持久内存系统及应用综述

柴世欣[1)]

[1)](华中科技大学 计算机科学与技术学院, 武汉 430074)

摘 要 持久内存是一种新兴的存储设备，具有字节可寻址和非易失性，同时有着类似 DRAM 的性能。其构想曾在过去很长一段时间给予了存储界新的希望，而 19 年傲腾持久内存的推出和商业落地更是激发了学术界和工业界对于持久内存研究的热潮。近些年，大量工作基于持久内存特性展开，涉及编程模型和库、文件系统和数据库等一众领域。本文将梳理一些近期持久内存系统及应用的相关工作，并进行分析归纳。最后在傲腾持久内存停产的大背景下，展望持久内存系统及应用的未来研究方向。

关键词 持久内存；字节可寻址；非易失性；傲腾持久内存；编程模型；文件系统；数据库

Persistent Memory Systems and Applications: A Survey

Shixin Chai[1)]

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

Abstract Persistent memory is an emerging storage device that is byte addressable, non-volatile, and has DRAM-like performance. Its concept has given new hope to the storage industry for a long time in the past, and the launch and commercial implementation of Optane persistent memory in 2019 has inspired an upsurge in persistent memory research in academia and industry. In recent years, a large amount of work has been carried out based on persistent memory features, involving programming models and libraries, file systems, databases and other fields. This article will sort out some recent work related to persistent memory systems and applications, and analyze and summarize them. Finally, under the background of the discontinuation of Optane persistent memory, we look forward to the future research directions of persistent memory systems and applications.

Key words Persistent memory; byte addressable; non-volatile; Optane persistent memory; programming model; file system; database

1 引言

持久内存（Persistent Memory，PM），也可以称为非易失性内存（Non-volatile Memory， NVM）的构想和出现是学术界和工业界新的希望。它的字节可寻址性和持久性将 DRAM 和持久性存储设备的两大优点相结合，对于存储软件栈可以说带来了翻天覆地的变化。同时，它还有类似于 DRAM 的读写性能，这将大大提升计算机的存储性能。

因为持久内存的诸多特性和好处，对于它的探索从很早就开始了。起先是大量基于持久内存硬件的研究，即，如何构建持久内存。这包括大量技术，譬如相变存储器、忆阻器、3D X Point 等技术。在硬件的研究热潮结束后，大量在持久内存上软件的研究也不断涌现。软件本文主要关注于持久内存上的软件栈构建：

（1）文件系统。持久内存的到来改变了存储界的生态，首当其冲的就是其基础软件设施：文件系统。如何在持久内存上高效构建文件系统，而不局限于之前基于块设备的文件系统，是一项重大的问题和挑战，学术界和工业界早已进行了大量相关探索。

（2）系统和库。如何重新构建 OS 和库，使得用户能够充分利用持久内存特性来进行计算机的基本使用和编程；同时非易失性内存带来崩溃一致

柴世欣，男，2000年生，硕士，主要研究领域为计算机系统结构、存内计算. E-mail: shixinchai@hust.edu.cn.

性的挑战也促使着系统的发展。

（3）数据库及其相关领域。数据库领域总是和存储领域密不可分。存储生态的改变也预示着数据库生态的变化。数据库分为许多部分，上层包括一些优化器、解析器等不受影响，其底层存储引擎则受到冲击，大量基于持久内存优化的数据库索引等工作在近些年涌现出来。

当然，与持久内存相关的领域与方向即为广阔，不可枚举，鉴于时间和本人能力有限，本文无法完全囊括。本文主要就围绕上述三个方向简要描述持久内存系统与其上的应用，给出分析和展望。本文的主要贡献如下：

（1）介绍持久内存和相关技术，并抛出问题和挑战。

（2）概括了近些年有关与持久内存系统和应用的相关工作，便于有志于该领域和方向的研究者快速了解概况。

（3）总结和分析研究现状，并给出未来展望。

本文结构组织如下：首先，给出构建持久内存系统和应用的问题和挑战；之后，概括相关的研究现状，并作出分析；最后，给出未来持久内存系统和应用的研究方向，并做出总结。

2 问题与挑战

本章首先描述持久内存，之后描述在持久内存上构建系统和应用的问题与挑战。

2.1 持久内存

持久内存（Persistent Memory，PM），是最近存储系统新的展望和方向。持久内存，顾名思义兼具外存的持久性和 DRAM 的字节可寻址性。同时，它既没有传统外存随机访问的性能劣势，又不像 DRAM 需要通过刷新来保持数据。正是因为这样的特性，它在数据库、高性能计算等一系列领域都具有极高的应用前景。

