基于持久内存的文件系统综述

刘一航[1)]

[1)]（华中科技大学计算机科学与技术学院湖北武汉430074)

摘　要　近年来，新型非易失性存储器引起了广泛关注，其具有速度快、容量大、字节编址且提供非易失性存储的特性。为了能将它高效地应用于系统，面临很多的挑战，因此需要改变传统存储结构和去除不利于发挥新型存储设备特性的因素。这其中包括系统软件栈的开销、存储设备接口的开销等。这篇综述从微观和宏观层面探讨了英特尔新Optane DIMM的性能特性和特点。注意其性能是相对于传统DRAM或其他过去用来模拟NVM的方法特有的特殊方式。另外，本文也介绍了Kuco，一种新颖的直接访问文件系统架构，其主要目标是提高系统的可扩展性。Kuco利用三种关键技术－－协作索引、两级锁定和版本控制读取－－将耗时的任务(如·径名解析和并发控制)从内核转移到用户空间，从而避免了内核处理瓶颈。在Kuco的基础上，论文作者提出了KucoFS的设计和实现，并通过实验证明了KucoFS在广泛的实验范围内具有良好的性能。重要的是，在元数据操作方面，KucoFS比现有的文件系统有更好的扩展性，可以达到一个数量级，并且可以充分利用设备带宽进行数据操作。

关键词　持久内存；非易失存储；文件系统；Kuco；

Survey on File System Based on Persistent Memory

Liu Yihang[1)]

[1)](School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, Hubei 430074))

Abstract　In recent years, a new type of non-volatile memory has attracted widespread attention, which has the characteristics of fast speed, large capacity, byte addressing and providing non-volatile storage. This review discusses the performance characteristics and characteristics of Intel’s new Optane DIMM from both the micro and macro levels. In addition, this article also introduces Kuco, a novel direct access file system architecture whose main goal is to improve the scalability of the system. On the basis of Kuco, the author of the paper proposed the design and implementation of KucoFS. The important thing is that in terms of metadata operations, KucoFS has better scalability than existing file systems and can reach an order of magnitude.

Keywords　Persistent memory; Non-Volatile Storage; File System; Kuco

1 背景介绍

在过去的十年里，研究人员一直期待着商业上可用的、可扩展的非易失性主存(NVMM)技术的到来，这种技术可以提供“字节寻址”存储，使数据能够在断电后还能保存。随着英特尔Optane DC持久内存模块(Optane DIMMs)的问世，研究者使用广泛的微基准测试、基准测试和应用程序，通过收集的数据表明，以前研究人员的许多关于NVDIMMs的行为和性能的假设是不正确的。人们普遍认为，NVDIMMs的特性与基于DRAM大致相似，但性能较低(即更高的延迟和更低的带宽)。测试发现，与DRAM性能相比，Optane DIMM性能更强烈地依赖于访问粒度大小、访问类型(读与写)、模式和并发程度。这篇综述介绍了Optane内存在微基准测试和应用程序上的行为和性能的详细评估，并提供了具体的、可操作的指南，指导程序员应该如何调整他们的程序，以充分利用这些新的内存。

同时，持久内存的出现增加了重新设计高效文件系统的重要性。在过去的十年里，系统社区已经提出了许多文件系统，如BPFS、PMFS和NOVA，以减少由传统文件系统架构引起的软件开销。但是，这些PM感知的文件系统是操作系统的一部分，应用程序需要进入内核才能访问它们，其中系统调用和虚拟文件系统(VFS)仍然会产生不可忽略的开销。在这方面，最近的研究提出在用户空间中部署文件系统直接访问文件数据(即直接访问)，从而利用PM的高性能。尽管做出了这些努力，我们发现另一个重要的性能指标——可扩展性——仍然û有得到很好的解决，特别是当多核处理器满足快速PM时。NOVA通过对内部数据结构进行分区和避免使用全局锁，提高了多核可扩展性。但是，我们的评估表明，由于VFS层的存在，它仍然不能很好地解决这个问题。更糟糕的是，一些用户空间文件系统设计通过引入集中式组件反而进一步加剧了可扩展性问题。例如，Aerie通过向具有更新元数据权限的可信进程(TFS)发送昂贵的进程间通信来确保文件系统元数据的完整性。另一个例子是Strata，它避免了通过在PM日志中直接记录更新，在正常操作中涉及到一个集中的进程，但需要一个KernFS将它们(包括数据和元数据)应用到文件系统，这将导致多一次数据复制。两个文件系统中的可信进程(如TFS或KernFS)也负责并发控制，这不可避免地成为高并发下的瓶颈。

