计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2020.20190574
57(2):281-290,2020

一种持久性内存文件系统数据页的混合管理机制

陈游旻[1] 朱博弘[1] 韩银俊[2] 屠要峰[2] 舒继武[1]
[1](清华大学计算机科学与技术系 北京 100084)
[2](中兴通讯股份有限公司 南京 210012)
(chenym16@mails.tsinghua.edu.cn)

A Hybrid Approach for Managing Data Pages in Persistent Memory File Systems

Chen Youmin[1], Zhu Bohong[1], Han Yinjun[2], Tu Yaofeng[2], and Shu Jiwu[1]
[1](*Department of Computer Science and Technology, Tsinghua University, Beijing* 100084)
[2](*ZTE Corporation, Nanjing* 210012)

Abstract Intel has officially released the Optane DC Persistent Memory based on 3D-Xpoint technology in April 2019, which provides new opportunities for building efficient persistent memory storage systems. However, existing software is far from fully exploiting the hardware performance, due to the ignorance of utilizing the byte-addressable feature of persistent memory. This paper proposes a hybrid data page management (HDPM) mechanism. It manages file data by selectively using the copy-on-write technique and log-structure, so as to fully utilize the byte-addressable feature of persistent memory. It can avoid the redundant copy overhead as in traditional approaches when processing un-aligned or small-sized writes. To guarantee the read performance unaffected, HDPM introduces reverse-scanning mechanism, which avoids the additional data copying when rebuilding data pages from the log. HDPM also introduces a multi-stage garbage collection mechanism for log cleaning. When a single log is too large, it's automatically reclaimed by read/write system calls. When the persistent memory space is limited, a background thread asynchronously reclaims the log space with a lock-free approach, without affecting the normal read/write performance. Experiments show that HDPM provides high write performance. Compared with NOVA, a state-of-the-art persistent memory file system, HDPM exhibits 58% lower write latency at most with the small-sized and write-intensive workload, and provides comparable performance for read operations. Our evaluation with Filebench shows that HDPM outperforms NOVA by 33% at most with 10 concurrent threads.

Key words persistent memory; file system; copy-on-write; log-structure; garbage collection

摘 要 英特尔于2019年4月正式发布基于3D-Xpoint技术的傲腾持久性内存(Optane DC persistent memory),这为构建高效的持久性内存存储系统提供了新的机遇.然而,现有的存储系统软件并不能很好地利用其字节寻址特性,持久性内存性能很难充分发挥.提出一种文件系统数据页的混合管理机制HDPM,通过选择性使用写时复制机制和日志结构管理文件数据,充分发挥持久性内存字节可寻址

收稿日期:2019-08-19;**修回日期**:2019-11-18
基金项目:国家重点研发计划项目(2018YFB1003301);中兴通讯股份有限公司合作项目(20182002008);广东省科技创新战略专项项目(2018B010109002)
This work was supported by the National Key Research and Development Program of China (2018YFB1003301), the Project of ZTE (20182002008), and the Science and Technology Innovation Project of Guangdong Province (2018B010109002).
通信作者:舒继武(shujw@tsinghua.edu.cn)

特性,从而避免了传统单一模式在非对齐写或者小写造成的写放大问题.为避免影响读性能,HDPM 引入逆向扫描机制,实现日志结构重构数据页时不引入额外数据拷贝.HDPM 还提出一种多重垃圾回收机制进行日志清理.当单个日志结构过大时,通过读写流程主动回收日志结构;当持久性内存空间受限时,则通过后台线程使用免锁机制异步释放日志空间.实验显示,HDPM 相比于 NOVA 文件系统,单线程写延迟降低达 58%,且读延迟不受影响;Filebench 多线程测试显示,HDPM 相比于 NOVA 提升吞吐率 33%.

关键词 持久性内存;文件系统;写时复制;日志结构;垃圾回收

中图法分类号 TP316

新型的非易失内存(non-volatile memory, NVM),例如相变存储器(phase change memory, PCM)[1-3]、阻变存储器(resistive random-access memory, ReRAM)[4]、3D-Xpoint 等,能够提供与 DRAM 相近的访问性能,同时还能像磁盘一样持久性地存储数据.其中,通过内存接口与 CPU 相连,从而具有字节可寻址能力的非易失内存被称作持久性内存(persistent memory, PM).2019 年 4 月,英特尔公司正式发布了基于 3D-Xpoint 技术的傲腾持久性内存(Optane DC persistent memory)[5],这是迄今为止全球首款大规模商用的持久性内存器件,其单条最高容量可达 512 GB,访问带宽及延迟相比于固态硬盘具有一到多个数量级的提升.

持久性内存为构建高效的存储系统提供了性能保障.然而,现有的持久性内存存储系统软件或直接从传统软件改进而来[6-7],或依旧借鉴了面向传统磁盘或固态硬盘的设计思想[8-10],从而导致持久性内存的硬件特性没能被很好地利用,其硬件性能很难得到充分发挥.例如,传统硬盘和固态硬盘的最小访问粒度分别为扇区(512 B)和页(4 KB),因此,传统文件系统通常以 4 KB 的数据页为最小粒度管理文件数据.然而,现有的持久性内存文件系统依旧沿用了该设计,以 4 KB 为最小粒度管理文件数据读写,并通过写时复制机制(copy-on-write, CoW)保证数据写入的原子性[9-10],使持久性内存的字节寻址特性未能得到充分发挥.通过应用负载分析,发现大量应用有一定比例的写操作具有“不对齐”、“小写”等特性,页粒度的写时复制机制将引入严重的数据写放大效果,从而影响文件系统的整体性能.

