基于持久内存的文件系统综述

张宇健[1)]

1) (华中科技大学计算机科学与技术学院, 武汉 430074)

摘要 随着科技的发展，新兴的非易失性存储器（NVM）逐渐成熟到接近大规模生产阶段的水平，并被广泛采用。持久性内存同时具备类似 DRAM 的性能和类似磁盘的容量和持久性等特点。传统的外存和内存间的性能差距使得数据的输入输出（Input/Output，I/O）成为计算机系统数据处理过程中的性能瓶颈。新型可字节寻址的非易失性存储器兼具了传统内存和外存的特性，既可字节寻址，又具有数据非易失性，它的出现模糊了内存与外存的区别，可以将其合理应用在存储系统之中，作为持久性内存使用，改变目前限制计算机数据处理性能的内、外存架构。但是，由于新型可字节寻址的非易失存储器本身的一些特点与 DRAM 和传统外存存在着差异，传统文件系统已经不完全适用于对持久性内存的管理。首先，目前主流的文件系统大多是基于磁盘的实现，在操作系统底层（块设备层）和页缓存（page cache）中的软件设计不能够完全贴合持久性内存的优点，而且其中存在的优化算法甚至会起到相反的效果。其次，由于磁盘和持久性内存截然不同的硬件差异和读写方式，使其在维护数据一致性上所采用的方法也需要做出相应改变。另外，由于持久性内存直接接入内存总线，可像传统的内存 DRAM 一样按字节寻址。这将导致持久内存所在的虚拟地址空间直接暴露给操作系统内核的其他进程，存在安全问题。为此需要研究适用于持久性内存的新型体系结构及关键技术，研究新型数据组织模式及管理方法。

关键词 持久内存；文件系统；虚拟内存

1 引言

现如今，随着科学技术的高速发展，人们对计算机中存储资源的要求也在逐步提高。一方面，由于大数据时代的到来，人们需要处理、存储大量的数据，这就要求存储设备需要有可观的存储容量；另一方面，由于需要频繁的对数据进行读取和修改，同时要求存储设备有着较高的读写性能。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。近年来，各种新型非易失存储（Non-Volatile Memory，NVM）器件相继出现，如相变存储器（Phase Change Memory，PCM）[1]，阻变存储器（Resistive Random Access Memory，ReRAM）[2]，自旋矩存储器（Spin Transfer Torque Random Access Memory ，STT-RAM）[3]，给存储系统的设计带来前所未有的机遇，打破了传统存储设备所面临的僵局，为当今人们解决问题提供了新的思路。这些新型非易失存储设备可字节寻址，且同时具有与现有内存相当的性能和外存的非易失性，因此不仅可用作为工作内存（Working Memory）以缓解现有内存容量扩展能力不足的问题，也可用做持久性外存（Persistent Storage）。从产品角度而言，目前已量产的 Intel 傲腾持久内存是第一款商用的 NVM。它在提供了接近 DRAM 的访问延迟的同时，保证了数据是掉电不丢失的。与固态硬盘类似，傲腾内存具有显著的读写不对称的特性：读取带宽大约是写入带宽的 2 倍；延迟方面，对比 DRAM，两者在写入延迟并无明显差异，顺序读取延迟是 DRAM 的 2 倍，而随机读取延迟是 DRAM 的 3 倍[4]。此外，傲腾内存提供了 Memory 与 App Direct 两种模式[5]。在 Memory 模式下，DRAM 被视作 NVM 的缓存，上层软件将仅仅视其为一个更大的内存，在这种模式下，数据并不是非易失的。而 App Direct 模式将傲腾内存视为不同的 NUMA 节点，拥有更大的内存容量与更高的延迟，同时允许应用绕开文件系统，直接使用 mmap 的方式进行 DAX 访问，从而提供接近内存访问的效率。

