下(1 kB),现有的预测拆分方法会使得性能下降约 10%。这主要是因为在文件系统中增加了写线程个数统计的开销,这些开销在小粒度操作时影响会比较大。除此之外,现在的预测方法和统计信息针对 5.2 节中的第 2 条结论不够精确,需要更细粒度的分析。未来计划通过傲腾 PMM 设备的写放大情况来决定数据的拆分策略。当前设备的写放大信息能更精确地反映傲腾 PMM 的使用冲突情况,有利于准确地拆分数据。

6　应用级评测

本节对第 5.2 节中提出的写优化策略进行验证。使用 filebench[23] 中的测试集 fileserver 基于两块傲腾 PMM 设备(交错模式)进行测试。Fileserver 是一个文件服务器,文件读写操作占比为 1:2,包括文件创建、删除、追加、读和写操作。测试的文件大

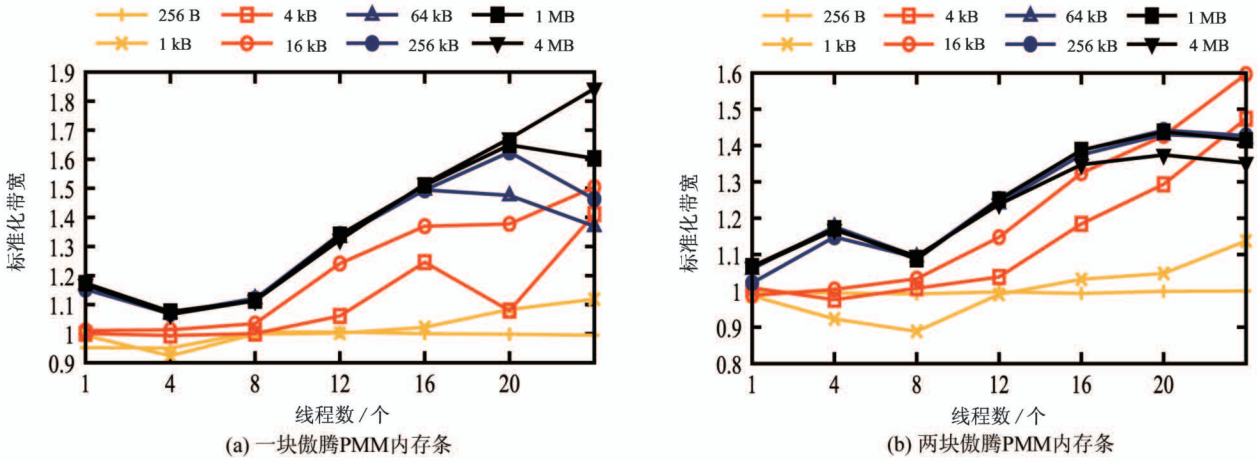


图 11　拆分后的带宽对比上没有拆分后的结果

小为 10 MB，文件读写粒度为16 kB，一共测试 1 万个文件。图 12 展示了分别以 20、12、2 个线程运行时 PMFS、NOVA 以及对它们进行优化后（PMFS _ W、NOVA _ W）的吞吐率。

当运行 20 和 12 个线程的时候，将写操作拆分成 256 B 可以提升 fileserver 在文件系统中的吞吐率。这主要是因为多线程情况下傲腾 PMM 的访问压力大，大粒度写操作容易导致写放大，降低系统的性能。以 256 B 的粒度向傲腾 PMM 设备发送写操作可以缓解访问压力，减少设备的写放大。测试结果显示在运行 20 个线程时，拆分操作使得 NOVA 和 PMFS 的吞吐率分别提升 25.1% 和 30.1%。

对于 2 个线程的操作，写线程比较少，傲腾 PMM 访问压力小。第 5 节提出的建议只有在写粒度大的时候才进行写拆分操作，因此在操作小于 16 kB的时候，PMFS _ W（NOVA _ W）和 PMFS（NOVA）的性能几乎没有区别。对于大粒度操作，每个线程除了写操作之外还有其他操作，而且其他操作的占比相比于第 5 节中的 fio 的评测更大，因此性能提升效果不明显。