本文提出了一种数据页混合管理机制(hybrid data page management, HDPM),通过选择性使用写时复制机制和日志结构(log-structure)管理文件数据,从而避免非对齐写或者小写造成的写放大问题.具体地,HDPM 通过同时维护一个固定粒度的持久性内存块和一个日志结构管理各文件数据页:其将对齐的写操作数据直接存入固定粒度的持久性内存块中,而将非对齐的写操作数据以日志项的形式追加到日志结构末尾,从而避免了传统写时复制机制引入的额外拷贝操作,消除了写放大问题.为避免影响读性能,HDPM 引入逆向扫描机制,从日志结构末尾逆向扫描,从而保证在数据页重构过程中不引入额外的数据拷贝.为避免日志结构无限增长,HDPM 提出一种多重垃圾回收机制,在单个日志结构过大时,通过读操作在重构数据页时主动回收日志结构,当持久性内存空间受限时,则通过后台线程使用免锁机制异步释放日志空间,从而消除垃圾回收对系统整体性能的影响.

本文的主要贡献有 3 个方面:

1) 提出了一种数据页混合管理机制 HDPM,通过选择性地将对齐和不对齐的数据存放到持久性内存块和日志结构,避免了额外的数据拷贝开销.

2) 提出逆向扫描机制降低文件读取时的拷贝开销;提出多重垃圾回收机制,高效回收日志空间.

3) 通过微观和宏观的实验分析,HDPM 相比于传统途径能显著提升写入性能.其中,小粒度写入操作负载下,HDPM 的写入延迟比 NOVA 低 58%;Filebench 多线程测试显示,HDPM 相比于 NOVA 提升吞吐率 33%.

1 背景介绍与研究动机

本节主要介绍持久性内存的基本硬件特性、文件系统中常见的原子性更新机制的基本工作原理,以及典型的应用负载分析.

1.1 持久性内存

持久性内存通过内存总线直接与 CPU 相连,CPU 可通过内存指令直接读写持久性内存.持久性内存的出现彻底颠覆了传统的存储体系结构,从

“内存+外存”的2级结构变成了内存级存储的单层结构[11].持久性内存还带来了2个层面的变化:1)读写不对称性,持久性内存的写性能往往不及读性能,例如,傲腾持久性内存的读带宽最高可达39.4 GB/s,而写带宽最高只有13.9 GB/s[12];2)持久性-易失性边界发生变化,传统情况下需要通过软件方式将数据从易失性内存(dynamic random-access memory, DRAM)搬运到持久性的磁盘或固态硬盘,而在持久性内存架构下,该边界位于CPU缓存和持久性内存之间,且数据到达持久性内存的过程完全由硬件控制.CPU会通过打乱内存读写操作来提升访存性能,因此,数据到达持久性内存的顺序往往并不能按照程序写入的次序进行,彼此有依赖关系的数据写入会因为乱序执行导致掉电故障时出现不一致的情况.为此,需要额外执行强制缓存刷新指令(如clflushopt,clwb等)将数据主动逐出缓存.然而,这些缓存刷写指令开销大,对系统性能影响十分严重.

1.2 文件原子性更新机制

为保证文件数据在发生系统崩溃或断电故障时要么写入完成,要么未执行,文件系统需要额外的机制来保证其原子执行.常见的原子性更新机制包括日志(journal)、写时复制、日志结构(log-structure)等.

其中,日志方式将存储空间分成数据区和日志区,在更新数据前,先将旧版本或新版本数据写入日志,待持久化完毕后再原地更新数据.若在此过程中发生故障,可根据日志状态重做或撤回写入操作,实现数据更新的原子性.日志机制的缺陷是所有的数据均需写2次,对写性能影响严重.图1展示了写时复制机制的过程,该方案需将原数据页中未修改的数据先拷贝到新的数据页中,然后再将新写入的数据写到新数据页中.该方案在对齐写场景下性能优异,避免了所有的拷贝开销;然而,针对非对齐写,其额外拷贝开销甚至可能比日志方式还高.日志结构与日志机制完全不同:日志结构将整个存储空间组建成一个日志,而不是像日志机制一样将存储空间划分为数据区和日志区;进而,所有的更新数据都被追加到日志末尾,并借助相应的索引结构引导数据读取.为防止日志任意增长,还需要垃圾回收机制消除日志中的旧版本数据.在基于磁盘的日志结构文件系统中,每次追加日志项时仍需要按页对齐,额外拷贝无法完全消除;然而,在字节可寻址的持久性内存中无需再按页对齐日志项,日志结构可以紧凑地排布在持久性内存中,因而避免任何额外拷贝.

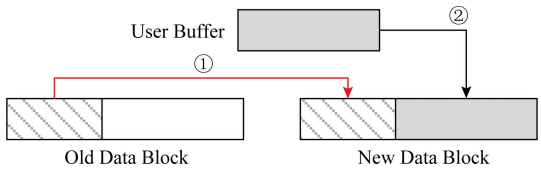


Fig. 1 Copy-on-write mechanism

图1 写时复制机制

1.3 负载分析

为理解写时复制机制在真实应用中造成的影响,我们进一步分析了多个真实负载的数据写入特性.如图2所示,其横坐标代表对一个4 KB数据页实际更新的数据量大小,纵坐标代表其累计分布(CDF).观察可见,Iphoto,Usr1,Usr2等负载具有超过30%的写操作为非对齐写,而Facebook和Twitter也具有10%左右的非对齐写操作,仅TPCC表现出良好的对齐写特性.通过一定的真实负载分析,我们发现真实负载均不同程度地存在非对齐写操作,现有的写时复制机制将引入额外的拷贝开销.