在这篇综述中，我们介绍一种内核-用户空间协作体系结构(Kuco)来重新审视文件系统设计，以实现直接访问性能和高可扩展性。Kuco遵循典型的客户端/服务器模型，包含两个组件，包括一个用户空间库(Ulib)，用于提供基本文件系统接口，以及一个位于内核中的线程(Kfs)，用于处理由Ulib发送的请求并执行关键更新。受分布式文件系统设计的启发，例如AFS，它通过最小化服务器负载和减少客户端/服务器交互来提高可扩展性，Kuco提出了一种新颖的任务划分，它将大多数任务转移到Ulib，以避免Kfs可能出现的瓶颈。为了实现元数据的可扩展性，我们引入了一种协作索引技术，允许Ulib在向Kfs发送请求之前执行·径名解析。通过这种方式，Kfs可以使用由Ulib提供的预先定位的地址直接更新元数据项。对于数据的可扩展性，我们首先提出了一个两级锁机制来协调并发写入共享文件。具体来说，Kfs管理ÿ个文件的写租约，并将其分配给打算打开该文件的进程。相反，这个进程中的线程在用户空间中使用范围锁来锁定文件。其次，我们引入了一个版本化的读协议来实现直接读，即使不与Kfs交互，尽管存在并发写入器。Kuco还包括加强数据保护和提高基线性能的技术。Kuco以只读模式将PM空间映射到用户空间，以防止有bug的程序破坏文件数据。用户空间的直接写是通过一个三相写协议来实现的。在Ulib写入文件之前，Kfs通过切换页表中的权限位，将相关的PM 页面从只读切换到可写。在编写文件时，还使用了一种预分配技术来减少Ulib和Kfs之间的交互。

利用Kuco架构，构建了一个名为KucoFS的PMfile系统，它获得了用户空间直接访问性能，同时提供了高可扩展性。我们用文件系统基准和真实世界来评估KucoFS应用程序。评估结果表明，在高竞争的工作负载下(例如，在同一个目录中创建文件或在一个共享文件中写入数据)，KucoFS比现有文件系统的扩展性好一个数量级，并且在低竞争的情况下提供略高的吞吐量。对于正常的数据操作，也会影响PM设备的带宽。总的来说，本篇综述的大致内容可以概括为以下几点：

(1) 具体介绍PM的相关特性及其与DRAM在读写速度，访问延迟等方面的差异。

(2) 对最先进的PM感知的文件系统进行了深入的分析，并总结了它们在解决软件开销和可扩展性问题上的局限性。

(3) 介绍了一种用户空间-内核协作架构Kuco，其通过三个关键技术（协作索引、两级锁定和版本化读取）以实现高可扩展性。

(4) 基于Kuco架构，介绍了一个名为KucoFS的PMfile系统，实验表明，KucoFS在元数据操作方面实现了高达一个数量级的可扩展性，并充分利用PM带宽进行数据操作。

2 原理和优势

2.1 Optane Memory

Optane DIMM是第一款可扩展的商用NVDIMM。与现有的存储设备相比，Optane DIMM具有更低的延迟、更高的读带宽，并且采用基于内存地址的接口，而不是基于块NVMe接口。与DRAM相比，它具有更高的密度和持久性。Optane DIMMs位于内存总线上，并连接处理器的集成内存控制器(iMC)(图1)。英特尔的Cascade Lake处理器是第一个(也是唯一一个)支持Optane DIMM的cpu。在这个平台上，ÿ个处理器包含一个或两个处理器模，它们组成独立的NUMA节点。一个处理器有两个iMC，一个iMC支持三个通道。因此，总的来说，一个处理器模可以在其两个iMC上支持总共6个Optane DIMMs。