在没有出现真正的硬件之前，已经有大量持久内存的技术研究。比如，相变存储器（Phase Change Memory，PCM）、忆阻器（Resistive Memory，ReRAM）以及 3DXPoint。根据当时的一些研究，持久内存具有与 DRAM 相近的读性能，但是写性能较差，并且写耐久度不佳。

2019 年，英特尔傲腾 DC 持久内存（Intel Optane DC Persistent Memory，以下简称傲腾持久内存）的出现终于将真实的持久内存硬件带给学术界和工业界。图 1.1 是英特尔认为的加入持久内存后的内存/存储层次结构。根据论文中的大量实验，傲腾持久内存的出现了与先前持久内存相关研究不同的新特性，包括顺序访问和随机访问性能不一致、读性能远不如 DRAM、虽然字节可寻址但写入大小为 256B 等。基于傲腾持久内存的表现，本文将持久内存的特性总结如下：

（1）读性能远高于写性能，写性能和写耐度差。持久内存的读写性能差距在 3 倍以上，同时其读写性能仍不及 DRAM，但大幅度优于外存设备。

（2）字节可寻址。持久内存支持高性能的随机访问，并且可以直连内存总线。

（3）访问粒度为 256B。也就是说，小于 256B 的写入将产生高开销的写入放大。

（4）顺序访问比随机访问性能更好。傲腾持久内存包含对合并逻辑的访问以合并重叠的内存请求，使得顺序访问不承担与 256 字节访问大小相关的写入放大成本。

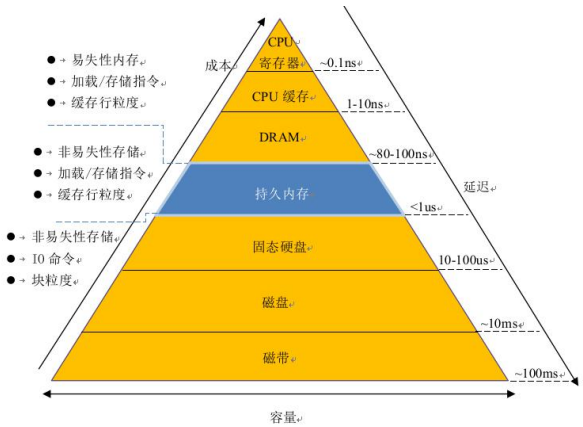

图 1 内存/存储层次结构金字塔

同时，由于持久性的要求，在使用持久内存时往往需要大量调用缓存刷新指令，包括 clflush、clflushopt，以及 clwb 等。这是因为，对于支持异步 DRAM 刷新（Asychronous DRAM Refresh，ADR）的英特尔 CPU，缓存并不属于持久性域。数据只有写入到内存控制器的写入挂起队列才能在掉电后不丢失，而缓存中的数据是易失的。频繁的缓存刷新将更频繁地影响持久内存地写入性能，一定程度也影响读取性能。

2022 年 7 月，傲腾持久内存宣布停产，这宣告了持久内存商业化的第一次失败，持久内存的发展进入后傲腾时代。但这次失败并不意味着持久内存这一构想是错误的。英特尔在商业模式上的错误选择是其失败的一大原因。恰恰相反，英特尔的退出

了给予了其他厂商更多的机会来制造和研发持久内存，日后持久内存产品迎来百花齐放的第二春看起来页并不遥远。对于学术界，持久内存的相关研究也需要更近一步，让持久内存能更好地落地和应用。

2.2 相关的问题和挑战

鉴于持久内存的一众独特特性，现在逐条说明在其上构建系统和应用的问题和挑战：

（1）读写性能不一致而导致的性能挑战。许多研究和傲腾持久内存都表明，持久内存的写入性能远不如读取性能。这对持久内存上的软件设计提出了相关的要求。针对特殊的数据结构或者场景，能否尽可能减少写入来提升性能，这是一个相当困难、复杂和需要精细思考的问题。