虽然持久性内存具有诸多优点，但是目前现有的文件系统并不能很好地利用持久性内存。首先，在传统的操作系统设计中，外存主要是一类慢速设备（如磁盘等），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来保证对数据的缓存，减少对慢速设备的访问次数，如由于磁盘顺序化的工作方式，一般的，传统的日志文件系统会将日志顺序存放，以此来增大磁盘的吞吐率，使得对日志的处理更为迅速。同时为了贴合磁盘的工作模式，现有的操作系统使用页缓，IO 排序等操作来对磁盘进行优化。但对于持久性内存而言，由于具备按字节随机寻址，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。其次，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，如在 Linux 下，应用程序调用文件 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。

本文重点讨论 NVM 用做持久性外存（即外存）时面临的诸多挑战、研究现状及未来的方向。众所周知，现有的操作系统都是针对二级存储模型设计的，即工作内存和持久性外存，通过虚拟内存和文件系统两个子模块分别对它们进行管理。与传统的硬盘相比，NVM 有两个根本的变化，一是可字节寻址，二是延迟有几个数量级的下降。因此，尽管可以沿用现有的面向块设备的文件系统，但软件栈开销所占比重太大，难以发挥 NVM 的低延迟优势。围绕如何使用 NVM 的持久性，学术界和工业界已进行了大量探索，大致有两个方向，即使用持久性堆[6~8]的内存管理模式和使用文件系统[8~14]的外存管理模式。持久性堆的管理模式涉及到编程模型、编译器、操作系统等整个软件生态的重构，工程浩大。短期来看，在兼容传统文件系统编程接口的基础上构建轻量级的 NVM 文件系统是较为现实的选择，因此最近几年，面向 NVM 的文件系统研究得到了极大关注[15,16]。

随着器件技术的不断成熟，应用单一 NVM 构建统一的单级存储结构，实现内存与外存的最终融合在理论上已可行。单级存储结构彻底消除了数据在内存与外存之间的流动，从而显著降低了传输延迟和功耗[17]。内/外存融合后，现在的操作系统中分别管理内存和外存的内存管理和文件系统两个功能模块将合二为一，现在的内存和外存两个完全不同物理存储介质也合二为一。然而要实现内存与外存的真正融合，需要重构整个软件栈，这是一个系统性工程，但由于缺少工业界的参与，进展缓慢，但毫无疑问内外存融合是终极目标。

2 原理与优势

持久性内存展现出了与传统的计算机内存和外存截然不同的特性，给计算机系统设计者带来新的机遇和挑战[18]。如何定位持久性内存在存储系统中的角色，合理的使用持久性内存，兼容现有应用，提升系统性能，是计算机系统结构设计者需要考虑的一大问题[19,20] 。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[18,21,22]，从而颠覆了传统计算机存储体系的内、外存架构。如图 1-1 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。

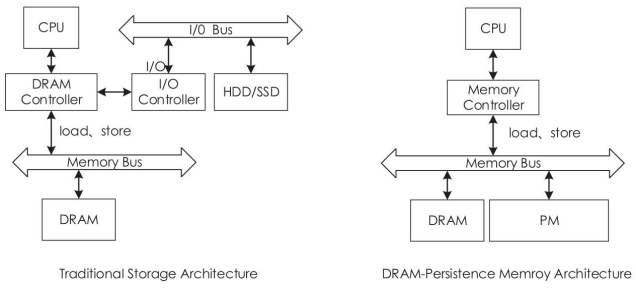

图 1-1

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。将持久性存储器直接作为内存应用到计算机的存储系统中，部分替代 DRAM 或者完全替

代 DRAM，通过内存管理单元（Memory Management Unit，MMU）进行管理，可以最大程度的发挥持久性内存的访问性能[23]。但是，由于持久性内存存储器的耐久性与 DRAM 还存在差距，直接使用内存管理单元 MMU 进行管理会加速持久性内存的磨损。CPU 高速缓冲存储器（Cache）的存在会给持久性内存带来数据一致性问题[24]。要发挥持久性内存的数据非易失特性，需要操作系统和应用软件能够感知易失和非易失内存，并将持久性对象和易失性对象进行区分，这会对现有的计算机系统和应用编程模型带来很大的挑战[25]。