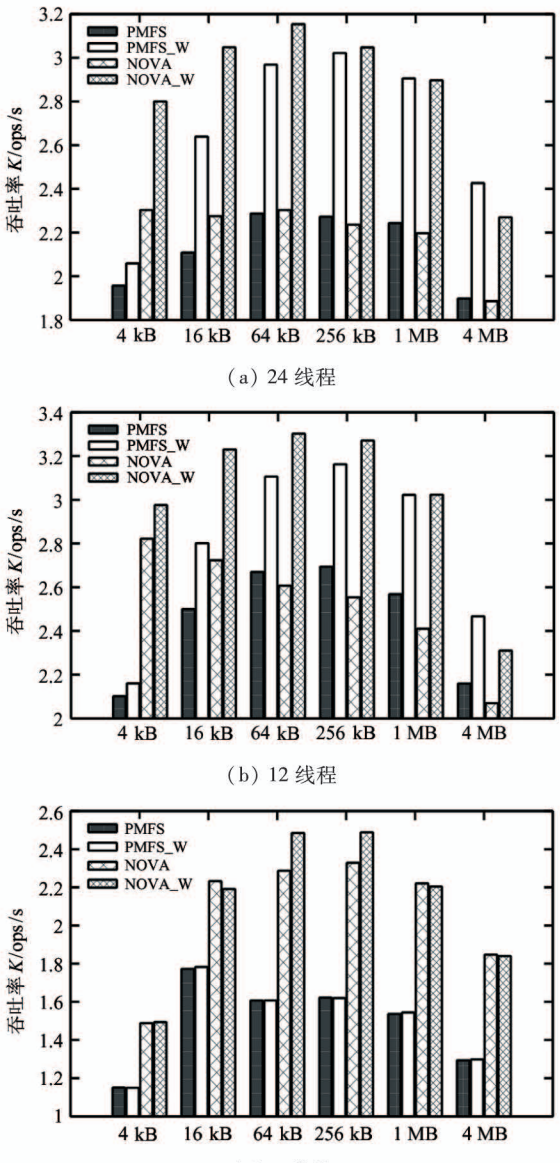


图 12　Fileserver 的吞吐率（“_ W”代表将写操作进行拆分以减少傲腾 PMM 写冲突问题）

7　结　论

数据读写是文件系统中的重要操作。非易失性内存可以直接连接在地址总线上，提供低延迟且支持字节访问。文件系统可以使用内存指令直接进行数据操作，提高了文件读写的性能。然而，现有的文

件系统和应用程序基于磁盘设计,为了减少磁头旋转的开销,多使用大粒度顺序操作来提升空间局部性。除此之外,现有的 NVMM 文件系统忽略了读写粒度对性能的影响,对应用发出的请求直接使用文件系统的存储粒度进行操作。本文对文件系统在真实 NVMM 硬件傲腾 PMM 中的评测结果进行分析,发现大粒度数据操作(如大于等于 1 MB 的读操作)不利于 NVMM 文件系统充分使用设备带宽,文件系统应根据设备的特点做出优化,提升整体性能。对此,面向非易失性内存文件系统的数据读写粒度的控制策略为:对于读操作,应用应尽可能避免大粒度操作,如将 1 MB 的操作拆分为256 B的操作;对于写操作,在多线程操作下应该将数据操作粒度拆分为 256 B,避免在傲腾 PMM 中产生写放大。实验表明,将大粒度写操作拆分为 256 B 可以使得应用程序的性能提升 30.1%。

参考文献

[1] Mathur A, Cao M, Bhattacharya S, et al. The new ext4 filesystem: current status and future plans[C]//Proceedings of the Linux Symposium, Ottawa, Canada, 2007:21-34

[2] Janner W, Yuan J, Zhan Y, et al. BetrFS: a right-optimized write-optimized file system[C]//Proceedings of the 13th USENIX Conference on File and Storage Technologies, Santa Clara, USA,2015:301-305

[3] Baek I, Lee M, Seo S, et al. Highly scalable nonvolatile resistive memory using simple binary oxide driven by asymmetric unipolar voltage pulses[C]//Proceedings of Electron Devices Meeting, IEDM Technical Digest, San Francisco, USA, 2004: 587-590

[4] Akel A, Caulfield A, Mollov T. Onyx: a prototype phase change memory storage array[C]//Proceedings of the 3rd USENIX Conference on Hot Topics in Storage and File Systems, Portland, USA, 2011:2-2

[5] Raoux S, Burr G, Breitwisch M, et al. Phase-change randomaccess memory: a scalable technology[J]. *IBM Journal of Research and Development*, 2008, 52(4/5): 465-479

[6] Condit J, Nightingale E, Frost C, et al. Bpfs:better I/O through byte-addressable, persistent memory[C]//Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles, Big Sky Montana, USA, 2009: 133-146

[7] Dong M K, Chen H B. Soft updates made simple and fast on non-volatile memory[C]//Proceedings of the 2017 USENIX Annual Technical Conference, Santa Clara, USA, 2017: 719-731

[8] Dullor S, Kumar S, Keshavamurthy A, et al. System software for persistent memory[C]//Proceedings of the 9th European Conference on Computer Systems, EuroSys '14, Amsterdam, Netherlands, 2014: 1-15