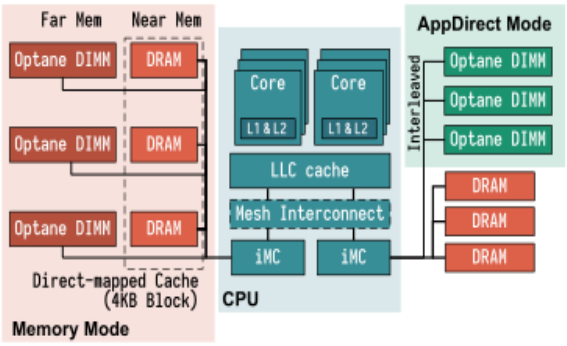

图1

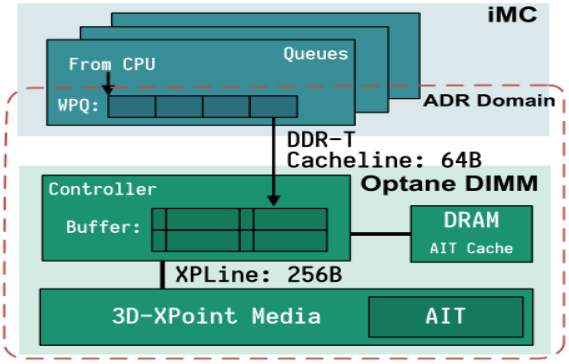

图2

为了确保持久性，iMC位于异步DRAM刷新(ADR)域内——英特尔的ADR功能确保到达ADR域的CPU存储在掉电后仍能存活(即，在等待时间¡ 100 µ s)内刷新到NVDIMM。iMC维护ÿ个Optane的读写挂起队列(RPQs和WPQs)(图2)，ADR域包括WPQs。一旦数据到达WPQs, ADR确保iMC在断电时将更新刷新到3D-XPointý体。ADR域不包括处理器缓存，因此存储只有在到达WPQs时才持久。iMC使用缓存线(64B)粒度的DDR-T接口与Optane DIMM通信，该接口与DDR4共享接口，但使用不同的协议，允许异步命令和数据计时。内存访问NVDIMM首先到达控制器(图2中XPController)，它协调对Optane介质的访问。与ssd类似，Optane DIMM为磨损均衡和坏块管理执行内部地址转换，并为该转换维护一个地址间接表(AIT)。在地址转换之后，就会发生对存储ý体的实际访问。由于3D-XPoint物理ý体访问粒度为256字节(图2中XPLine)，XPController将较小的请求转换为较大的256字节访问，当小型存储变为读-修改-写操作时，会导致写放大。XPController有一个小的写合并缓冲区(XPBuffer)，用于合并相邻的写。重要的是要注意，到达XPBuffer的所有更新已经是持久的，因为XPBuffer驻留在ADR内。

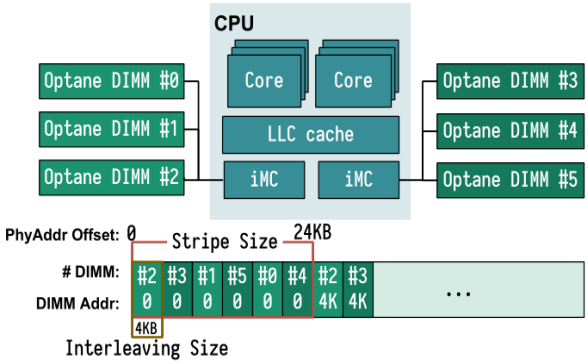

图3

OptaneDIMM有两种工作模式(图1): Memory和App Direct。内存模式使用Optane扩展主内存容量而不需要持久性。它将Optane DIMM与传统的DRAM DIMM结合在同一个内存通道上，作为NVDIMM的直接映射缓存。缓存块大小为64b, CPU的内存控制器透明地管理缓存。CPU和操作系统只是将Optane DIMM视为主内存的一个较大(易失性)部分。应用程序直接模式提供持久性，不使用DRAM缓存。Optane DIMM是一个独立的、持久的内存设备。文件系统或其他管理层提供对持久数据的分配、命名和访问。在这两种模式下，Optane内存可以(可选)交叉使用通道和内存(图3)。在我们的平台，唯一支持的交错大小是4 kB，这确保了对单个页面的访问落在单个内存中。配置6根内存条，大于24kb的访问将访问所有内存。

2.2 Kuco

现有的许多PM文件系统大致分为三类。首先基于内核的文件系统，应用程序通过捕获到内核

中进行数据和元数据操作来访问它们。第二，用户空间文件系统(例如Aerie，Strata和ZoFS)。其中，Aerie依赖于可信进程(TFS)来管理元数据，并保证元数据的完整性。TFS还通过分布式锁服务协调对共享文件的并发读写。相反，Strata允许应用程序直接将更新附加到ÿ个进程的日志中，但需要后台线程(KernFS)将记录的数据异步地消化到存储设备中。ZoFS避免使用集中式组件，并允许用户空间应用程序在新硬件特性的帮助下直接更新元数据。需要注意，Aerie、Strata和ZoFS仍然依赖于内核来执行粗粒度的分配和保护。第三，混合文件系统(例如，SplitFS和我们提议的Kuco)。SplitFS提供了用户空间库和现有内核文件系统之间的粗粒度分割。它完全在用户空间中处理数据操作，并通过Ext4file系统处理元数据操作。表1总结了现有的pm感知文件系统，以及它们在各个方面的表现。

1）多核可扩展性。NOVA是一种最先进的pm内核文件系统，经过精心设计，通过引入逐核分配器和逐inode日志来提高可扩展性。然而，VFS仍然限制了它对ij些操作的可扩展性。我们通过在Intel Optane dcpmm上部署NOVA实验证明了这一点并使用多个线程来创建、删除、或重命名同一目录中的文件。Aerie依赖于一个集中的TFS来处理元数据操作和执行并发控制。