第二种，持久性内存作为存储设备使用。文件系统是当前计算机系统中管理存储设备的最常用方式，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[26,27]。但持久性内存性能相比于磁盘有了巨大的提升，对数据一致性保护也有更高的要求[28]。根据磁盘的性能特性进行设计的传统文件系统在性能和数据一致性等方面都无法完全适用于持久性内存。所以，使用文件系统管理持久性内存需要根据持久性内存特性对现有的文件系统重新设计，从而出现了很多基于持久性内存的文件系统。

现有基于持久性内存的文件系统，在持久性内存被直接连接到内存总线上，能够直接通过 load、store 指令对其进行访问的前提下，主要着眼于解决一致性挑战。 同时，去除通用块层等冗余的软件层次，减少了文件访问过程中的软件开销，绕过内存中的页高速缓存，避免了数据在页高速缓存和持久性内存之间的额外拷贝[29]。另外，利用持久性内存的 CPU 可直接访问特性，现有的一些持久性内存文件系统使用内存映射文件时，将持久性内存上文件的数据页直接映射到进程的虚拟地址空间中，从而避免了数据从持久性内存到 DRAM 的拷贝[30~32]。使用内存映射进行文件访问，能够为应用提供更灵活的文件访问方式，简化开发，也能够减少文件数据在访问过程中的拷贝开销，提升文件访问性能。另外，利用内存映射文件可以进行同一台计算机上多个进程间的数据共享，内存映射文件是单个计算机上多个进程进行互相通信的最有效方法[33,34]。但是，由于需要动态地分配页表项用于记录地址映射信息，现有的持久性内存文件系统在通过使用 load、store 命令访问内存映射后的文件时仍会有缺页中断开销，影响了持久性内存文件系统中内存映射文件的访问性能。

3 研究进展

新型非易失存储器出现开始，由于其出色的性能，便有研究开始尝试将其纳入内存系统之中，作为一个慢速的内存使用，以缓解 DRAM 容量扩展能力不足，成本高，能耗高等一些问题[35,36]。目前最常用的新型非易失存储器 NAND Flash 只支持页（page）粒度的读写操作[37]，访问特性类似于传统的块设备，通过针对 Flash 特性优化 DRAM 和 Flash 之间的数据传输方案，一些研究通过将 Flash 作为交换（Swap）空间[38,39]，构建对象存储[31,40]，使用内存映射文件[34]等一些方法，将 Flash 纳入到内存系统，对整个内存子系统进行容量的扩展。新型可字节寻址的非易失存储器，例如相变存储器等新型存储器，可以像 DRAM 一样可字节寻址，出于兼容性，安全性等一些因素，研究者们提出了利用文件系统对持久性内存进行管理的方案[41,42]。一些研究通过类似 RamDisk 的方式对持久性内存进行封装，通过增加一个软件转换层，利用传统的块设备文件系统使用持久性内存。一些设计了针对持久性内存进行设计的文件系统，减少文件系统带来的软件开销，并为持久性内存提供数据保护[43]。 本小节将从数据一致性保证、降低软件栈开销、内存管理与文件系统的融合、分布式文件系统和文件系统的安全与可靠机制等几个方面介绍关于基于持久内存的文件系统的国内外相关研究状况。

3.1 数据一致性保证

数据一致性是文件系统的基本功能，也是核心功能，没有数据一致性的保证，程序的正确性也将得不到保证。持久性数据的一致性有两层含义：a）持久性，即保证在掉电或系统失效前已经将数据从易失性存储区域（如高速缓存）写回到持久性外存中；b）顺序性，即保证数据是按照程序语义确定的依赖关系先后有序写入 NVM 中，不能乱序。传统的文件系统主要采取写前日志（WAL）或写时拷贝（COW）等技术保证数据的一致性，但这些技术相对 NVM 而言同样存在开销过大的问题。此外，

由于 NVM 直接挂在内存总线上，CPU 可以直接通过访存指令读写持久性外存，无须像传统的二级存储系统那样必须先经过内存。为了保证一致性，需要及时将高速缓存中的数据刷回（flush）到 NVM 中，频繁的 flush 操作对性能影响很大，是需要研究的新问题。