[9] Kadekodi R, Lee S K, Kashyap S, et al. Splitfs: reducing software overhead in file systems for persistent memory[C]//Proceedings of the 27th ACM Symposium on Operating Systems Principles, New York, USA, 2019:494-508

[10] Ou J, Shu J, Lu Y Y. A high performance file system for non-volatile main memory[C]//Proceedings of the 11th European Conference on Computer Systems, London, UK, 2016:12:1-12:16

[11] Volos H, Nalli S, Panneerselvam S, et al. Aerie: flexible file-system interfaces to storage-class memory[C]//Proceedings of the 9th European Conference on Computer Systems, Amsterdam, Netherlands, 2014: 1-14

[12] Xu J, Swanson S. NOVA: a log-structured file system for hybrid volatile/non-volatile main memories[C]//Proceedings of the 14th USENIX Conference on File and Storage Technologies, Santa Clara, USA, 2016: 323-338

[13] 何耀,蔡涛,彭长生, 等. 面向 NVM 的混合粒度文件系统[J]. 软件导论, 2016,7(15): 17-20

[14] 蔡涛,王杰,牛德娇,等. 基于冲突检测的高吞吐 NVM 存储系统[J]. 计算机研究与发展,2020, 57(2):257-268

[15] 何耀,牛德娇,蔡涛,等. SCMPFS:面向 SCM 的聚合文件系统[J]. 计算机研究与发展,2015,52(S2):18-28

[16] Kwon Y, Fingler H, Hunt T, et al. Strata: a cross media file system[C]//Proceedings of the 26th Symposium on Operating Systems Principles, Shanghai, China, 2017: 460-477

[17] Intel. Intel(R) Optane(TM) DC Persistent Memory[EB/OL]. https://www.intel.com/content/www/us/en/architecture-and-technology/optane-dc-persistent-memory.html: Intel, 2021

[18] Rudoff A. Deprecating the PCOMMIT instruction[EB/OL]. https://software.intel.com/en-us/blogs/2016/09/12/deprecate-pcommit-instruction: Intel, 2016

[19] Wang Y, Jiang D J, Xiong J. Caching or not: rethinking virtual file system for non-volatile main memory[C]//Proceedings of the 10th USENIX Workshop on Hot Topics in Storage and File Systems, Boston, USA, 2018: 1-6

[20] Zhou D, Pan W, Wang W. A file system bypassing volatile main memory: towards a single-level persistent store[C]//Proceedings of the 15th ACM International Conference on Computing Frontiers, Ischia, Italy, 2018:97-104

[21] Dong M, Bu H, Yi J, et al. Performance and protection in the ZoFS user-space NVM file system[C]//Proceed-

ings of the 27th ACM Symposium on Operating Systems Principles, Huntsville, Canada, 2019: 478-493
[22] Axobe J. Fio-2.14[EB/OL]. https://github.com/axboe/fio: GitHub, 2021
[23] Filebench 1.4.9.1[EB/OL]. https://github.com/filebench/filebench/wiki: GitHub, 2021
[24] Yang J, Kim J, Hoseinzadeh M, et al. An empirical guide to the behavior and use of scalable persistent memory[C]//Proceedings of the 18th USENIX Conference on File and Storage Technologies, Santa Clara, USA, 2020: 169-182

A strategy of data accessing granularity for non-volatile main memory file systems

Wang Ying, Jiang Dejun, Xiong Jin
(Advanced Computing System Laboratory, Institute of Computing Technology Chinese Academy of Sciences, Beijing 100190)
(University of Chinese Academy of Sciences, Beijing 100049)

Abstract

Data reading and writing are important operations of the file system. The traditional block file system is designed on disk, and the data read and write require moving the disk head. Therefore, data reading and writing are slow. The file system uses I/O scheduler layer to split and merge read and write operations to reduce disk seek time and improve system performance. The emerging non-volatile main memory (NVMM) supports byte-addressable and can be directly attached to memory bus. File systems can directly use the memory command to operate data, improving the performance. Therefore, the existing NVMM file systems, such as ext4-dax, PMFS and NOVA, no longer consider the data operations size, and they directly transfer data between the application and file system according to the storage size of the file data (4 kB). However, the data read and write performance in the file system is still affected by the granularity of operations. The performance of file systems on real NVMM hardware is analyzed, it is found that large granularity operations reduce the performance of the file system, and optimization strategies for data operation granularity are proposed. The experimental results show that these strategies can improve the performance of NVMM file system by 30.1%.

Key words: non-volatile main memory (NVMM), file system, performance, data reading and writing size