		NOVA	Aerie/Strata	ZoFS	SplitFS	KucoFS
Category		Kernel	Userspace		Hybrid	
❶ Scalability	Metadata	Medium (§5.2.1)	Low (§5.2.1)	Medium (Fig. 7g in [13])	Low (§5.2.1)	High (§5.2.1)
	Read	Medium (§5.2.2)	Low (§5.2.2)	High	Low (journaling in Ext4)	High (§5.2.2)
	Write	Medium (§5.2.3)	Low (§5.2.3)	Medium (Fig.7f in [13])		High (§5.2.3)
❷	Softeware overhead	High	Low	Medium (`sigsetjump`)	Medium (metadata)	Low
❸ Other issues	Avoid stray writes	✓	✗	✓	✗	✓
	Read protection	POSIX	Partition	Coffer	POSIX	Partition
	Visibility of updates	Immed-iately	After batch/ After digest	Immed-iately	append: After sync	Immed-iately
	Hardware required	None	None	MPK	None	None

图4

2）软件开销。在内核中放置文件系统会面临两种类型的软件开销，即系统调用和VFS开销。我们仍然通过分析NOVA来研究这种开销，在NOVA中我们收集了常见文件系统操作的延迟分解。ÿ个操作都用一个线程在100 万个文件或目录上执行。我们从图4中得到两个观察结果。首先，系统调用占用总执行时间的21%(例如stat和open)。另外，在一个进程进入内核后，操作系统可能会安排其他任务，然后将控制权返回给原来的进程。因此，系统调用为对延迟敏感的应用程序带来了额外的不确定性。其次，Linux内核文件系统是通过覆盖VFS函数来实现的，而VFS会导致不可忽视的开销。

3）其他问题。首先，使用不当的指针可能导致写入错误的位置并损坏数据，这被称为偏离写入。Strata向用户空间应用程序公开ÿ个进程的操作日志和DRAM缓存(包括元数据和数据)。Aerie和SplitFS映射文件系统映像的子集到用户空间。因此，ż然的写操作很容易破坏这些区域的数据，而且这种破坏在NVM中是永久的，即使是在重新引导之后。总而言之，现有的文件系统设计很难实现高可扩展性和低软件开销，这促使我们引入Kuco架构。

3 研究进展

3.1 延迟测试

读写时延是内存技术的关键参数。我们通过对平均延迟计时来测量读延迟对于单个8字节的指令加载顺序和随机内存地址。为了消除缓存和排队的影响，我们清空CPU队列，并在测量之间发出内存栅栏。对于写操作，我们将缓存行加载到缓存中，然后测量以下两个指令序列中的一个的延迟:64位存储、一个clwb和一个mfence。这些测量结果反映了软件所看到的负载和存储延迟，而不是这些底层存储设备的负载和存储延迟。对于负载，延迟包括来自片上互连、iMC、XPController和实际3DXPointý体的延迟。我们的结果(图5)显示Optane的读延迟比DRAM高2-3倍。我们认为这主要是由于Optane的介质延迟较长。Optane存储器也比DRAM存储器更依赖模式。随机与顺序的间隙为DRAM的20%，而Optane内存的80%，这种间隙是XPBuffer的结果。对于存储，一旦数据到达iMC的ADR，内存存储和栅栏指令就会提交，所以DRAM和Optane都显示类似的延迟。非时态存储比使用缓存刷新(clwb)的写操作开销更大。一般来说，Optane的延迟变化是非常小的，除了极少数的“异常值”(我们将在下一节中研究)。Optane dimm的顺序读延迟有更高的差异，因为第一个缓存线访问将整个XPLine加载到XPBuffer中，接下来的三次访问将读取缓冲区中的数据。

在许多系统中，内存和存储系统的尾部延迟严重影响响应时间和最坏情况下的行为。在我们的测试中，我们观察到负载和存储的延迟非常一致，除了一些“异常值”，当访问集中在一个“热点”时，存储的数量会增加。图6测量了尾部延迟和访问位置之间的关系。该图显示了作为热点大小的函数的99.9、99.99和最大延迟。我们为ÿ个热点大小分配一个循环缓冲区，并使用一个线程发出2,000万个64字节的写操作。异常值的个数256B