（2）其余的性能问题，包括写入粒度、顺序和随机访问特性等。由于傲腾持久内存的停产，许多傲腾持久内存上专有的性能特征已经不再被研究和做对应优化，但其仍然在此提及。这是因为，未来新的持久内存设备仍然会有其独特特性，考虑去研究这些特性，或者参考研究对应特性的工作仍然有一定的价值。

（3）数据持久性和崩溃一致性保证。持久内存作为一种非易失性的存储介质，其上的数据应该具有持久性。但是由于 CPU 对于持久内存的仅保证 8 字节的原子性写入，这就对数据写入的一致性做出要求。而大量的数据或者数据结构都远大于 8 字节。假如在更新数据中途发生系统崩溃，数据将处于一种中间状态并不可恢复，这对于系统可靠性有重大影响，因此如何保证崩溃一致性是大量系统和应用所面临的难题，包括文件系统、数据库索引等。同时由于持久内存的写入性能低、缓存易失需要显式的缓存刷新指令和内存屏障等情况下，如何在不特别影响性能的情况下做到崩溃一致性是又是一个需要精细思考的任务。

（4）编程模型设计。持久内存不同于传统的块设备和 DRAM 设备，因此，如何组织新的编程模型并引入系统，使之对于原先的软件能够兼容，并且也能够更好地引入新的应用，是一个值得思考的问题。

（5）持久内存库的设计。持久内存编程往往涉及底层和系统编程，需要考虑大量底层的细节，包括缓存、内存屏障等；加之崩溃一致性的考虑使得程序员编写程序的负担极具增加。将底层语义和崩溃一致性保证封装为高层次的库不仅有助于提升编程效率，也将有效指导实践，提升程序性能。如何封装这样的库也是一项重大挑战。

（6）安全性问题。持久内存（PMem）作为一种非易失性存储介质，带来了新的安全挑战。其数据持久性可能导致长期存储敏感数据，增加了数据泄露的风险，特别是在未经加密的情况下，可能存在物理访问 PMem 进行数据窃取的风险。此外，PMem 上的数据在断电后仍然存在，因此即使数据被删除，也可能仍然可以恢复，增加了隐私泄露的风险。针对 PMem 的持久性特性，开发者需要关注设计和实现持久性数据结构时的一致性问题，确保数据在写入时的一致性，以防止数据损坏或不一致。在应对这些安全挑战时，加密敏感数据、实施严格的访问控制、考虑持久性数据的完整性和物理安全防护等措施是至关重要的，有助于降低 PMem 带来的安全风险，保护系统和数据的安全性。

（7）持久内存的成本问题。根据傲腾持久内存的定价，持久内存廉价于 DRAM，但是远远昂贵于传统的块设备。如何利用 PM 做到极值的性价比，即每 bit 最高性价比，也是一项极具挑战的是工作。

2.3 本章小节

本章主要介绍了持久内存的相关背景知识，并详细描述了在持久内存上构建系统和应用的问题和挑战。

3 研究现状

本章就现有的持久内存系统和应用相关的研究现状做出说明，主要包括持久内存编程模型和库、持久内存文件系统以及持久内存数据库索引三个大方向。

3.1 持久内存编程模型和库

近年来，有许多持久内存编程库的提出，如 PMDK（Persistent Memory Development Kit）等。它们都需要程序员付出相当大的努力来进行编程并保证应用程序的性能和崩溃一致性。*Zhuque: Failure is Not an Option, it's an Exception* 提出了一种全新的编程模型 Zhuque，它保证了 WPP（Whole Process Persistent）。这个编程模型能保证某些特定进程的持久性，并且能让许多程序不加修改地直接运行在其系统中，就能直接获得持久性的好处。

大量的持久内存编程库的发布提升了程序员的开发效率，但仍然需要程序员对程序的精细控

制。许多库和编程模型采用以下三种模式之一：（1）基于事务；（2）基于 FUSE；（3）全系统持久。基于事务的编程模型具有加/解锁限制，几乎不能灵活地使用细粒度锁；基于 FUSE 的程序虽然有良好的扩展性和兼容性，但是需要动态的跟踪运行时状态，开销比较大；全系统持久对于系统做了过大的调整，有些进程并不需要持久性的保证，这可能导致不必要的性能下降。