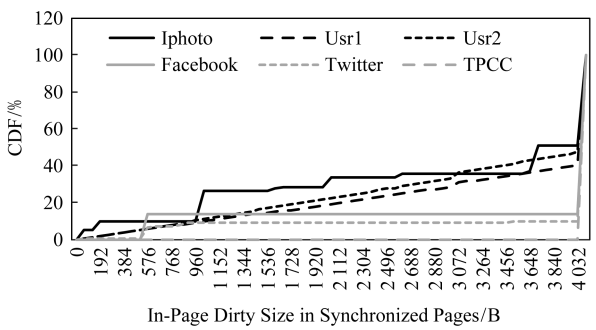


Fig. 2 CDF of in-page update size

图2 页内更新数据大小累积分布图

2 HDPM设计

本节首先描述HDPM的系统整体架构,然后逐一介绍HDPM的设计细节,具体包括混合数据页管理机制,基于逆向扫描的文件读取流程,以及高效的多重垃圾回收机制.

2.1 系统架构

图3上半部分展示了HDPM的系统总体架构:应用程序通过POSIX标准读写接口访问文件系统,其中,混合数据页管理模块(2.2节)用于处理文件写入流程,逆向扫描读取模块(2.3节)用于处理文件读取流程.持久性内存空间被切分成固定大小的持久性内存块,用于存放数据页和元数据.为方便实现,文件的inode采用了块映射机制,即为每个数据

页存放一个映射条目(如图 3 下部分).与传统的块映射机制不同的是,每个映射条目包含 2 个指针,分别指向一个写时复制页和一个日志结构,用于支持混合的数据页管理.当然,HDPM 不局限于使用块映射机制进行数据块索引,广泛使用的范围索引(extent)方式同样适用于 HDPM,本文将不再赘述.上述日志结构由一个或多个持久性内存块串接而成,当日志结构空间不足时,将分配一个新的内存块串接到日志尾部,用于存放新的数据.当日志结构的长度超过阈值,或剩余空间不足时,将触发多重垃圾回收机制(2.4 节),选择性地在前台或后台清除日志结构引入的旧版本数据.

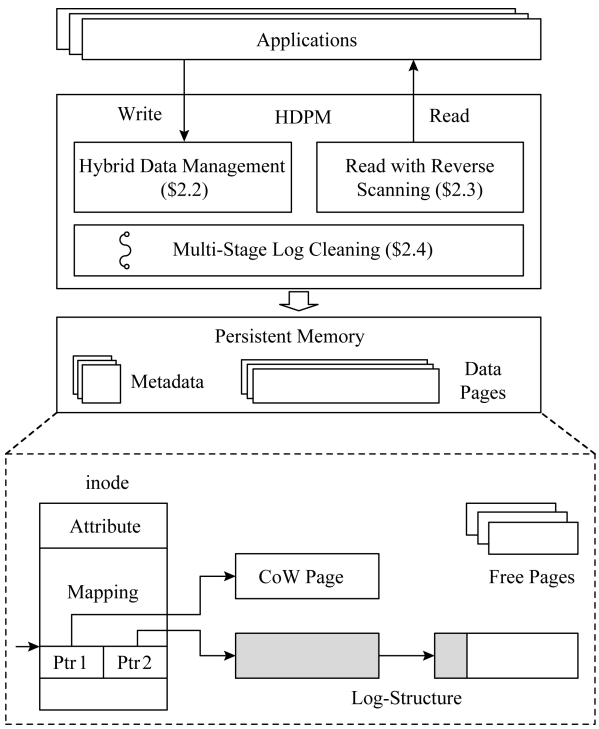


Fig. 3 Overall architecture of HDPM

图 3 HDPM 总体架构

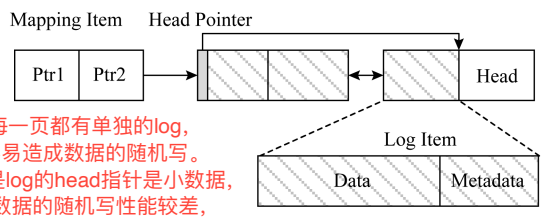
每个page单独一个log,空间开销会比较大,如果GC不及时,一个逻辑page可能需要独占两个物理page的空间

2.2 混合数据页管理

混合数据页管理旨在选择性利用写时复制页和日志结构存储写入数据,从而避免额外的冗余拷贝开销.

当写入的数据恰好整页对齐,HDPM 直接通过写时复制机制存储数据:首先从空闲空间中分配 1 个新的数据页,将数据写入新数据页,并通过硬件指令将所写数据从 CPU 缓存逐出,从而实现数据的持久化存储;然后将映射表对应映射条目的第 1 个指针指向新的数据页,使得新写入数据可以被其他读者读取到;最后将旧数据页释放回收.可以注意到,该过程未发生任何额外的拷贝开销,且数据可原子地从旧版本切换到新版本(原子地修改指针),崩溃一致性得到保障.

当写入数据不能整页对齐,HDPM 则将数据追加到日志结构中.图 4 描述了日志结构的布局,它是由若干个持久性内存块串接而成.每个日志结构的起始位置预留了一个日志头指针,用于指向最后一个有效日志项.特别地,日志项使用了倒序排布方式,即元数据放置在数据之后,用于支持逆向扫描文件读取(2.3 节).每个日志项的元数据包含 2 个字段,分别为日志项包含的数据在文件数据块中对应的偏移和大小.向日志结构追加数据时,首先在日志末尾初始化一个日志项,将数据、元数据拷贝到该日志项,然后更新日志头指针指向最新的位置.当日志空间不够时,则重新申请一个新的内存块,并链接到日志尾部.通过引入该日志结构,在发生非对齐写操作时依旧无需执行额外的拷贝操作,且更新原子性得到保证.值得注意的是,当一个数据页发生过一次或多次非对齐写之后,如果紧接着执行一次整页对齐的写操作,则整个日志空间将变为旧数据,因此日志空间可以被释放回收.