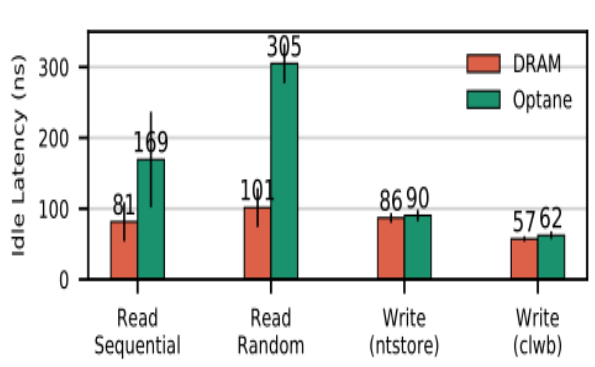

图5

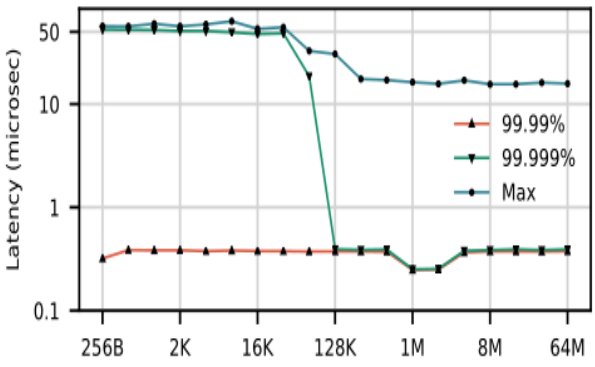

图6

2K 16K 128K 1M 8M 64M Hotspot Region 0.1 1 10 50 Latency (microsec) 99.99% 99.999% Max图3:尾延迟一个实验显示了按顺序写一个小内存区域(Hotspot)的尾延迟。Optane内存有罕见的“异常值”，少量写操作需要50 µ s才能完成(比通常的延迟时间增加了100倍)。(特别是超过50 µ s的)随着热点大小的增加而减少，DRAM不存在。这些峰值非常罕见(仅占0.006%)，但它们的延迟比普通Optane访问高出2个数量级。

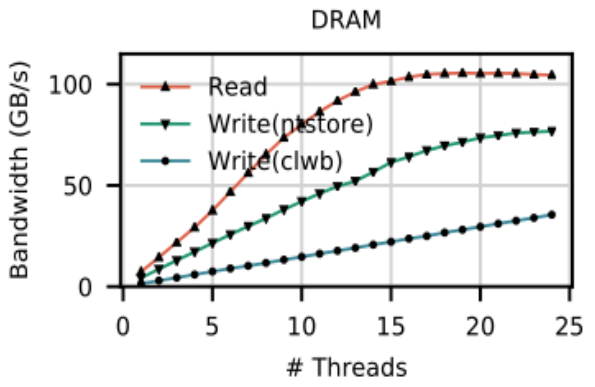

图7

3.2 带宽测试

详细的带宽测量对应用程序设计人员是有用的，因为它们提供了对内存技术如何影响总体系统吞吐量的洞察。首先，我们测量了在不同并发级别下随机和顺序读写的Optane和DRAM带宽。图7显示了256 B访问粒度的顺序访问在不同线程数下获得的带宽。

我们展示了装载和存储(Write(ntstore))，以及缓存的刷新写(Write(clwb))。所有实验都使用AVX-512指令。最左边的图绘制了交错DRAM的性能，而最中间和最右边的图绘制了非交错和交错Optane的性能。在非交错测量中，所有访问都击中一个内存。图5显示了性能如何随访问大小而变化。这些图描绘了给定大小的随机访问的聚合带宽。数据表明，DRAM的带宽既高于Optane，又可预测地(且单调地)随着线程数的增加而扩展，直到达到饱和的DRAM的带宽，这主要与访问大小无关。Optane的结果是非常不同的。

首先，对于单个内存，最大读带宽是maxi-的2.9×其中，DRAM的读写带宽差较小(1.3×)。其次，除了交错读取之外，随着线程数的增加，Optane性能是非单调的。对于非交错(即单内存)的情况，性能在1到4个线程之间达到峰值，然后逐渐下降。交错将store+clwb的峰值推至12个线程。我们将回到线程数上升对性能的负面影响。第三，在256b 下的随机访问Optane带宽很差。这个“膝盖”对应于XPLine的大小。DRAM带宽在8 kB(典型的DRAM页面大小)时不会出现类似的“膝”，因为打开DRAM页面的成本要比访问Optane的新页面低得多。交错(将访问分散到所有六个内存上)进一步增加了复杂性:图4和图5测量了跨越六个交错的nvdimm的带宽。交错使峰值读写带宽分别提高了5.8×和5.6×。这些加速与系统中的内存数量相匹配，并突出了Optane的ý个内存带宽限制。图中最显著的特征是4 kB时的性能下降——这种下降是由iMC争用引起的突发效应，当线程执行接近交错大小的随机访问时，性能会最大化。

3.3 Kuco

Kuco的核心思想是客户端和服务器之间的细粒度任务分工和协作，大部分负载都被转移到客户端，以避免服务器成为瓶颈。

图8显示了Kuco体系结构。它遵循客户机/服务器模型，其中包括两个部分，包括一个用户空间库和一个全局内核线程，这两个部分分别称为Ulib和Kfs。Ulib通过协作索引与Kfs交互;Read:通过版本读取直接访问;Write:直接访问，采用三相写协议，两级锁并发控制。Ulibfirst和不同的Ulib实

例(即应用程序)通过单独的内存消息缓冲区与Kfs交互。为了保护文件系统元数据不被破坏，Kuco不允许应用程序直接更新元数据;相反，这些请求被发布到Kfs，然后Kfs代表它们更新元数据。Kuco通过细粒度任务划分和Ulib与Kfs之间的协作提供了高可扩展性。