WPP 基于芯片制造商的故障刷新保证，即如果出现断电，缓存中的数据会通过一个低级别的中断刷新到主存中。Zhuque 将进程中的所有内存分配转换到在 PMEM 中进行；如果系统出现断电故障，进程会接收到操作系统发出的信号以保存程序状态；在系统崩溃重启后，程序将从故障中断点处继续执行。

图 2 是 Zhuque 的整体架构。

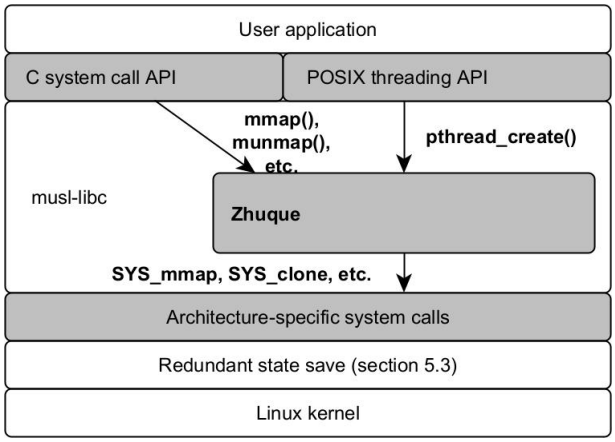

图 2 Zhuque 整体架构图

Zhuque 通过修改了 C 绑定库 libc 以实现 WPP——拦截和转换关于内存、线程和文件管理的 API 调用，比如 mmap、pthread_XXXd 等；这使得 Zhuque 实际上对应用程序透明，用户仅仅只需要设置一个环境变量就可以开启 Zhuque，使得程序无需修改就可保证持久性和一致性。

在系统的常规执行阶段，Zhuque 对系统进行了改变。最主要的就是对于内存的使用变化（全部迁移到 PMEM 上）。对于匿名（anonymous）mmap() 返回 PMEM，而对于静态内存，将将私有的、可写的文件映射转换到 PMEM 上的映射。同时，由于操作系统内核是易失性的，因此，在每个内核入口处，Zhuque 保存体系结构状态（寄存器文件等）到 PMEM 上。在系统崩溃时，Zhuque 将保存易失性的体系结构状态，这包括寄存器文件、FP/向量上下文等；同时在刷新缓存的固件触发中断保存这些状态到内存中。这样，与进程相关的所有数据都具有持久性和崩溃一致性。断电后，如果机器未损坏则，系统在重启后进行恢复，即进程重启时 Zhuque 检测到崩溃状态并执行恢复，这包括：恢复用户地址空间；恢复操作系统专有的状态：Zhuque 跟踪线程和文件描述符，并在重启时重新创建他们；恢复体系结构状态（包括栈指针和程序计数 PC）。这相当于重新启动启动。最后，如果应用程序提供了故障处理程序（failure handler），Zhuque 在进程继续执行前运行它。

Zhuque 在性能方面表现优异，相对于现有的 PMEM 库，它在所有基准测试中的平均加速比为 5.24 ×（相对于 PMDK）、3.01 ×（相对于 Mnemosyne）、5.43×（相对于 Atlas）和 4.11×（相对于 Clobber-NVM）。更重要的是，与现有系统不同，Zhuque 对应用程序实现并发的方式没有任何限制，这使更新版本的 Memcached 能够在 Zhuque 上运行，相对于现有最快的持久性实现，吞吐量提高了 7.5×以上。

这个工作为持久内存的研究提供了独特的见解。之前的大量工作都在精心设计如何保证崩溃一致性，而这篇工作从基础设施层面解决了这个问题，有很大的参考价值和实用意义。

3.2 持久内存文件系统

持久内存文件系统是持久内存的一个非常重要的应用场景。近年来有大量的持久内存文件系统陆续提出，包括内核文件系统和用户空间文件系统。本次介绍的这篇 *MadFS: Per-File Virtualization for Userspace Persistent Memory Filesystems* 就是一篇介绍用户空间文件系统的文章。

用户空间通常可以获得比内核文件系统更好的性能优势，原因在于，持久内存可以直接从用户空间访问，而无需内核参与，但大多数持久文件系统仍然在内核中执行元数据操作以确保安全，并依赖内核进行跨进程同步。该论文介绍了一种每文件虚拟化（Per-File Virtualization）机制，其中虚拟化层在用户空间中实现了一套完整的文件功能，包括元数据管理、崩溃一致性和并发控制，并提出了 MadFS，一个库 PM 文件系统，将嵌入的元数据作为紧凑的日志进行维护。