每一页都有单独的log,容易造成数据的随机写。特别是log的head指针是小数据,小数据的随机写性能较差,没有很好的局部性

Fig. 4 Layout of log-structure

图 4 日志结构布局图

HDPM 引入的日志结构与传统日志式文件系统(log-structured file system, LFS)[13]完全不同:LFS 将整个磁盘组织成一个全局大日志,所有文件数据均追加到日志末尾,相反,HDPM 将每个数据页组织成一个小日志.在未对某数据页发生非对齐写时,其对应的日志结构为空,因此,HDPM 不引入额外的空间浪费.该设计具有 3 方面的优势:

1) 多核并发.传统的全局大日志在慢速磁盘场景下可以工作得很好,这是因为磁盘内部的旋转式机械部件天然地支持顺序访问,这与顺序地将数据追加到日志尾部的访问模式完全吻合.然而,持久性内存可以支持更高的并发,且随机和顺序访问性能差异不明显,如果依旧使用全局大日志抽象持久性内存空间,多个线程将从同一个日志头部竞争存储空间,严重影响扩展性.相反,HDPM 为每个数据页

可以在内存中维护一个LRU buffer，每次读页数据时，在内存中重构，这样后面的读就不用多次log跳跃索引，不过这样显然会增加NVM的写入流量，适当增加DRAM buffer的大小，可以吸收大部分更新

配备一个日志结构,多个线程之间的竞争概率将大幅降低,能够更好地利用持久性内存高带宽特性.

2) 文件读取性能影响小.传统日志式文件系统或需要在内存中构建复杂的索引结构,用于辅助查找日志中的目标数据,或需要多次日志跳跃查找,这极大地限制了文件读取的性能.HDPM 的混合数据页管理机制无需额外的索引机制,且避免了大量的日志扫描,仅需从一个小日志中便可拼合出最新数据.

3) 垃圾回收.传统的日志式文件系统需要全局扫描磁盘检测盘内垃圾占比,并挑选垃圾比例最高的数据块进行合并整理,以腾出空闲空间.此过程将引入大量的扫描操作,垃圾回收开销较高,且影响前台应用的运行.相比之下,为每个数据页配备一个日志结构具有更好的数据局部性:执行垃圾回收时只需挑选出最长的日志,将内部有效数据合并,并写到写时复制页即可,无需额外的腾挪过程,垃圾回收的过程将更加简洁高效(2.4 节).

全盘扫描肯定不行，但可以优化，内存中维护GC链表，每线程一个

2.3 基于逆向扫描的文件读取

HDPM 的另一个设计原则是在引入日志结构后不影响正常的文件读取性能.根据 2.2 节所述,数据页在写时复制页和日志结构中的新旧程度具有如下关系:日志结构尾部的数据比日志结构头部的数据要新,且日志结构头部的数据比写时复制页中的数据要新.因此,一种直观的数据读取方案为:首先将写时复制页中的数据拷贝到用户缓冲区,然后在日志中从头至尾扫描,将相关的日志项拷贝到缓冲区,直至日志末尾.然而,当该数据页发生大量的重复写入时,日志结构中将有多个日志项的数据彼此重叠,从而造成额外的拷贝开销,降低文件读取性能.

为解决该问题,HDPM 引入一种逆向扫描的文件读取方案.HDPM 将日志结构的内存块链接方式设计为双向链表,并将日志项进行倒序排布,用于支持文件读取时从日志尾部向头部扫描,从而避免上述的额外拷贝开销.具体而言,首先根据读请求的大小及偏移确定是否包含当前日志项,如果包含,则将其拷贝到用户缓冲区,并记录已读取区间;否则,从日志结构前移一个日志项并重复判断.若当前日志项在读取区间内,且之前已经记录该区间已经被成功读取,则直接忽略该日志项,这是因为日志尾部的数据是最新的版本,无需额外拷贝旧版本数据.当记录的已读取区间完全覆盖了读请求的覆盖区间,则停止扫描并返回数据.若扫描完整个日志结构依旧无法覆盖整个读区间,则进一步从写时复制页中读取相应数据.

2.4 多重垃圾回收机制

为避免各数据页对应的日志结构增长过快,占用过多的持久性内存空间,需要及时将日志结构进行合并整理,消除文件旧版本数据,并回收持久性内存空间.然而,传统的日志清理机制需要通过扫描存储设备探测垃圾数据,并通过锁机制强制阻塞前台服务,从而实现安全的垃圾回收过程,该过程会严重影响系统性能.为此,HDPM 提出一种多重垃圾回收机制,其包括在正常读写流程中的机会性垃圾回收和后台免锁模式的垃圾回收.

如 2.2 节所述,在正常写操作过程中,当发生整页对齐写操作时,可直接将对应日志空间进行回收,这将一定程度上缓解日志的空间占用.进一步,HDPM 还在读操作中引入机会性垃圾回收流程.在 HDPM 执行正常的数据页读取时,首先检查其对应的日志结构长度,若该日志结构超过某阈值,则对该数据页进行完整的逆向读取,整理出最新版本的数据页信息,并写回到写时复制页,同时回收相应的日志空间.此过程会一定程度影响当前读操作的性能,然而,通过定期整理过长的日志结构,有助于提升后续的数据读取性能.

上述前台垃圾回收机制只能回收部分的持久性内存空间,因此还需要专门的后台垃圾回收机制.为避免传统垃圾回收机制阻塞文件系统正常读写流程,HDPM 引入一种基于 Epoch 技术[14]的免锁垃圾回收方法.如 2.2 节所述,HDPM 在更新文件时,可以通过修改指针将数据从旧版本原子地切换到新版本,因此,后台垃圾回收机制同样可以使用原子指令修改指针,用于日志结构的合并整理及回收.然而,后台的垃圾回收机制使用免锁机制回收持久性内存块,而不阻塞文件系统读写流程,有可能前台的读线程正在读取已经被回收的持久性内存区域.此过程中如果已被回收的内存块迅速被重新分配,并用于服务新的数据写入,则读线程可能读到错误的数据.Epoch 技术正是用于避免上述问题,其具体实现流程如下.