对于元数据的可扩展性，Kuco整合了协作索引机制，将·径名遍历作业从Kfs卸载到用户空间。ulib不会直接向Kfs发送元数据操作(如创建或解除链接)，而是在用户空间中查找所有相关的元数据项，然后在发送请求之前将这些信息封装在请求中。因此，Kfs可以直接对给定的地址执行元数据修改。为了数据的可扩展性，使用了两级锁机制来处理并发写入共享文件。具体来说，Kfs使用基于租赁的分布式锁来解决不同应用程序(或进程)之间的写冲突。来自同一个进程的并发写操作使用一个纯用户空间范围锁进行序列化，可以在不需要Kfs的情况下获取该锁。Kuco进一步引入了版本化的读取技术来执行用户空间中的文件读取。

通过在数据块映射(将逻辑文件数据映射到物理PM地址)中添加额外的版本位，Kuco可以读取一致版本的数据块，而不需要与Kfs交互来获取锁，尽管还有其他并发写入器。为了进一步防止有bug 的程序破坏文件数据，PM空间以只读模式映射到用户空间。Kuco通过在内核中使用三相写协议将Kfs放置在只读地址上，从而允许用户空间直接写。在Ulib写文件之前，Kfs首先修改页表中的权限位，将相关的数据页从只读切换到可写。为了进一步减少编写文件时Ulib和Kfs之间的交互次数，Kuco采用了预分配(pre-allocation)，这样Ulib可以从Kfs分配比预期更多的空闲页面。除了写保护Kuco的PM空间被划分成不同的分区树，这些分区树作为读保护的最小单元。通过将Kuco应用于名为KucoFS的文件系统中，将所有技术结合在一起，获得了直接访问性能，提供了高可扩展性，并确保了内核级的数据保护。

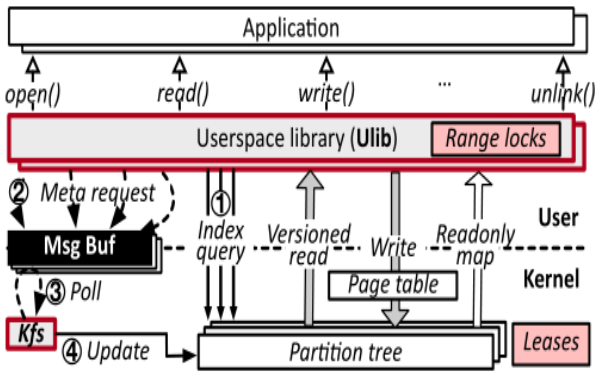

图8

3.4 KucoFS

在KucoFS中，创建操作需要向Kfs发布请求，所以我们选择这个操作来显示协作索引的效果。FxMark通过让ẏ个客户端线程在私有目录(即低共享级别)或共享目录(即中共享级别)中创建10个kfile来评估创建操作。如图9a和9b所示，在比较的文件系统中，KucoFS表现出最高的性能，而且无论共享级别如何，它的吞吐量都不会崩溃。XFS-DAX、Ext4-DAX和PMFS使用全局锁在共享日志中执行元数据日志记录，这导致它们的可扩展性很差。NOVA通过避免全局锁(例如，它使用ẏ个inode日志并对其空闲空间进行分区)，在低共享级别下显示了出色的可伸缩性。然而，所有的内核文件系统都不能在中共享级别下扩展，因为VFS需要在创建文件之前锁定父目录。

SplitFS依赖于Ext4来创建文件，这导致了它的可伸缩性很低。Aerie支持通过批处理(通过降低可视性)将创建文件的元数据更新同步到TFS，因此它可以实现与KucoFS相当的性能。当线程数增加时，Aerie 无法正常工作。而KucoFS的吞吐率仅随介质共享级别的提高而略有下降，其吞吐率比其他文件系统高一个数量级，比ZoFS高3倍。我们从以下几个方面来解释KucoFS的高可扩展性。首先，在KucoFS中，所有元数据更新都被委托给Kfs，因此Kfs可以在û有任何锁定开销的情况下更新它们。其次，通过将索引任务转移到用户空间，Kfs只需要执行轻量级工作。更大的工作负载。此外，我们通过扩展工作负载大小来衡量KucoFS在数据容量方面的可伸缩性。

具体来说，我们让ẏ个线程创建100万个文件，比FxMark中的默认大小大100倍，结果如图9c所示。与工作负载较小的结果相比，KucoFS的吞吐量下降了28.5%。这主要是因为当文件数量增加时，文件系统需要更多的时间在父目录中找到合适的插入槽。即便如此，KucoFS仍然比其他文件系统的性能好一个数量级。冲突处理。当发生冲突时，KucoFS要求Kfs退回并重试，这可能会影响整体性能。在这方面，我们还测试了KucoFS在处理相互冲突的操作时的行为。具体来说，如果同一个文件不存在，我们使用多个线程并发地创建它，或者在它已经创建时删除它。我们收集这些成功创建和删除的吞吐量，结果如图10所示。作为比较，NOVA的结果也显示在图中。