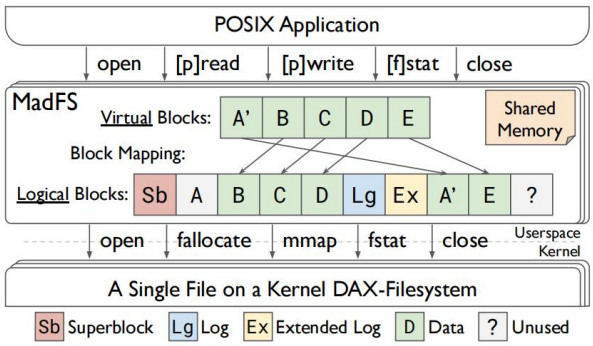

图 3 MadFS 整体架构

图 3 是 MadFS 整体架构图。上层是 MadFS 库对于文件系统 syscal 的拦截。作者注意到并非所有的文件元数据都需要由内核维护，因此，MadFS 将不敏感的元数据嵌入到文件中以进行用户空间管理，这显著减少了原本的内核软件开销。对于崩溃一致性，MadFS 使用写时复制（CoW）来保证。写时复制受益于块映射的嵌入，因为映射可以在不涉及内核的情况下有效更新。对于跨进程同步，该工作在用户级别引入了无锁乐观并发控制（OCC），它可以容忍进程崩溃并提供更好的可扩展性。实验结果表明，在并发工作负载下，MadFS 的吞吐量高达 ext4-DAX 的 3.6 倍。对于真实世界的应用程序，与 NOVA 相比，MadFS 在 LevelDB 上为 YCSB 提供了高达 48%的加速，在 SQLite 上为 TPC-C 提供了 85%的加速。

此用户空间文件系统通过块映射嵌入、写时复制和乐观并发，绕过了内核软件开销，容许用户进程崩溃，性能良好。但是在安全性方面仍然略显不足。

3.3 持久内存数据库索引

自持久内存的构想提出以来，数据库索引就一直是热门领域。传统索引（在此不讨论学习型索引）包含两种类型，一种是范围索引，其增删改查的时间复杂度在对数级，并且支持范围查询，这包括 B 树、B+树、LSM 树和跳表等；另一种是哈希索引，理论上的任何操作的平均时间复杂度可以到 O(1)但不支持范围查询。持久内存数据索引一般就是针对这两种索引进行的优化，接下来就通过 3 篇文章了解一下进几年的研究现状。

ChameleonDB: a Key-value Store for Optane Persistent Memory 提出了一种结合 LSM 树和哈希表的索引结构，使得其性能相对于其余 LSM 树或者哈希表都非常均衡，有着非常好的写和读性能和并行带宽。

该论文首先提出了针对傲腾内存上构建索引的三点挑战：

（1）傲腾内存是块设备，如果使用较小的写会有较大的写入放大；

（2）傲腾内存是高速设备，LSM 树多层级的结构会影响整体性能。

（3）傲腾持久内存是非易失性的，为了性能将大量索引组织在 DRAM 上是不合理的，具有极大的恢复开销。

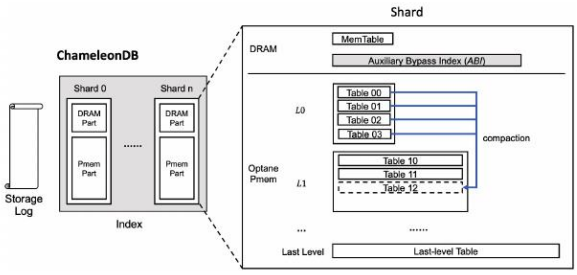

图 4 ChameleonDB 整体架构图

图 4 是 ChameleonDB 整体架构图。其中将整个索引分成多个 shards 形成 multi-shard 结构，这种结构有很大的并发性能。同时，在 DRAM 上使用一个 memtable 来聚合写操作和加速最近查询，在 pmem 上使用多级的哈希表来进行分层存储以减少写入放大。pmem 和 DRAM 上的哈希表都是用线性探测提高负载因子和写入冲突，进一步减少写放大。同时，引入 Direct Compaction 来实现多层级的 hashtable 压缩，减少压缩开锁，提升空间利用率。