HDPM 引入一个全局的 Epoch 计数器,其初始值为 0,同时为每个正在访问文件系统的进程分配一个本地计数器.每当各进程发起系统调用开始访问文件系统之前,首先将全局计数器与本地计数器的值进行比对,如果不相等,则将本地计数器设置为与全局计数器相等.后台垃圾回收线程将所有的空闲持久性内存块通过 2 个空闲链表进行管理,分别对应到当前的 Epoch 值和上一个 Epoch 值.后台

线程定期执行垃圾回收:每次执行垃圾回收时,首先将全局寄存器加 1,然后将回收的空闲块增加到当前活跃的空闲链表中.当所有进程的本地计数器都为最新值时,上一次活跃的空闲链表内所包含的数据页将不再有进程访问,此时可以安全地被重新分配并使用.在实际回收过程中,后台线程根据各日志结构的长度确定回收的优先级别,即优先回收长度较大的日志结构.

通过上述 Epoch 机制,垃圾回收过程完全无锁化,因而不会阻塞前台系统执行,其对系统性能影响极小.

2.5 讨 论

HDPM 引入了混合数据页管理、基于逆向扫描的文件读取机制以及多重垃圾回收机制.这些技术一定程度给文件系统带来了额外的计算开销,但是,这些开销相比于其带来的性能提升十分微小.例如,引入的混合数据页管理只需要 CPU 通过写入的尺寸和偏移进行一次额外的逻辑判断(纳秒量级);同时,维护一个日志结构的开销与维护传统的 CoW 机制的开销相当,而这种做法的效果却是有效降低持久性内存的数据写入量(微妙量级).同样,逆向读取避免数据的额外拷贝(微妙量级),而反向扫描的开销所占总体执行时间(纳秒量级)的比例很小.多重垃圾回收机制是 HDPM 相比于 NOVA 额外引入的开销.然而,多重垃圾回收能够同时在前台和后台进行.其中,大量的垃圾回收操作能够直接被读操作吸收,即在读取文件数据页的过程中自然地完成垃圾回收操作,而非常少的一部分垃圾数据留给了后台进行回收.同时,HDPM 仅在空闲(系统负载低)的情形下执行后台回收.综上,多重垃圾回收技术基本不影响文件系统的前台处理性能.

3 系统实现

HDPM 包含的技术可以广泛地应用到现有的持久性内存文件系统中.作为示例,本文基于 NOVA[10] 实现了 HDPM 的各技术细节.

NOVA 将每一个 inode 组织为一个日志结构,因此,HDPM 依然沿用 NOVA 的设计,仅将映射表的每个条目改为使用 2 个指针.NOVA 在常规情况下并不会扫描 inode 的日志结构进行元数据查询,而是同时在 DRAM 中为每个文件构建一棵 Radix 树,用于映射表的索引查询.因此,HDPM 也同时在每个 Radix 树的叶子节点额外增加一个指针,用于指向各数据页的日志结构.

为保证文件系统崩溃一致性,HDPM 需要严格按照顺序持久化已经更新的数据.与 NOVA 相似,HDPM 使用 Non-temporal 指令持久化数据,使用 clflushopt 指令持久化指针.其中,Non-temporal 指令可以绕过 CPU 缓存,实现数据的即时持久.在更新数据和更新指针之间,添加 mfence 指令,用于强制执行持久化的顺序性,保证数据的一致性更新.

4 实 验

本节将从微观测试、宏观测试、关键技术 3 个方面对比 HDPM 和现有系统,并详细分析其性能差异.

4.1 实验平台

本实验使用的实验平台配置信息如表 1 所示.特别地,本实验使用了 Intel 最近推出的 Optane DC 持久性内存作为非易失内存存储介质,其单条容量为 256 GB,总容量达 768 GB.为兼容持久性内存设备,相应的 CPU、主板、BIOS 均为专门配备.Optane 可配置为 2 种工作模式,分别为 APP-Direct 模式和内存模式.由于文件系统需直接管理持久性内存空间,因此本实验采用 APP-Direct 模式.HDPM 基于 NOVA 进行改造,因此本实验将重点与 NOVA[10] 进行性能对比.作为参考,HDPM 同时也与 Ext4,Ext4-DAX[6] 等传统的文件系统进行了对比.

Table 1 Platform Configuration

表 1 实验平台配置信息

Item	Configuration
CPU	Intel® Xeon® Gold 6240M×2
Memory	Samsung 32 GB×6
PM	Intel Optane DC Persistent Memory 256 GB×3
Disk	1TB Seagate,256 GB Intel SSD×2
OS	Ubuntu 19.04,Linux 5.1

4.2 微观测试

图 5 展示了在不同的 I/O 尺寸下 HDPM 及对比系统的写入操作延迟.从图 5 可知,Ext4-DAX 的访问延迟最低,而 Ext4-JBD 的延迟最高.这是因为 Ext-DAX 直接对文件数据进行原地更新,没有额外的崩溃一致性保护,其开销最低;而 Ext4-JBD 需要对每次元数据和数据操作记录日志,这将引入额外的数据持久化开销和软件执行开销,其延迟远高于其他 3 个系统.NOVA 和 HDPM 均引入了文件数据的原子更新策略,延迟保持在较低水平.特别地,HDPM 在 8 B 写入时,延迟相比于 NOVA 降低了

58%,而在 4 KB 整页写入时 2 个系统表现出相近的访问延迟.在写入尺寸较小的情况下,HDPM 无需额外的数据拷贝操作,只需要在日志结构中追加 8 B 的日志项即可,而 NOVA 基于 CoW 技术,需要将整个 4 KB 数据页从旧的地方拷贝到新的地方,其持久化开销远高于 HDPM.当写入粒度为 4 KB 时,二者需要写入的数据大小相同,因而访问延迟无显著差异.同时,可注意到在 I/O 尺寸发生变化时,NOVA 的访问延迟不升反降,这主要是因为在执行写时复制流程中,需要从旧页中拷贝的数据量越来越小,而这部分数据极大可能不在 CPU 缓存中,因此其造成缓存缺失的开销更小.