我们可以看到，KucoFS的吞吐量比NOVA高2.4倍。在NOVA中，线程需要在创建或删除文件之前获取锁。更糟糕的是，如果创建或删除失败，其他并发线程将被不必要地阻塞，因为锁不能保护有效的操作。相反，在KucoFS中，

线程可以向Kfs发送创建或删除请求，而不会被阻塞，Kfs负责确定这个操作是否可以成功处理。此外，由于Ulib已经在请求中提供了相关的地址，Kfs 可以直接使用这些地址来验证元数据项，这带来的开销并不大。我们还通过与禁用此优化的KucoFS的一种变体进行比较，来衡量协作索引的好处。图10显示了使用不同数量的客户机测量create吞吐量的结果。

我们作出以下观察。首先，在单线程评估中，协作索引并不有助于提高性能，因为将元数据索引任务从Ulib移回Kfs并不能减少ÿ个操作的总体延迟。其次，当客户端线程数量增加时，我们发现协作索引可以将吞吐量提高55%。由于KucoFS只允许Kfs代表多个Ulib实例更新元数据，理论上的吞吐量限制是Tmax = 1/L (Ops/s，其中L是Kfs处理一个请求的延迟)。因此，卸载机制通过缩短ÿ个请求的执行时间(即L)来提高性能。

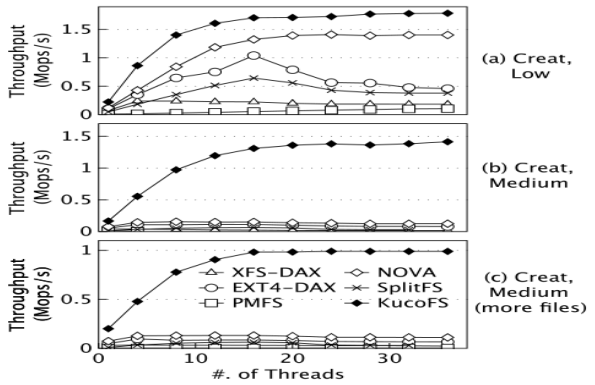

图9

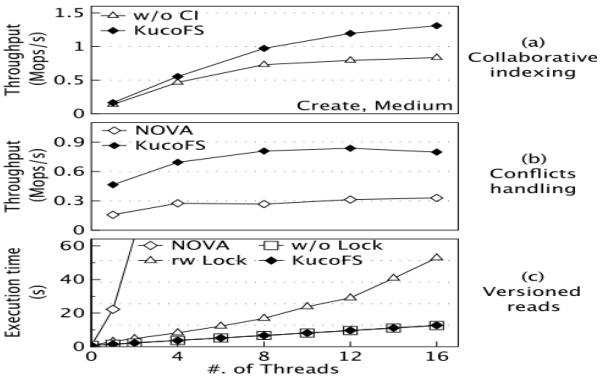

图10

3.5 KucoFS性能表现

我们使用Filebench作为一个宏基准来评估KucoFS的性能。图11显示了两种工作负载设置(与NOVA论文中的设置类似)以及使用1和16个线程的实验结果(使用Filebench增加更多线程不会提高吞吐量)。我们可以看到，首先，在所有评估的工作负载中，KucoFS的性能最高。在使用Fileserver工作负载进行单线程评估时，其吞吐量分别比XFS、Ext4、PMFS、NOVA和Strata高2.5×,1.9×，比Varmail工作负载高3.2×,5.6×,1.9×,1.45×和1.13×。对于以读为主的工作负载，KucoFS还显示出略高的吞吐量。性能的提升主要来自于KucoFS的直接访问特性。Strata也受益于直接访问，并在大多数工作负载中表现第二好。我们还观察到，KucoFS的设计非常适合Varmail工作负载。

这是可以预期的:Varmail经常创建和删除文件，因此它会产生更多的元数据操作，并更频繁地发出系统调用。如前所述，KucoFS消除了操作系统部分的开销，并且更擅长处理元数据操作。此外，Strata比NOVA显示出更高的吞吐量，因为Varmail中的文件I/O很小。Strata只需要在操作日志中追加这些小的更新，大大减少了写放大。其次，KucoFS在处理并发工作负载方面做得更好。在Fileserver工作负载下有16个客户端线程，KucoFS 的性能比XFS-DAX好4.4倍，比PMFS好1.2倍，比NOVA好27%。对于Varmail工作负载，性能提升更为明显:它的平均性能比XFS-DAX和Ext4-DAX高10倍。