BPFS[44]和 PMFS[45]是两个早期和著名的 NVM 改进文件系统的例子，旨在通过 POSIX 接口提供对 NVM 的有效访问。这两个系统都是为内存总线连接的 NVM 存储而设计的，并攻克与 NVM 相关的常见问题，比如高效的一致性机制，易失性处理器缓存的一致性和 NVM 优化结构。一方面，BPFS 提出了 epoch barrier 机制，以加强向 NVM 写入数据时的顺序和保持一致性，同时避免缓存刷新。BPFS 还提出了一种短路式的影子分页，即对传统影子分页进行细粒度 NVM 友好的重新设计，相比于传统的影子分页，SCSP 解决了单一的一个更新可能会递归向父节点执行直到根节点的递归更新问题，能够更高效的利用持久性内存的可字节寻址特性。另一方面，PMFS 使用细粒度日志来确保元数据更新的原子性，同时采用写时复制技术来确保文件数据更新的原子性。PMFS 还为文件系统页面提供内存保护，将它们标记为只读，并允许内核代码仅在必要时（在小时间窗口期间）通过操作处理器的写保护控制寄存器来更新它们。Jian Xu 等人提出了一种基于混合型内存的文件系统——NOVA[46]，其同时使用内存与持久化内存。NOVA 文件系统是一种提供强数据一致性保证的高效文件系统，NOVA 在持久性内存上除了保证元数据的一致性，还可以保证文件数据的一致性。为了高效的达到这一目的，NOVA 并没有使用传统日志结构文件系统的实现方式，其重新设计了一种针对持久性内存的日志结构，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 降低软件栈开销

对 NVM 而言传统文件系统的软件栈开销过大，因此，降低软件栈开销也就成为了 NVM 文件系统研究的主要方向。比如，传统文件系统中的块设备层和驱动层对 NVM 而言就是多余的[47, 48]。此外，减少持久性外存与工作内存、内核空间与用户空间之间的拷贝也形成共识[46]。NVM 可字节寻址，因此 NVM 文件系统可以不需要页缓存（page caching）而使用 DAX 技术直接访问 NVM，从而减少了存储栈中 NVM 与 DRAM 之间不必要的数据拷贝。DAX 的核心是零拷贝的内存映射机制（I/O memory mapping，mmap），与传统文件系统中的 mmap 不同，DAX 中的 mmap 是将持久化的数据直接映射到进程的虚拟地址空间，完全不需要页缓存。而传统文件系统中的 mmap 在将文件映射到用户空间时，虽然减少了一次数据拷贝，但依然需要页缓存。DAX 已引入到 Linux 内核中，可有效挖掘 NVM 的低延迟特性[49]。DAX 技术利用了 NVM 的字节寻址能力，但频繁的映射迫使应用程序静态预留部分存储区域且自行管理，影响了存储空间的利用率[50]。此外，由于 NVM 的写性能差，如何扬长避短也是研究的方向之一。

为了充分利用持久性内存的读写优异性，PMFS 被设计成一个轻量且高效的文件系统，其直接使用 CPU 的 load/store 指令对持久性内存进行访问操作，直接规避了内核的 page cache，块设备层等软件层面。同时，PMFS 在 x86 体系结构下提供了大页的支持（huge page），减少了 TLB 的使用量并且加快了虚拟内存的寻址。PMFS 中使用直接访问的方式对持久性内存进行访问，DRAM 和持久性内存之间的数据传输通过内存拷贝进行，大大减少了数据传输过程中的软件开销。

现有块设备文件系统在内存映射文件访问过程中，由于页表中映射信息不存在，会触发缺页中断，缺页中断处理程序为所访问的地址空间分配页表项，记录映射信息，并将所访问的数据加载到内存之中。PMFS 中对传统的内存映射文件访问方式进行了优化，减少了缺页中断中数据拷贝的开销，但仍需要为所访问的地址空间分配页表项，记录地址映射信息，缺页中断的开销依然存在。 Jian Xu 等[46]针对持久性内存的一致性问题，对 PMFS 进行了优化，利用持久性内存的快速随机访问特性，提出了每节点独立日志，使用链表链接日志项减少了日志的空间开销。

