# 持久性内存文件系统综述

戴秋爽[1)]

**摘　要**　传统的内存或外存管理方法对新兴的可字节寻址的非易失存储器的管理并不高效，不能充分发挥持久性内存的优势，如何既兼容现有的应用，又管理好持久性内存，减小开销，广大学者在不同层面不同角度对持久性内存文件系统进行了创新和发展。本文根据持久性内存文件系统现有的发展分为了六类，对已有研究进行梳理和汇总，并提出了几点展望。
关键词　持久性内存；文件系统；日志结构；老化；虚拟内存；拆分架构；分布式存储

## Persistent Memory File System Overview

Dai Qiushuang[1)]

**Abstract**　Traditional memory or external memory management methods are not efficient for managing the emerging byte-addressable non-volatile memory and cannot fully exploit the advantages of persistent memory. How to be compatible with existing applications while managing persistent memory and reducing overhead, scholars have innovated and developed persistent memory file systems at different levels and from different perspectives. In this paper, we have divided the existing developments of persistent memory file systems into six categories, sorted out and summarized the existing research, and presented several perspectives.
**Key words**　Persistent memory; file systems; log structures; aging; virtual memory; split architecture; distributed storage

## 1　引言

随着科技的发展，移动互联网的普及，人类社会每天产生的数据量剧增，人们对计算机中存储资源的要求也在逐步提高，一个快速、可靠且容量庞大的存储系统的需求显得越来越紧迫。一方面，存储设备需要有可观的存储容量来应对大数据下处理、存储大量的数据的要求；另一方面，存储设备需要较高的读写性能对数据进频繁的行读取和修改。在传统的存储介质中，磁盘容量大，但是读写速度慢，内存读写速度快，但是容量小，均不能同时满足这两个需求。而持久性内存的出现，缓解了内存与外存间的巨大差异，也为人们解决当前面临的问题给出了新的思路。

持久性内存，也被称为非易失性内存，是一种高速存储介质。其具有以下两个主要优点：首先持久性内存可以像内存一样使用内存总线，按字节随机寻址，而且读写速度接近内存；二是持久性内存所保留的数据是非易失性的，不会像内存一样在断电之后丢失所有的数据。

由于持久性内存的这两种优势，使其成为了介于磁盘和内存之间的一种高效的存储介质。

持久性内存可字节寻址的特点，使得可以将其直接挂载在内存总线上通过 load、store 命令访问，从而颠覆了传统计算机存储体系的内、外存架构。如图 1 所示，左边是传统的计算机内、外存架构，内存 DRAM 是可字节寻址的，CPU 通过 load、store 命令对其进行访问，作为外存的硬盘（Hard Disk Drive，HDD）或者固态硬盘（Solid State Drive，SSD）是以块为单位进行寻址的，CPU 通过 I/O 命令对它们进行访问。右边是使用了持久性内存的计算机系统存储架构，动态内存 DRAM 和持久性内存 PM 都是可字节寻址的，CPU 通过 load、store 命令对它们进行访问，通过地址的不同进行区分。

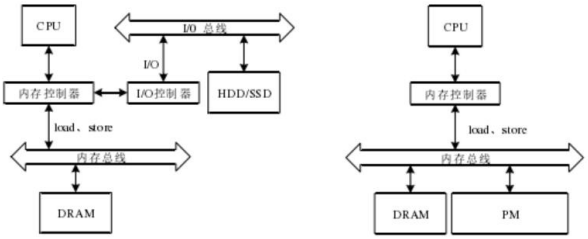


图 1 计算机存储体系架构

从计算机系统软件的角度上看，有两种管理持久性内存的方式。

第一种，持久性内存作为工作内存使用，通过内存管理单元管理，最大程度发挥持久性内存的访问性能。第二种，持久性内存作为存储设备使用，使用文件系统对持久性内存进行管理可以为应用软件提供通用的 I/O 接口对持久性内存进行访问，同时，文件系统的保护机制可以为存储在持久性内存上的数据提供一致性保护。

由于目前现有的文件系统是针对 HDD 和 SSD 这类介质进行设计的，并不能很好地利用持久性内存。

在传统的操作系统设计中，外存主要是一类慢速设备（磁盘等），由于磁盘与 CPU 寄存器，缓存等设备之间存在的速度差异，需要使用一些优化策略，比如文件系统缓存、块设备请求重排序等，减少对慢速设备的访问次数。在非易失性内存中，缓存和重排在整体访问流程中占的比例增大，而在访问速度快的非易失性内存中性能提升并不明显。对于持久性内存而言，由于可以按字节随机寻址，读写速率贴近内存的优点，可以摒弃原操作系统中对于慢速设备的优化设计。