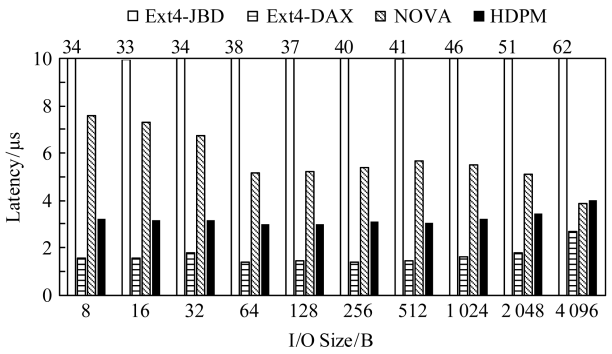


Fig. 5 Latency of write syscall with varying I/O sizes

图 5 写系统调用在不同 I/O 尺寸下的延迟

进一步,图 6 展示了在不同 I/O 尺寸下,各系统的读操作延迟.具体的测试方法为,首先对一个文件随机写入一百万次非对齐的数据块,然后按照不同的 I/O 尺寸从文件中顺序读取.从图 6 中可以观察到,专门针对持久性内存优化过的文件系统(如 NOVA,HDPM)的读延迟均低于传统文件系统(如 Ext4-JBD,Ext4-DAX),这主要得益于 NOVA 精简了软件栈,其执行逻辑更加简明高效;另一方面,HDPM 几乎表现出与 NOVA 相同的读延迟.这主要因为 HDPM 引入了基于逆向扫描的读取技术,

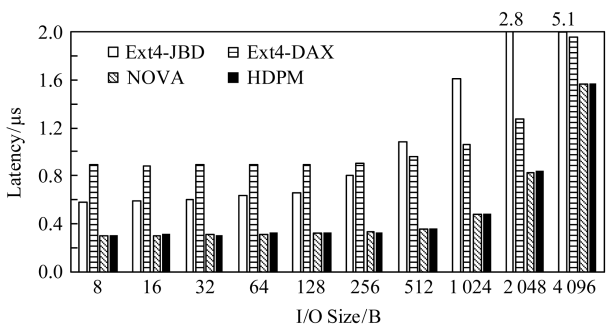


Fig. 6 Latency of read syscall with varying I/O sizes

图 6 读系统调用在不同 I/O 尺寸下的延迟

在此过程中 HDPM 避免了为拼接日志项引入的额外拷贝开销,同时,逆向搜索带来的计算开销相比于数据拷贝开销基本可以忽略不计.特别地,4.4 节进一步分析了该关键技术.

4.3 Filebench

为测试 HDPM 在真实负载下的性能表现,本文选取了 Filebench[15] 中的 2 个代表性负载 Webproxy 和 Varmail 进行测试.本文采用了与文献[10]中相同的配置参数进行测试:Webproxy 负载的文件平均大小为 64 KB,读操作平均 I/O 大小为 1 MB,写操作为 16 KB,读写比例为 5∶1,总文件数量为 10 万个.Varmail 负载的文件平均大小为 32 KB,读操作平均 I/O 大小为 1 MB,写操作为 16 KB,读写比例为 1∶1,总文件数量为 10 万个.可见,Webproxy 为读密集型负载,而 Varmail 则包含大量小尺寸同步写操作.在测试过程中,通过逐渐增加客户端线程数量来测试 Filebench 的总体吞吐.我们发现,当线程增加到 10 之后,总性能不再上升,这一方面是因为 Optane 持久性内存的写入带宽有限,在 10 个线程后其带宽趋于饱和;另一方面,Filebench 原本针对磁盘设计,其内部同于性能统计、数据测试的模块本身设计扩展性不好.因此,本实验未给出 10 个线程之后的实验数据.

从图 7 中可见,Ext4-JBD 的性能最差,而其他 3 个系统的总体吞吐十分接近,这主要是因为 Webproxy 为读密集负载,系统间性能差异不明显.

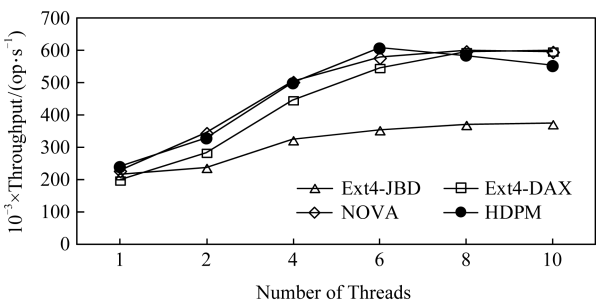


Fig. 7 Webproxy throughput with varying number of threads

图 7 不同线程下 Webproxy 吞吐

同时,从图 8 中可以发现,在写密集应用下,NOVA 的总体吞吐远高于 Ext4-DAX 和 Ext4-JBD,这主要是因为 NOVA 专门针对持久性内存重新设计了软件栈以及数据管理策略.而 HDPM 则表现出最好的性能,在 10 个线程下 HDPM 的总体吞吐比 NOVA 高 33%.这主要是因为 Varmail 包含了较多非对齐写入操作,HDPM 有效降低了非对齐写入

操作引入的拷贝开销.同时可以注意到,在线程数量增大时,HDPM 相比于 NOVA 的性能优势更加明显.这主要受 Optane DC 带宽的影响.在线程数量较少情况下,Optane DC 的带宽不是瓶颈,HDPM 与 NOVA 之间的性能差异完全取决于各操作延迟.由于 Filebench 本身具有较大的软件开销,文件系统层次的操作延迟占比不高,HDPM 优化技术带来的延迟降低归结到系统层面效果不明显.但当线程数量上升之后,HDPM 与 NOVA 之间的性能差异变大.HDPM 通过混合的数据页管理能够更加高效地处理非对齐小写,从而有效降低了数据总写入量,节省了 Optane DC 的写入带宽,此时 Optane DC 的带宽决定了总吞吐,因而 HPDM 拥有更加明显的性能优势.