Jiaxin Ou 等[51]，在 PMFS 的基础上，针对持久性内存的读写不对称特性，提出在内存中使用写缓存，设计了相应的写缓存判断模型判断哪些写

操作需要进行缓存，发挥了了持久内存高读性能的同时，隐藏了持久性内存的写延迟。HM Sha 等[52]提出了 SIMFS（Sustainable In-Memory File System），在文件读写过程中，利用内存地址映射单元进行数据的寻址。Jian Xu 等和 Jiaxin Ou 等的在内存映射文件上沿用了 PMFS 中的内存映射方案，缺页中断开销依然存在。而 HM Sha 等的方案没有对内存映射文件进行优化，缺页中断开销也依然存在。

Rohan Kadekodi 等人设计了 SplitFS[53]，这是一种用于持久内存（PM）的文件系统，与最先进的 PM 文件系统相比，它可以显著降低软件开销。SplitFS 在用户空间库文件系统和现有内核 PM 文件系统之间提出了一种新的职责分割。用户空间库文件系统通过拦截 POSIX 调用、内存映射底层文件以及使用处理器装载和存储服务读和覆盖来处理数据操作。元数据操作由内核 PM 文件系统（EXT4 DAX）处理。SplitFS 引入了一个称为重链接的新原语，以有效地支持文件追加和原子数据操作。SplitFS 提供了三种一致性模式，不同的应用程序可以从中选择，而不会相互干扰。与 NOVA 持久文件系统相比，SplitFS 减少了高达 4 倍的软件开销，与 EXT4 DAX 相比减少了 17 倍。

Ian Neal 等人发现在 PM 文件系统上文件映射占据了 IO 路径开销的一个重要部分[54]，不能再通过页面缓存来缓解。于是设计了四种不同的 PM 优化映射结构，以探索 PM 上与文件映射相关的不同挑战。作者对这些映射结构的分析表明，PM 优化哈希表结构 HashFS 平均性能最好，在实际应用程序工作负载上提供了高达 45% 的改进。

3.3 内存管理与文件系统的融合

文件系统的访问是基于文件索引的，纯软件方式，效率低。NVM 可字节寻址，借助与工作内存相关的软硬件特性（如 MMU）来加速文件系统的访问成为研究方向之一；此外，通过扩展内存管理的功能，对某些持久性外存对象进行高效管理也是努力的方向，也为内外存融合奠定了一定的基础。

Wu 等人[10]提出了第一个内外存融合管理的文件系统 SCMFS，旨在利用操作系统中现有的内存管理模块来辅助管理文件系统空间。SCMFS 利用内存管理单元（MMU）将文件系统的逻辑地址空间映射到 NVM 的物理地址空间上，并尽可能地为每一个文件分配连续的地址空间，以加速文件系统对连续数据块的访问；此外还采取了空间预分配机制及相应的垃圾回收机制以减少内存管理开销。Sha 等人[47,52]也基于类似的思想提出了一个真正的用户空间 NVM 文件系统 SIMFS。与传统的用户空间文件（如 FUSE）不同，SIMFS 不需要依附内核级文件系统来处理用户请求，因此对文件的访问更高效。SIMFS 仍沿用标准的 POSIX 接口，文件能够以 O(1) 的时间复杂度暴露给用户空间，与文件大小无关。每个打开的文件都有一块连续的虚拟地址空间，由一个称之为文件页表的分级页表来管理，该页表保存文件中每个数据页的地址映射信息，利用 MMU 地址转换硬件可以快速定位文件中的数据。SIMFS 以页表的形式访问元数据，而其他文件系统（如 PMFS）的元数据是以 B-tree 的形式存放，前者的查找速度要快很多。