传统的文件系统都是以块为单位访问 SSD 或 HDD 的，不能很好的利用非易失性内存的可字节寻址特性，无法充分发挥非易失性内存的性能优势。

非易失性内存更高的写入次数也带来了相应的磨损的问题，需要通过软硬件实现磨损均衡。传统的基于 SSD 的文件系统，所采用的磨损均衡方案，并不适合于非易失内存。传统文件系统的写入以块为单位，导致统计磨损次数精度不高，同时会出现写放大的问题。同时针对 SSD 特性进行优化的以 block 为单位擦除，以 page 为单位写入对非易失性内存来说也是画蛇添足的操作。

对于传统的磁盘文件系统，在操作系统发出一个文件相关的系统调用后，需要经过漫长的软件层来到达底部的文件系统。如图 2 所示。

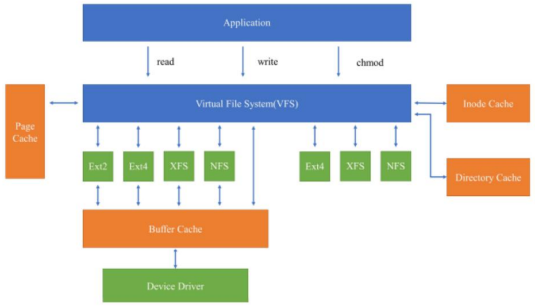


图 2 各软件层在 Linux 内核中的位置

应用程序使用文件 API（read/write 等）后，需要经过虚拟文件系统 VFS 层，文件系统层，块设备层，磁盘驱动层，最终才到达物理磁盘。整个过程的 IO 路径过长，并且由于页缓存(page cache)的设计，数据需要经过多次拷贝才能到达用户。这些问题都导致持久性内存的优点无法得到充分发挥。

综上所述，为了充分发挥持久性内存文件系统的优势，需要对现有的文件系统做出一些修改和调整，设计一种新的基于持久性内存的文件系统。

## 2 原理和优势

传统存储架构、软件及各层次都是针对传统器件设计的，难以发挥新型存储器件的特性;同时新型存储器件本身也有写性能和器件寿命等不足，这些都是需要解决的问题。

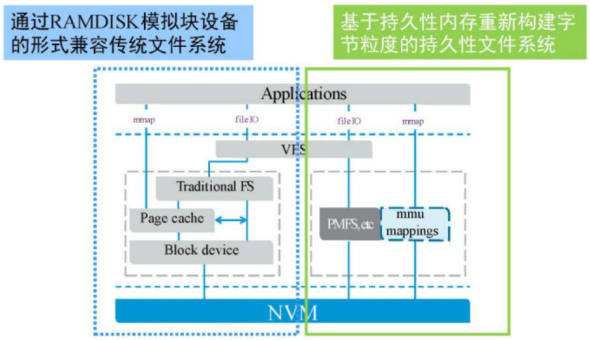


图 3 持久性内存文件系统

持久性内存文件系统有两类，如图 3 所示，一类是将传统的文件系统梳理成一个块，默认成一个块的设备，不需要修改传统文件系统就可以在持久化内存中使用。在没有针对持久化内存设计的文件系统是，RAMDISK 的形式使得传统的文件系统快速受益于内存级的数据持久化，相比于外存性能有数量级的提升。但软件层的开销巨大，无法充分利用持久性存储介质的优势。为了更加充分的利用持

久性内存存储的优势，对传统的文件系统进行改造。为了实现一个更加符合持久性内存存储特点的文件系统，综合考虑持久性内存的存储特点。

持久性内存存储具有低功耗、非易失、高可靠性、抗震动、以及更大容量高性能片上存储和更快的外存读写速度。并且其可字节寻址的特点，使得可以将其直接挂载在内存总线上。完全针对持久性内存存储的特点，设计一个字节粒度的文件系统。主要包括细粒度的数据访问形式和内外存的融合管理。此外，持久化内存挂载在内存总线上，有一些可以直写的内容可以避免双层拷贝，减少块层开销。

1）在文件系统中设计细粒度的数据访问形式，避免以块为单位的 SSD 或 HDD 访问，充分利用非易失性内存的可字节寻址特性，使用原子原地更新，细粒度日志以及写时更新策略的组合优化了数据一致性，发挥非易失性内存的性能优势。