有两个原因促成了它的良好性能:首先，KucoFS结合了协作索引等技术，使Kfs能够提供可伸缩的元数据访问性能;第二, KucoFS通过让ÿ个客户端管理私有的自由数据页面来避免使用全局锁。NOVA还展示了良好的可伸缩性，因为它使用ÿ个inode的日志结构，并对空闲空间进行分区，以避免全局锁。

Workload	Fileserver		Webserver		Webproxy		Varmail	
R/W Size	16 KB/16 KB		1 MB/8 KB		1 MB/16 KB		1 MB/16 KB	
R/W Ratio	1:2		10:1		5:1		1:1	
Total number of files in each workload is 100K.								
Threads	1	16	1	16	1	16	1	16
XFS-DAX	39K	127K	121K	1.35M	192K	863K	99K	319K
Ext4-DAX	52K	362K	123K	1.33M	316K	2.50M	57K	135K
PMFS	72K	317K	110K	1.25M	218K	1.54M	169K	1.06M
NOVA	71K	537K	133K	1.43M	337K	3.02M	220K	2.04M
Strata	75K	-	105K	-	420K	-	283K	-
KucoFS	99K	683K	141K	1.48M	463K	3.22M	320K	2.55M
ÿ	32%	27%	6%	3%	10%	7%	13%	24%

"ÿ" indicates the performance improvement over the 2nd-best system.

图11

3.6 应用测试

Redis导出了一组api，允许应用程序处理和查询结构化数据，并使用文件系统进行持久的数据存储。Redis有两种方法来持久地记录数据:一种是将

操作记录到仅追加文件(AOF)中，另一种是使用异步快照机制。

本文仅用AOF模式对Redis进行评估。图12显示了使用不同值大小的12字节键的SET操作的吞吐量。对于小值，Redis在KucoFS上的吞吐量平均比PMFS、NOVA和Strata高53%，比XFS-DAX和Ext4-DAX高76%。这与之前append 的结果一致。由于对象大小较大，KucoFS的吞吐量比其他文件系统略高，因为大部分时间都花在写数据上。请注意，Redis是一个单线程应用程序，因此KucoFS在8KB对象(约800MB/s)下实现100 Kops/s的吞吐量是合理的。SplitFS很擅长处理追加操作，因为它在用户空间中处理数据平面操作。然而，它的性能仍然不如KucoFS，因为Redisÿ次添加新数据时都会发布fsync来刷新AOFfile。因此，SplitFS需要捕获到内核中来更新元数据，这再次导致了VFS和系统调用开销。

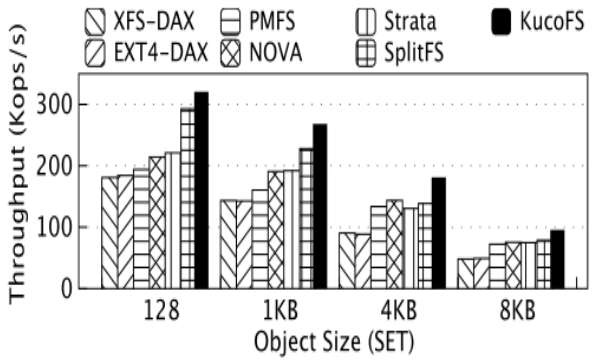

图12

4 总结与展望

本篇综述先描述了英特尔的新Optane DIMMs在微观和宏观层面的性能基准。在这样做的过程中，我们为程序员提取了可操作的指导方针，以充分利用这些设备的优势。这些设备具有介于传统存储设备和内存设备之间的性能特征，但它们也呈现出有趣的性能特性。我们相信，这些设备将有助于扩展可用内存的数量，并提供低延迟存储。另外，基于NVM的特性，介绍了一个名为Kuco的内核和用户级协同架构，它在用户空间和内核之间展示了细粒度的任务划分。在Kuco的基础上，进一步介绍了一个名为KucoFS的PM文件系统，实验表明，KucoFS提供了高效和高可扩展的性能。

参考文献

[1] Jian Yang, Juno Kim, and Morteza Hoseinzadeh; Joseph Izraelevitz, Boulder; Steve Swanson. An Empirical Guide to the Behavior and Use of Scalable Persistent Memory.//Proceedings of the 18th USENIX Conference on File and Storage Technologies (FAST’20) February 25–27, 2020, Santa Clara, CA, USA

[2] Youmin Chen, Youyou Lu, Bohong Zhu, Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau?, Jiwu Shu. Scalable Persistent Memory File System with Kernel-Userspace Collaboration.//Proceedings of the 19th USENIX Conference on File and Storage Technologies. February 23–25, 2021

[3] Ian Neal, Gefei Zuo, Eric Shiple, and Tanvir Ahmed Khan, Youngjin Kwon, Simon Peter, Baris Kasikci. Rethinking File Mapping for Persistent Memory.//Proceedings of the Proceedings of the 19th USENIX Conference on File and Storage Technologies. February 23–25, 2021