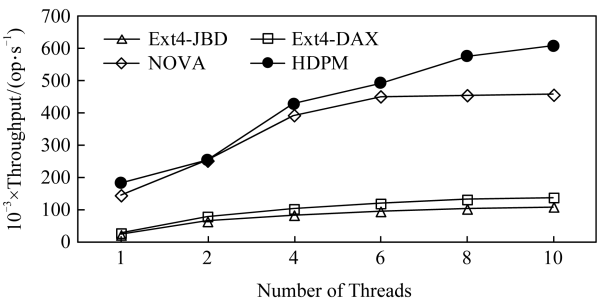


Fig. 8 Varmail throughput with varying number of threads

图 8 不同线程下 Varmail 吞吐

4.4 关键技术分析

为理解在读操作中引入的逆向扫描技术带来的性能优势,我们关闭了这一技术,并实现了 HDPM w/o opt 版本,该方案将之前的逆向扫描改为顺序扫描,该扫描过程在遇到多次重复写时,会依次将目标数据拷贝到用户缓冲区,从而造成额外开销.本实验中,依次对文件中的某数据块重复写入 N 次,然后测试 2 个版本的系统的读取性能,其评测结果如图 9 所示.可发现,在重复写入次数变化时,HDPM 的读取延迟基本不发生变化,而 HDPM w/o opt 的读取延迟显著上升.因此,在数据读取中,有效降低额外的拷贝次数对提升系统整体读取性能有极大的帮助.

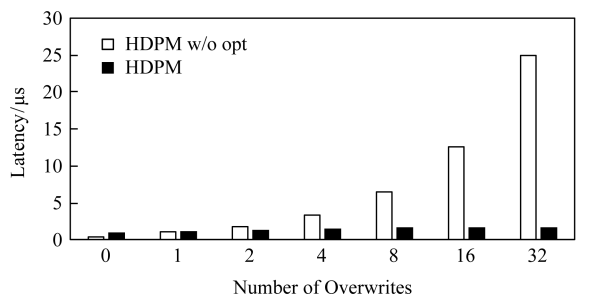


Fig. 9 Read latency with varying number of overwrites

图 9 不同重复写次数下的读取延迟

5 相关工作

持久性内存直接通过内存总线接入处理器,存储访问延迟极低,且处理器可以按照字节粒度访问持久性内存.这些硬件上的差异促使了新的文件系统设计.

1) 一致性保障.微软研究院于 2019 年提出的字节寻址的持久性内存文件系统 BPFS[8],通过短路影子页(short-circuit shadow paging)方式提供数据的原子性更新,同时提出一种硬件 Epoch 提交策略降低缓存刷新的开销.英特尔公司于 2014 年提出的 NVM 直写文件系统 PMFS[9],采用原子的原地更新及细粒度日志机制保证元数据更新的原子性,通过 undo 日志和写时复制的混合方式保证数据的一致性.加州大学圣地亚哥分校将传统日志结构文件系统进行重新设计和扩展,研制出日志结构持久性内存文件系统 NOVA[10].NOVA 将每个文件 inode 组织为一个日志,因此,其对单个 inode 的修改操作可以直接在日志尾部原子追加;重命名等操作通常涉及对多个日志结构的更新,因此 NOVA 还采用轻量级的日志技术保证多操作的原子性.与 PMFS 相同,NOVA 对文件数据使用写时复制技术.虽然上述系统在不同程度上结合了持久性内存的字节寻址特性,并通过不同的手段解决了文件系统一致性管理.然而,上述所有系统均使用了基于页粒度的写时复制技术,这会造成额外的拷贝开销.北京航空航天大学提出的 noseFS[16] 则通过构建额外的 B^+ 树维护更细粒度的持久性内存页,从而降低写时复制机制引入的拷贝开销,然而,B^+ 树的维护管理同样造成额外的时间及空间开销.

2) 降低软件开销.操作系统管理的页缓存在持久性内存中将造成冗余的数据拷贝,严重影响性能.因而,Ext4,BtrFS 等传统文件系统均兼容了直接访问模式(direct access, DAX).通过这种方法,应用程序可以直接访问非易失内存中存储的文件数据,而不需要将数据额外拷贝到页缓存中.PMFS, NOVA 等专门为持久性内存设计的文件系统则通过内存映射的方式绕开了文件系统页缓存.然而,操作系统本身依旧十分笨重,其系统调用引入的现场切换开销,虚拟文件系统(virtual file system, VFS)

引入的软件开销等在面向持久性内存时也变得愈发严重.因此,维斯康星大学麦迪逊分校于2014年提出一种用户态的持久性内存文件系统Aerie[17],从而使得应用程序可以直接在用户态访问文件数据,而不引入额外的操作系统开销.Strata[18]同样也是一个用户态文件系统,不同的是,Strata同时管理多种存储设备(硬盘、固态硬盘、持久性内存等),通过合理的调度,Strata在性能、容量等方面均表现优异.