2）针对内外存的融合管理，传统的文件系统中，PM 连接在外存总线上，对内存的 stray writes 并不会影响外存中数据的持久性。由于软件可以像访问普通内存一样访问 PM，这时一些 stray writes (一般由程序 bug 导致)很可能会修改 PM 中的值并使其持久化，PMFS 使用 writes window 来解决这个问题。

3）在避免双层拷贝造成的开销部分，在传统的文件系统中，当对磁盘上的一个文件发起读操作，首先要使用 memory-mapped I/O 需要将 storage 中的页先拷贝到 DRAM 中。当使用内存映射文件的方式对文件发起访问，则不需要数据拷贝可以直接访问。为了减少数据拷贝开销，以 PMFS[1] 为代表的持久性内存文件系统优化了 mmap IO，使得 PM 内存页直接映射到用户态应用的地址空间。PMFS 还是用了透明大页支持来进一步优化 mmap。

设计持久化内存存储文件系统针对持久化内存的特点，充分利用持久化内存访问速度快容量大和字节寻址的特点，并且解决了持久化内存直接挂载在内存总线上可能导致的安全性问题、频繁的访问带来的一致性和磨损问题。充分发挥持久化内存的优势，提高了文件读写访问性能，为文件系统的形式和应用场景提供了新思路。

# 3 研究进展

## 3.1 日志结构持久性文件系统

日志结构文件系统（LFS）原始地设计用于利用磁盘顺序访问的高性能。LFS 首先缓存随机大小写到内存，再将它们转换成大的顺序写到磁盘中，大大提高了磁盘的写性能。实现日志结构文件系统复杂，原因是它需要顺序地写磁盘的连续空闲区域。日志清理增加了 LFS 的开销并且降低了性能。

在设计针对非易失性内存文件系统时，发现了以下现象：1)支持原子更新的日志很容易在 NVMM 中实现，但支持快速搜索的树结构（eg.B+树的插入中包含随机和顺序写入的混合，开销比较大）在 NVMM 中实现相对困难。2）NVMM 具有快速随机访问的特性，不需要复杂的日志清理以提供连续的空间 3）使用单个日志限制了并发访问

以 NOVA[2]为代表的日志结构持久化文件系统扩展了 LFS，并针对 NVMM 日志友好的特点进行改进，利用 NVMM 中日志易实现、不需要复杂的日志清理的优势，并设计解决日志结构在持久化内存存储中难以并发访问的问题。NOVA 针对以上问题提出 5 点设计：

1)在 NVMM 中保存日志，在 DRAM 中保存索引并在 DRAM 中构建 radix 树结构保存索引以便加速查找操作

2)每个 indode 保存自己的日志，允许跨文件并发更新并且无需同步化操作。该结构使得文件访问高并发，并且在故障恢复期间，可以同时重新执行多个日志。

3)对于复杂的原子操作（设计多个 inode）使用 logging 和轻量级 journal。为了原子地写数据到一个 log，NOVA 首先将数据追加到 log，然后原子地更新该 log tail 指针以便提交该更新，这避免了 journaling 文件系统的双倍写开销以及 shadow paging 系统的迭代更新问题。

4)以单链表的方式实现 log。不需要分配大量的连续区域用于 log；可以执行细粒度，页大小粒度的日志清理；回收仅包含过时条目的日志页只需要几个指针分配。

5)在索引节点的日志中仅记录元数据，使用写时复制（COW）的方式更新数据页。日志变得更短，加快恢复过程；不需要拷贝文件数据，使得垃圾回收更简单和高效；从 DRAM 空闲列表中添加和移

除页即可完成回收过时页以及分配新的数据页操作；在高写负载和高 NVMM 下也能够立刻回收过时的数据页。

NOVA 在传统日志结构文件系统的实现方式基础上针对 NUMM 设计了一种新的日志结构，并且在此之上设计了高效的垃圾回收策略。在提供同样强大数据一致性保证的文件系统中，NOVA 有着非常高的 IO 吞吐量。

### 3.2 针对文件系统老化问题的文件系统

当文件系统是新创建的空文件系统时，现代持久内存(PM)文件系统在基准设置中表现良好。长期来看，随着文件的长期创建、修改和删除，持久内存文件系统会经历显著的性能下降，当运行数据密集型应用程序时，

尤其是当运行具有数百万个小文件(KB)的邮件服务器应用程序时，这个问题会更加严重。这种现象被称为文件系统老化。我们发现性能下降归因于文件空间和数据空间的存储管理效率低下。并且随着文件系统的老化，持久内存磨损得更快。针对文件系统的老化问题，有以 SanGou 和 WineFS 为代表解决方法。老化中具有重大影响的磨损问题，也提出了 coutour 方法。