这种类 LSM 树的多层级哈希表在读性能上不佳，于是 ChameleonDB 在每个 shard 额外维护了一个 Auxiliary Bypass Index（ABI）来加速读查询。在插入到最后一层之前，数据索引都会记录在 ABI 中，因此查询最多只会查询三次哈希表（memtable、ABI、last level hash），极大地减少了读取开销。

此外 ChameleonDB 又对长尾做出优化：在 ChameleonDB 中引入了一个动态的 Get-Protect Mode 来监控读操作的尾延迟并调整 Compaction 的时机。具体而言，就是一个读操作尾延迟到达一个阈值的时候，ChameleonDB 挂起所有的上层 Compactions，包括 flush 操作，延缓最后一层压缩的执行。

实验结果表明，与传统的基于 LSM 树的 KV 存储设计相比，ChameleonDB 将写入吞吐量提高了 3.3 倍，并将读取延迟降低了约 60%。ChameleonDB 通过使用更少的 DRAM 空间，即使 KV 存储完全

使用 DRAM 索引，也能提供具有竞争力的性能。与 CCEH（一种持久的哈希表设计）相比，ChameleonDB 提供了 6.4 倍的高吞吐量。

与之相似的其实还有一篇的工作，*Plush: a write-optimized persistent log-structured hash-table*，也是一种类似 LSM 树的多层级的哈希表。本文更加注重崩溃一致性方面的实现。

本文的设计思路在于，傲腾持久内存其实是一个块设备，所以持久内存中哈希表的写入放大可以通过应用传统数据库领域的 LSM 树来缓解。于是就设计了多层次的哈希表结构，入下图 5 所示，包括一层 DRAM 哈希表和最多四层的 PMEM 哈希表。上层哈希表的某一个通满时，将自动将向下层对应的位置迁移。这样，对于 PMEM 的小写入将被移除，转而形成对于 DRAM 的小写入，聚合后再写入 PMEM。每一个桶中，Plush 还额外加入了一组布隆过滤器，用于提升读取性能。

由于存在 DRAM 索引，这将导致系统崩溃后的不可恢复，因此 Plush 同时维护一个恢复日志。同时由于傲腾内存 membuffer 的存在，恢复日志的顺序写入不会引起写放大。恢复日志如图 6 所示。