3)多核扩展性.在多核场景下,如何设计可扩展的文件系统以充分利用持久性内存的高吞吐特性变得十分重要.NOVA在改造传统日志式文件系统时,重点考虑了多核扩展性.NOVA将每个inode组织为一个日志,而不是仿造传统方式构建一个全局大日志,通过这种方式,有效避免了并发向单个日志追加记录难扩展的问题.另外,NOVA仅将元数据放入日志结构,而数据部分则单独管理,这有效降低了垃圾回收的开销.最后,NOVA将空闲空间切分到不同的处理器核心,不同线程在分配内存空间时,仅从本地空闲空间进行分配,从而避免了额外的全局锁开销.通过上述设计,NOVA具有极强的扩展性.HDPM也采用了类似的设计理念,通过多个日志结构降低竞争.

6 结 论

持久性内存展现出与传统存储设备(如硬盘、固态硬盘)完全不同的硬件特性,这为构建高效的持久性内存存储系统提出了新的挑战.本文提出一种混合数据页管理机制HDPM,通过选择性使用写时复制机制和日志结构管理文件数据,充分发挥持久性内存字节可寻址特性,从而避免了传统单一模式在遇到非对齐写或者小写造成的写放大问题.为保持读性能不受影响,HDPM引入逆向扫描机制,在重构数据页时不引入额外数据拷贝.HDPM还提出一种多重垃圾回收机制,在不同阶段、不同时机进行高效的日志合并整理.实验显示,HDPM能够显著提升写入性能.

参 考 文 献

[1] Lee B C, Ipek E, Mutlu O, et al. Architecting phase change memory as a scalable dram alternative [C] //Proc of the 36th Annual Int Symp on Computer Architecture. New York: ACM, 2009: 2-13

[2] Qureshi M K, Srinivasan V, Rivers J A. Scalable high performance main memory system using phase-change memory technology [C] //Proc of the 36th Annual Int Symp on Computer Architecture. New York: ACM, 2009: 24-33

[3] Zhou Ping, Zhao Bo, Yang Jun, et al. A durable and energy efficient main memory using phase change memory technology [C] // Proc of the 36th Annual Int Symp on Computer Architecture. New York: ACM, 2009: 14-23

[4] Baek I G, Lee M S, Seo S, et al. Highly scalable nonvolatile resistive memory using simple binary oxide driven by asymmetric unipolar voltage pulses [C] // Proc of IEEE Int Electron Devices Meeting. Piscataway, NJ: IEEE, 2004: 587-590

[5] Intel. Optane DC Persistent Memory [OL]. (2019-04-01) [2019-10-12]. https://www.intel.com/content/www/us/en/architecture-and-technology/optane-dc-persistent-memory.html

[6] Matthew W. Support ext4 on NV-DIMMs [OL]. (2014-02-25) [2019-10-12]. https://lwn.net/Articles/588218/

[7] Sweeney A, Doucette D, Hu Wei, et al. Scalability in the XFS file system [C] //Proc of USENIX Annual Technical Conf. Berkeley, CA: USENIX Association, 1996: No.15

[8] Condit J, Nightingale E B, Frost C, et al. Better I/O through byte-addressable, persistent memory [C] //Proc of the 22nd ACM SIGOPS Symp on Operating Systems Principles. New York: ACM, 2009: 133-146

[9] Dulloor S R, Kumar S, Keshavamurthy A, et al. System software for persistent memory [C] //Proc of the 9th European Conf on Computer Systems. New York: ACM, 2014: No.5

[10] Xu Jian, Swanson S. NOVA: A log-structured file System for hybrid volatile/non-volatile main memories [C] //Proc of the 14th USENIX Conf on File and Storage Technologies. Berkeley, CA: USENIX Association, 2016: 323-338

[11] Lu Youyou, Shu Jiwu, Sun Long. Blurred persistence in transactional persistent memory [C] //Proc of the 31st Symp on Mass Storage Systems and Technologies. Piscataway, NJ: IEEE, 2015

[12] Izraelevitz J, Yang Jian, Zhang Lu, et al. Basic performance measurements of the Intel Optane DC persistent memory module [J]. arXiv preprint, arXiv:1903.05714, 2019

[13] Rosenblum M, Ousterhout J K. The design and implementation of a log-structured file system [J]. ACM Transactions on Computer Systems, 1992, 10(1): 26-52

[14] Hart T E, McKenney P E, Brown A D, et al. Performance of memory reclamation for lockless synchronization [J]. Journal of Parallel and Distributed Computing, 2007, 67(12): 1270-1285

[15] Tarasov V, Zadok E, Shepler S. Filebench: A flexible framework for file system benchmarking [J]. Login: The USENIX Magazine, 2016, 41(1): 6-12

[16] Yang Fan, Kang Junbin, Ma Shuai, et al. A highly non-volatile memory scalable and efficient file system [C] //Proc of the 36th IEEE Int Conf on Computer Design. Piscataway, NJ: IEEE, 2018: 431-438

[17] Volos H, Nalli S, Panneerselvam S, et al. Aerie: Flexible file-system interfaces to storage-class memory [C] //Proc of the 9th European Conf on Computer Systems. New York: ACM, 2014: No.14

[18] Kwon Y, Fingler H, Hunt T, et al. Strata: A cross media file system [C] //Proc of the 26th Symp on Operating Systems Principles. New York: ACM, 2017: 460-477

Chen Youmin, born in 1993. PhD candidate. His main research interests include distributed systems and storage systems.

Zhu Bohong, born in 1993. Master candidate. His main research interests include storage systems and file systems.

Han Yinjun, born in 1977. Senior Engineer. His main research interests include nonvolatile memories, file system, and database system.

Tu Yaofeng, born in 1972. Professor. His main research interests include cloud computing and distributed systems.

Shu Jiwu, born in 1968. Professor and PhD supervisor. Outstanding member of CCF. His main research interests include nonvolatile memory systems and technologies, network (/cloud/big data) storage systems, storage security and reliability, and parallel and distributed computing.