SanGou[3]提出了一种基于持久内存的存储系统上的老化感知自由空间管理机制。由两种技术组成，用于粗粒度自由空间管理的分散内存分配技术和用于细粒度管理的无聚集技术。

对于粗粒度的空闲空间管理，使用分散内存分配以块粒度(例如 4KB)管理持久内存的空闲空间。为了使持久内存分配和回收更快，散点分配将整个持久内存分成多个分配组，以避免分配争用，并允许在故障恢复中进行并行扫描。此外，通过执行随机游走算法，确保所有空闲块都有平等的分配机会，以避免局部写入。

对于细粒度的自由空间管理，使用无聚集技术来管理字节粒度的目录文件的自由空间。在无聚集技术中使用哈希表映射空闲列表来有效地管理所有空闲空间，加快目录文件中的空闲空间分配和回收。无聚集在 DRAM 中为每个目录文件保留一个 AVL 树，以索引 dentry 名称字符串的哈希值，有效的加快了 dentry 的查找和删除操作。

SanGou 数据结构布局如图 4 所示。

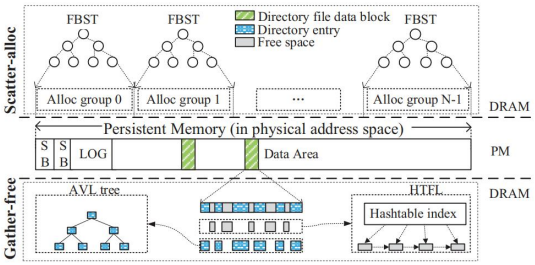


图 4 SanGou 数据结构布局

WineFS[4]通过设计 hugepage 感知的 PM 文件系统，将一个新的对齐感知分配器与避免碎片的一致性和并发性方法相结合，以保持使用 hugepage 的能力。WineFS 的体系架构如图 5 所示，将文件系统划分给每个逻辑 CPU，易于并发。每个逻辑 CPU 都有自己的日志、inode 表和针对对齐的区段和孔的自由列表。WineFS 为元数据使用 DRAM 索引来进行有效的目录和资源查找。VFS 层中的共享锁帮助 WineFS 协调多个日志。

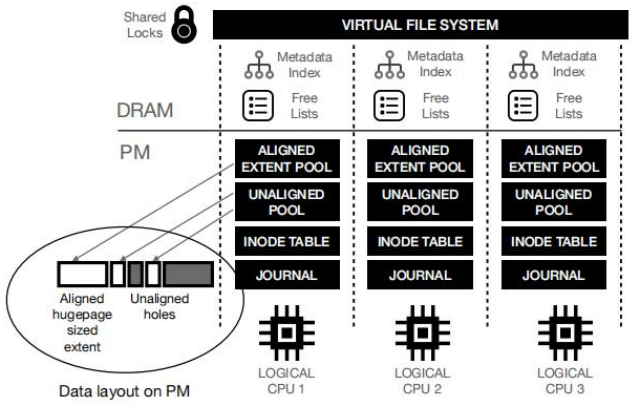


图 5 WineFS 的体系架构

WineFS 通过端到端的设计避免中断大量的内存使用。考虑到文件系统的许多相互关联的方面，从分配策略到崩溃一致性方案，再到一致性，都会影响其随着时间的推移继续使用大量文件的能力。WineFS 使用了一种新颖的对齐感知分配器，通过保留 2MB 对齐的 2MB 扩展区，使用大的分区进行映射。大的分配请求使用对齐的扩展来满足，而小的分配使用“孔”来满足。同时为了维护形成的布局，采用了受控的碎片方法，为元数据提供专门的存储空间，元数据和日志就地更新，使用日志来保持崩溃一致性，而不是在以系统调用为中心的 PM 文件系统中流行的日志结构，以避免数据重新定位，破坏精心规划的布局。采用混合技术对文件进行原子化更新，当数据布局重要时使用日志更新，不重要时写时复制，综合考虑布局和效率问题。有效的在