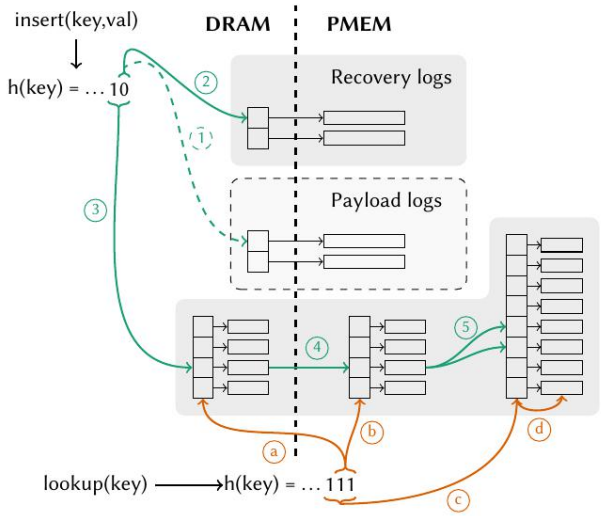

图 5 Plush 架构图

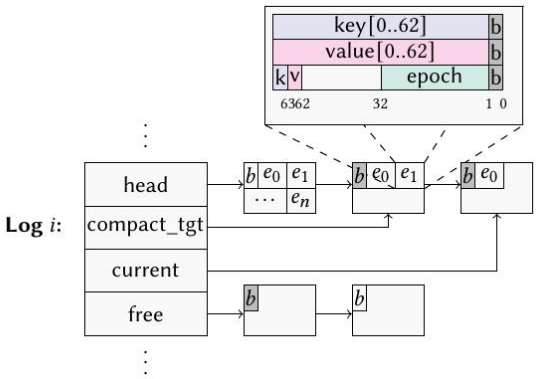

图 6 恢复日志

恢复日志是一个包含许多日志块的持久队列。每个日志块有 n 个日志条目，并包含一个有效位 b。日志条目的结构如图 3.3 最上面的部分。日志条目共 24B。每 8B 的最后一位都是一个有效位 b。当且仅当日志块中的有效位和日志条目的所有有效位完全一致，才表示这个日志条目是有效的；同时，通过简单的翻转日志块的有效位，就可以使整个日志块失效。对于日志条目的写入，其利用 8B 原子写，使得只需要使用三次原子写入和一次缓存行刷新与栅栏指令即可完成持久化的写入。因为即便在持久化前崩溃，任何有效位的不一致都将标志这是一次失败的写入，从而保证了崩溃一致性。如果不使用上述方法，那么一般需要使用两次缓存行刷新和栅栏指令分别持久化数据和有效位以保证崩溃一致性，效率较低。

恢复日志作为持久队列，包含队头指针、合并块指针、当前块指针、空闲块队列头指针，以及块之间的链接信息。新日志总是写入当前块。当当前块写满时，队列总是申请一个块继续写入，并尝试将合并块下一个块合并到合并块中，并将这个块链接到空闲块队列中。

对于具体操作，这里先说明迁移的概念。当某个目录条目的对应的所有桶都满了但仍需要在桶内插入数据时，迁移就会触发。具体来说，首先，根据扇出重哈希目录条目内的所有数据。比如，当扇出为 n，那么我们需要将键的哈希值再取高 $\log_2 n$ 位来将数据分成新的 $\log_2 n$ 组。其次，将分组的数据移动到对应的下一级的目录条目中。这就完成了一次迁移。迁移过后，原条目的数据将置空，因为数据已经被移动到了下一级哈希表中。

Plush 插入过程，首先数据插入恢复日志（1）。计算键的哈希值，并计算对应条目索引值，先向

DRAM 哈希表尝试插入数据（2），此时需要取得对应目录条目的锁。如果对应条目满了，先将此条目的数据迁移到持久内存第 0 层哈希表（3）。如果有必要，递归迁移，比如（4）。迁移完成后（或者根本没有必要迁移），插入数据（2）。

查询过程是对哈希表逐层进行的。查询过程使用乐观读。一旦在某一层查询成功，就不再继续查询。

在配备 768 GB Intel Optane DPCMM 的 24 核服务器上，Plush 的插入性能比最先进的 PMem 优化哈希表高出 2.44 倍，同时仅使用少量 DRAM。

它通过将写入放大降低 80%来实现这一加速。对于查找，其吞吐量与已建立的 PMem 优化的树状索引结构类似。

本小节最后一个需要介绍的工作来自于论文 *Viper: an efficient hybrid PMem-DRAM key-value store*。Viper 仍然针对写进行优化，并且保证读的性能。

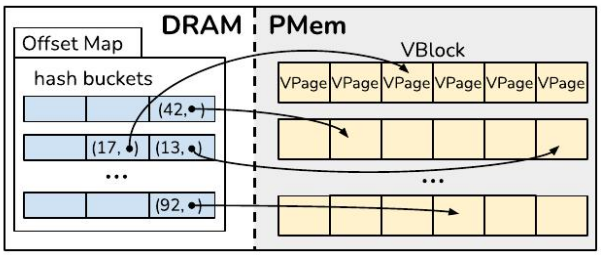

图 7 Viper 架构图

Viper 的设计目标如下：

（1）尽量将 PMEM 的随机写转化为顺序写；

（2）数据做 Page 对齐；

（3）多线程分散到多个 dimm 中，这可以提高并行度。

针对这些可选项，Viper 的设计如下：

将索引放在 DRAM，将数据放置在 PMem。在 PMem 里把数据又分成多个 VBLock，每个 VBlock 里面又包含多个 VPage，写的时候是一个 VPage 一个 VPage 顺序往下写。在写入时先找 VBlock，再找 VPage，保证顺序写以利用傲腾内存特性。同时每个 VBlock 由不同的线程负责，减少写冲突

评估表明，Viper 在核心键值数据库操作方面显着优于现有的键值数据库索引，同时提供完整的数据持久性。 此外，对于写入工作负载，Viper 的性能比现有的仅 PMem、混合和基于磁盘的键值数据库索引高出 4-18 倍，同时与它们的获取性能相当或者较优。

上面三个工作的特点都对于持久内存上数据库索引的写做了一定的优化，全部都采用了 DRAM+PMem 的混合架构。他们主要考虑傲腾内存的粒度问题，顺序写问题。同时考虑了并发控制等问题。虽然傲腾内存已经停产，但是这些思路仍然值得借鉴。