不牺牲性能的前提下避免老化，维持持久化内存的性能。

Coutour[5]重点解决在进程变化的 PM 设备上文件系统的索引节点部分的磨损均衡问题。通过将更新不均匀的索引节点与具有不同耐久性的内存页面相匹配来平衡存储索引节点的物理页面的磨损。

coutour 由两种技术组成。首先，通过信息节点虚拟化实现了信息节点的迁移。信息节点虚拟化通过逻辑信息节点和物理预防性维护上的信息节点插槽之间的“虚拟信息节点”层，将信息节点与“固定”物理位置分开。逻辑索引节点和对应的虚拟索引节点之间的映射关系被记录为偏转表中的偏移。给定逻辑索引节点的索引节点号(i-number)，文件系统可以使用相应虚拟索引节点的偏移量和虚拟地址来访问实际的索引节点插槽。因此，信息节点虚拟化使逻辑信息节点 1 能够通过更改其偏移量移动到不同的物理位置。

其次，通过一个 inode 迁移机制。一个永磁设备被分成多个内存域，显示出不同的耐久性。然而，同一域中的存储单元具有相同的耐久性。考虑到持久内存域的特性，使用跨域迁移和域内迁移来迁移信息节点。跨域迁移算法通过“写入预算”控制内存域上的写入平衡。根据索引节点的写入频率和内存域的相对写入预算，将索引节点迁移到适当的内存域。在域内迁移中，通过将信息节点迁移到磨损较小的插槽，将信息节点的写入均匀地分布到同一个域中的信息节点插槽。

通过对信息节点的迁移和 inode 节点的迁移，有效地避免了局部过热的问题，显著改善索引节点的磨损均衡。

### 3.3 融合虚拟内存管理技术的持久性内存文件系统

文件虚拟地址空间

在文件访问中，文件系统搜索元数据结构以找到文件数据的物理位置。现有的临时和持久内存文件系统都使用软件来搜索元数据。或者构建额外的映射表，将文件映射到虚拟地址空间。这些文件访问方法不能充分利用内存映射硬件的优势。

Edwin H[6]提出了一个新的框架，称为“文件虚拟地址空间”的高性能持久内存文件系统。在此框架内，每个打开的文件都有自己的连续虚拟地址空间，该空间由文件专用的分层页表组织。文件系统可以通过文件的连续虚拟地址空间，使用硬件内存管理单元来定位文件数据的物理位置。基于该框架设计并实现了一个文件系统——可持续内存文件系统 SIMFS。在 SIMFS 中，文件的数据是由一种叫做“文件页表”的结构来组织的。所提出的帧结构和 SIMFS 可用于连接到存储器总线的任何字节可寻址存储器。

打开的文件的文件虚拟地址空间嵌入到调用进程的地址空间中。这个嵌入过程可以通过更新很少的指针来有效地完成。一旦嵌入过程完成，数据页就可以通过直接使用虚拟地址的加载/存储指令来访问，并且文件数据可以在没有中断(如页面错误)的情况下被读取。SIMFS 能够利用 CPU 中的内存映射硬件，以显著的高吞吐量访问文件数据。

在一致性和应用程序接口部分进行了进一步优化。提出了一种称为伪文件写入(PFW)的技术，以有效地实现与写入时复制(COW)相同的一致性。如果文件系统通过 COW 将新数据写入新的备份空间，则文件系统调用的数据传输函数的数量等于新数据的页数。在所提出的框架中，PFW 只调用一次数据传递函数来写入整个新数据。为应用程序提供了新的接口。使用文件内执行方法，应用程序直接管理文件系统中的文件，而无需将数据复制到任何缓冲区。此外，这种方法可以减少所需的系统调用次数为应用程序带来比传统方法更高的效率。

虚拟超级页面机制

持久内存文件系统可以有效地定位具有较短文件索引的数据页，并减少大文件的空间管理开销。此外，使用超级页面的文件系统也可以从比 4KB 页面更高的 TLB 命中率中受益。但当文件系统使用超级页时，特别是当重写数据的大小远远小于超级页的大小时，CoW 可能会导致大规模的数据迁移。此外，使用超级页的文件系统可能会有巨大的内存空间浪费，因为每个文件的最后一个超级页中的空闲空间不能被其他文件使用。

为了解决这一问题，使用虚拟超级页面机制(VSM)[7]对超级页组织文件数据的文件系统进行改进，通过内存空间的现有虚拟到物理地址映射来利用文件系统的超级页优势。通过多粒度写时拷贝(MCoW)和零拷贝文件数据迁移(ZFDM)机制，可以很好的减少写放大，同时提高空间利用效率。

### 3.4 IO 路径优化的高性能文件系统

为了避免用户缓冲区、操作系统页面缓存和存储层之间的双重复制开销，最先进的 NVMM 感知文件系统绕过了直接在用户缓冲区和 NVMM 存储之间复制数据的操作系统页面缓存。然而，现有

NVMM 技术的一个主要缺点是写入速度慢。因此，对所有文件操作的这种直接访问可能会导致次优的系统性能。

为了解决这些问题，Jiaxin Ou 等人[8]设计了 HiNFS，一种用于非易失性主存储器的高性能文件系统。HiNFS 的体系结构如图 6 所示

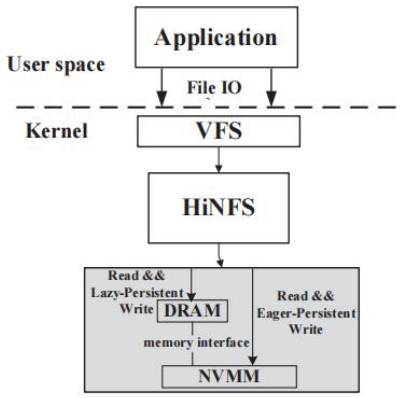


图 6 HiNFS 体系结构图

HiNFS 将延迟持久文件写入(即文件系统允许延迟持久的写入操作)暂时缓冲在 DRAM 中，以隐藏 NVM 的长写入延迟。为了提高缓冲区块的读/写性能，HiNFS 利用 NVMM 的字节可寻址特性，以细粒度的粒度管理动态随机存取存储器缓冲区。此外，HiNFS 使用内存接口在动态随机存取存储器缓冲区和 NVMM 存储之间进行交互，而不是通过通用块层，以避免高软件堆栈开销。为了消除关键路径的双拷贝开销，HiNFS 为需要立即持久化的写入操作提供了对 NVMM 的直接访问，并利用它们具有相似的读取性能直接从 DRAM 和 NVMM 读取文件数据。然而，向动态随机存取存储器和 NVMM 写入数据对确保读取一致性提出了挑战。同时，它还要求文件系统在发出写操作之前识别急切持久的写操作。为了确保读取一致性，高速缓存使用动态随机存取存储器块索引和高速缓存线位图的组合来跟踪动态随机存取存储器和 NVMM 之间的最新数据。最后，在发出写操作之前，HiNFS 使用缓冲区优势模型来识别急切持久的文件写操作。

### 3.5 使用拆分架构的持久性内存文件系统

传统文件系统会给每个文件系统操作增加大量开销，尤其是在写入路径上。开销来自于在关键路径上执行昂贵的操作，包括分配、记录和更新多个复杂结构。需要设计更合理的体系结构来降低开销。

SplitFS 在已有的拆分架构的持久性内存文件系统基础上进行进一步改进。

Aerie[9]是最早倡导从用户空间访问 PM 的系统之一。Aerie 提出了一种类似于 SplitFS 的拆分架构，具有用户空间库文件系统和内核组件。Aerie 使用用户空间元数据服务器来分发租约，并且只将内核组件用于诸如分配之类的粗粒度活动。相比之下，SplitFS 不使用租约(而是使大多数操作立即可见)，并使用 ext4 DAX 作为其 kernel 组件，将所有元数据操作传递给内核。Aerie 建议取消 POSIX 接口，旨在为应用程序提供接口灵活性。相比之下，SplitFS 旨在高效地支持 POSIX 接口。

SplitFS[10]引入了重新链接原语来优化文件追加和原子数据操作。重新链接以逻辑和原子方式将一个连续的范围从一个文件移动到另一个文件，而无需任何物理数据移动。Relink 建立在 ext4 DAX 中的 swap_extents ioctl 之上，并使用 ext4 日志来确保源文件和目标文件被自动修改。严格模式下的文件追加和数据覆盖都被重定向到临时预防性维护文件，称之为暂存文件。在 fsync()上，暂存文件中的数据被重新链接到原始文件中。Relink 提供原子数据操作，没有分页错误或数据复制。

SplitFS 还引入了优化的日志协议。在严格模式下，SplitFS 中的所有数据和元数据操作都是原子的和同步的。在通常情况下，SplitFS 将每个操作写入单个缓存行的数据(64B)，后跟一个内存整理,相比之下，NOVA 至少写入两条缓存线，并发出两次文件整理。由于这些优化，SplitFS 日志记录在关键路径上比 NOVA 快 4 倍。由于重新链接和优化的日志记录，SplitFS 中的原子数据操作比 NOVA-strict 快 2–6 倍，以较低的软件开销提供了强大的保证。