3.4 本章小节

本章主要介绍了持久内存系统和应用的研究现状，包括持久内存编程模型和库、持久内存文件系统以及持久内存数据库索引三个大方向。对于内存编程模型，主要介绍了 Zhuque 这个对应用程序具有良好兼容性的模型；文件系统方面说明了一些用户空间文件系统的设计思想；数据库索引方面讲解三篇关于混合索引的文献，

4 未来研究方向

随着傲腾持久内存退出市场，未来持久内存软件栈的研究应该不会再以傲腾持久内存的特性为优化点进行。由于在傲腾持久内存发布前，已经做了相当多的研究工作，因此针对持久内存基础特性而做老软件优化的潜力应该已经不太多了。

未来对于持久内存的研究应该主要是结合其他新兴的应用场景或者新兴技术，比如 CXL（Compute Express Link）内存、人工智能、大模型计算等等。安全方向应该也是一个非常不错的领域和机会，比如访问控制、加密以及安全删除等。

5 结论

近年来，持久内存作为一种新型存储设备，带有字节级寻址和非易失性，同时具备类似 DRAM 的高性能特点。一直以来，它被视为存储领域的一种新希望，特别是随着英特尔在 2019 年推出傲腾持久内存并投入商业应用，更是引发了学术界和工业界对持久内存研究的极大兴趣。近年来，涌现了大量基于持久内存特性展开的工作，涉及编程模型、库、文件系统以及数据库等多个领域。本文对近期涌现的持久内存系统和应用进行了梳理，并进行综合分析。最后，考虑到傲腾持久内存停产的情况，对未来持久内存系统和应用的研究方向进行了展望。

致 谢 感谢数据中心课程的几位老师的精彩的课堂以及循序渐进的教学，使我对于数据中心相关的知识有了一定的了解；同时，也感谢其他同学们对于论文分享的积极准备与精彩的论文讲解，使得我可以触摸到更多其他领域的前沿知识，让我更好地走向未来的研究生生活。学无止境，一切美好都将从此刻再次启程。加油吧，明天属于我们。

参 考 文 献

[1] Scargall S. Programming persistent memory: a comprehensive guide for developers. Santa Clara: Springer Nature, 2020.

[2] Hodgkins G, Xu Y, Swanson S, et al. Zhuque: failure is not an option, it’s an Exception//Proceedings of the 2023 USENIX Annual Technical Conference (USENIX ATC ‘23). Boston, USA, 2023:833-849

[3] Zhong S, Ye C, Hu G, et al. MadFS: per-file virtualization for userspace persistent memory filesystems//Proceedings of the 21st USENIX Conference on File and Storage Technologies (FAST ‘23). Santa Clara, USA, 2023: 265-280.

[4] Zhang W, Zhao X, Jiang S, et al. ChameleonDB: a key-value store for optane persistent memory//Proceedings of the Sixteenth European Conference on Computer Systems (EuroSys '21).New York, USA, 2021: 194-209.

[5] Vogel L, Van Renen A, Imamura S, et al. Plush: a write-optimized persistent log-structured hash-table. Proceedings of the VLDB Endowment, 2022, 15(11): 2895-2907.

[6] Benson L, Makait H, Rabl T. Viper: An efficient hybrid pmem-dram key-value store. Proceedings of the VLDB Endowment , 2021, 14(9): 1544-1556.

附录X.

课堂汇报问题记录：

在提到各种编程模型中，FASE 是如何实现的？

Atlas 提出了故障原子部分（FASE，Failure Atomic Section）的概念作为事务的替代方案。FASE 是一种故障原子性操作，它在线程获取第一个锁时开始，在没有锁时结束。重要的是，最终持有的锁可能与第一个锁不同。由于此锁定方案允许在某个 FASE 提交之前更新对其他 FASE 可见，因此需要基于 FASE 的库来跟踪线程之间的依赖关系，并在发生故障时回滚依赖的 FASE。由于 FASE 是在运行时动态形成的，因此现有的基于锁的代码不需要用户注释。

基本的实现为，使用运行时检测数据依赖，并通过 redo 和 undo log 的形式记录写入。在发生故障时根据日志回滚或者重做。