### 3.6 分布式存储文件系统

NVM 的出现重新定义了计算机体系结构中易失性与非易失性的界限，以往由系统软件保障的数据一致性转变为由 CPU 硬件保障．另一方面，远程直接内存访问（RDMA）因其高速、超低延时、零拷贝和内核旁路的网络传输特性，逐渐被应用到数据中心．RDMA 可以实现应用程序间点对点的通信，同时避免了上下文切换、冗余内存拷贝和内核网络协议栈等开销．持久性内存和 RDMA 的出现为构建新型存储系统提供了新的机遇．

Octopus[11]是第一篇尝试结合 RDMA 与 NVM 访问方式可以构建高效的分布式存储系统的文章，通过考虑 NVM 和 RDMA 特性来重新审视分布式文件系统的数据和元数据机制设计，提出了一个高效

的分布式持久存储文件系统 Octopus。

Octopus 是为一组配备非易失性存储器和支持 RDMA 的网络的服务器构建的。Octopus 由两部分组成:客户端和数据服务器。Octopus 没有集中的元数据服务器，元数据服务分布在不同的数据服务器上。在 Octopus 中，文件以基于哈希的方式分发到数据服务器。文件的元数据和数据块位于同一个数据服务器中。Octopus 的设计框架如图 7 所示。

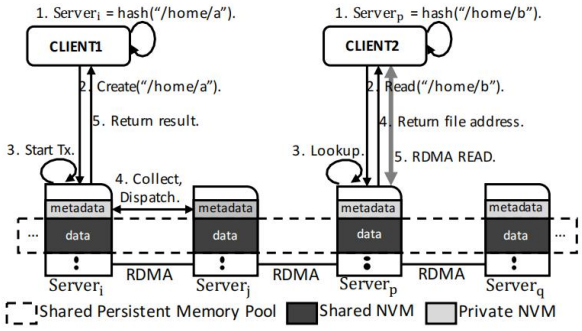


图 7　Octopus 设计框架图

为了有效地探索原始硬件性能的好处，Octopus 将 RDMA 与文件系统机制设计紧密结合起来。数据和元数据机制都进行重新考虑：通过共享持久内存池减少内存拷贝来实现高 I/O 带宽并使用客户端活动 I/O 提高小 I/O 的吞吐量。低延迟元数据访问，以提供具有自识别 RPC 的低延迟和可扩展的元数据 RPC，并使用收集-调度事务减少一致性开销。

在每台服务器中，数据区被导出并在整个集群中共享，用于远程直接数据访问，而元数据区出于一致性原因保持私有。

对于共享数据访问，Octopus 使用 RDMA 原语直接远程读取或写入数据对象。通过使用 RDMA，Octopus 通过引入共享持久内存池消除了文件系统映像和内存缓冲区之间的重复内存副本。该共享池由整个集群中每个数据服务器的导出数据区组成。在当前实现中，使用静态 XML 配置文件初始化内存池，该文件存储池大小和集群信息。Octopus 还通过牺牲网络往返来重新设计读/写流，以使用客户端活动的输入/输出来分摊服务器负载。

对于元数据机制，Octopus 利用 RDMA 写原语为元数据操作设计了一个低延迟和可扩展的 RPC。它还重新设计了分布式事务，通过从远程服务器收集数据进行本地日志记录，然后将它们发送到远程端，来降低一致性开销。

Octopus 实现了与大型 I/O 几乎相同的原始带宽，并且性能比早期的分布式文件系统高出一个数量级。

Octopus 在数据一致性方面的工作有所欠缺，持久性内存和 RDMA 具有的新特性对存储系统的一致性设计提出了 2 项挑战：1）现代处理器利用多核和内部缓存来提高整体运行速度，通常采用乱序执行技术，导致最终写入内存的数据并非是预期的结果，针对持久性内存会出现数据一致性的问题. 此时，为了将数据顺序地写入持久性内存，CPU 需要主动执行数据刷写指令，然而此过程开销高昂，降低 CPU 的处理能力，影响持久性内存和 RDMA 的性能优势；2）RDMA 的通讯过程需要注册内存，且提供了单向原语可以绕过服务器端 CPU 直接读写已注册的内存. 为了性能考虑，RDMA 会将数据先写入至末级高速缓存，由于 CPU 不能及时将数据刷写至持久性内存，因此，一旦出现系统崩溃，原始数据将无法有效恢复.

陈等人就此提出了一种分布式持久性内存文件系统的一致性机制 CCM[12]，该机制主要包括：首先提出并实现了一种基于操作日志的一致性保障策略，将元数据地址和数据的地址与大小写入操作日志，并原子地修改操作日志的尾指针且持久化，这样系统崩溃后，可通过持久化的操作日志进行恢复，这样可保证元数据和数据的一致性。然后，设计了一种客户端对服务器端远程写一致性策略，该策略中客户端使用 RDMA write-with-imm 原语远程写入服务器端内存，此原语附带客户端标识符，服务器端可以感知其携带的客户端标识符，进而服务器端 CPU 可以定位需要刷写的地址，对客户端写入的数据执行持久化操作，保证数据的持久化。最后，实现了一种服务器端的数据异步持久化，通过对元数据、数据和操作日志执行异步刷写，提高了系统的处理能力，降低了持久化操作的开销。

## 4　总结及展望

本文对持久化内存与传统内外存进行比较，分析了传统文件系统的特点，提出了在持久化内存中文件存储系统需要进行的改进和优化。针对持久化内存中可字节寻址，访问速度快的特点在文件系统中进行改进，同时由于在提升性能时造成的读写频繁，磨损不均衡，持久化内存直接连接总线可能受到恶意篡改，需要设计相应的策略进行保护以及处理老化问题。

将现有的研究根据持久化内存文件系统所解

决的问题和解决方法进行分类，将已有的研究分为六类。基于日志结构持久性文件系统、针对老化问题的文件系统改进、融合虚拟内存的文件系统、IO路径优化的高性能文件系统、使用拆分架构的文件系统、分布式文件系统。在每种分类中选出了较有代表性的论文进行介绍，对现有的持久性内存文件系统发展有了系统性的总结和归纳。

最后根据已有的研究现状，分析不同角度不同方法中存在的不足并参考多篇相关分析，对持久化存储文件系统相关方向提出了 3 点建议。

1.在存储结构的创新与优化方面如何优化或变革现有存储层次，包括多级持久化存储的设计。

2.持久性存储硬件性能相比传统磁盘存储的性能提升极大。存储系统中相应的软件开销显得尤为突出。所以针对软件的系统优化，采用软硬件结合设计以及细粒度精细化软件设计，将是未来存储系统的研究方向之一。

3.新型高速存储硬件和高速网络硬件动摇了传统分布式系统中存储与通信的条件假设，且这些硬件均提供了新的访问特性和访问模式。新型分布式存储系统的构建需要重新思考分布式存储协议的设计

## 参 考 文 献


[1] Dulloor S R, Kumar S, Keshavamurthy A, et al. System software for persistent memory[C]//Proceedings of the Ninth European Conference on Computer Systems. 2014: 1-15.

[2] Xu J, Swanson S. {NOVA}: A log-structured file system for hybrid volatile/non-volatile main memories[C]//14th {USENIX} Conference on File and Storage Technologies ({FAST} 16). 2016: 323-338.

[3] Zeng K, Lu Y, Wan H, et al. Efficient storage management for aged file systems on persistent memory[C]//Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017. IEEE, 2017: 1769-1774.

[4] Kadekodi R, Kadekodi S, Ponnapalli S, et al. WineFS: a hugepage-aware file system for persistent memory that ages gracefully[C]//Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles. 2021: 804-818.

[5] Chen X, Sha E H M, Wang X, et al. Contour: A process variation aware wear-leveling mechanism for inodes of persistent memory file systems[J]. IEEE Transactions on Computers, 2020, 70(7): 1034-1045.

[6] Sha E H M, Chen X, Zhuge Q, et al. A new design of in-memory file system based on file virtual address framework[J]. IEEE Transactions on Computers, 2016, 65(10): 2959-2972.

[7] Yang C, Liu D, Zhang R, et al. Optimizing performance of persistent memory file systems using virtual superpages[C]//2020 Design, Automation & Test in Europe Conference & Exhibition (DATE). IEEE, 2020: 714-719.

[8] Ou J, Shu J, Lu Y. A high performance file system for non-volatile main memory[C]//Proceedings of the Eleventh European Conference on Computer Systems. 2016: 1-16. *书籍格式*

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

[10] 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. 2019: 494-508.

[11] Yang C, Liu D, Zhang R, et al. Optimizing performance of persistent memory file systems using virtual superpages[C]//2020 Design, Automation & Test in Europe Conference & Exhibition (DATE). IEEE, 2020: 714-719.

[12] 陈波,陆游游,蔡涛,陈游旻,屠要峰,舒继武.一种分布式持久性内存文件系统的一致性机制[J].计算机研究与发展,2020,57(03):660-667.