由于内存访问的高效性，学术界也在积极探索以现有的内存管理模块为主，在其中加入对持久性的支持。Kannan 等人[55]在这方面做了一些尝试，针对容量扩展和对象存储（object storage）提出了一种持久化的虚拟内存（pVM）。pVM 是一个系统软件抽象，可以实现操作系统层内存容量自动扩展并且在 NVM 不同区域灵活地放置数据。

3.4 分布式文件系统

NVM 的出现不仅带给本地文件系统的设计一系列思考，同时也影响了分布式文件系统的设计，尤其是随着 RDMA 网络技术的应用，更使得分布式文件系统的设计发生了根本变化，主要体现在两个方面：a）传统分布式文件系统面向的是普通硬盘和 TCP/IP 网络，延迟相对较大，对软件的开销不敏感；而 NVM 和 RDMA 的延迟大幅降低，对软件开销十分敏感，并且基于 RDMA 的远程数据访问甚至低于 NVM 的本地访问；b）通过 RDMA 技术可以直接读写远程 NVM，无须远程 CPU 的干预，且 NVM 不仅字节可寻址而且具有非易失性，这些特性都深刻影响了分布式文件系统可靠性、一致性等功能的设计。

Islam 等人[56]提出的 NVFS 是一个较早利用 NVM 的特性来优化 HDFS 文件系统的工作，它将 NVM 和 RDMA 结合起来用于加速 HDFS。它挖掘 NVM 的字节寻址能力，通过在 I/O 操作

中使用访存操作原语减少计算和 I/O 时对访存的争用；它在底层文件系统中使用 NVM 存放 Spark 的 job 输出和 HBase 的写前日志。可见，它并没有真正重构 HDFS，只是利用 NVM 的特性做了一些改进，而 HDFS 本身设计厚重，使得 NVFS 很难充分发挥 NVM 和 RDMA 的硬件特性。

Yang 等人[58]提出了第一个具有全部特征的分布式文件系统 Orion，将 RDMA 深度集成到文件系统的整个设计中，对远程存储节点的访问完全无须远程节点 CPU 的干预，没有任何软件开销。将网络和存储功能集成到驻留内核的一个层次中，用于处理文件数据、元数据和网络访问。Orion 还尽可能将持久性数据迁移到本地，并使用一个新颖的委派分配策略管理空闲空间，通过这些方法提升挖掘的局部性，减少远程的 RDMA 访问。

Lu 等人[14]提出了一种新型的基于 RDMA 与 NVM 的持久内存分布式文件系统 Octopus，核心思想也是将 NVM 存储与 RDMA 通信两者的特征紧密地耦合起来，纳入统一的设计。对于数据操作，Octopus 利用 NVM 构建了一个可以直接访问的全局共享持久内存池，避免了在本地文件系统上再叠加一个分布式文件系统层，减少了数据拷贝过程，并且使用客户端的主动读写来减少服务器工作负载。对于元数据操作来说，Octopus 引入一种自识别远程过程调用 RPC 来减少文件系统层与网络层的通信延迟。

3.5 文件系统安全与可靠机制

NVM 支持 XIP（eXecute In Place）特性，该特性会增加不经意写操作的风险。NVM 的写次数有限，也要防止恶意的磨损攻击，同时还要能够抵御内存错误和软件错误，具有一定的鲁棒性。

PRAMFS[58]是一个专为 NVM 设计的具有写保护功能的持久性内存文件系统，可以针对文件系统中的所有文件进行直接的、同步的且不阻塞的 I/O 操作。PRAMFS 还支持 XIP，读写操作可以直接在内存中执行，数据无须在用户空间和内核空间之间来回拷贝，而是直接将其所在的内存映射到用户空间，大大加快了应用程序的启动时间。PRAMFS 的另一主要功能是写保护，文件映射之后的页表项通常标记为只读，在写操作之前要先将受影响的页临时标记为可写并上锁，写操作结束以后再将页表项标记为只读，避免了由于内核缺陷引起的错误写操作导致文件系统损坏情况的发生。PMFS 也提供了类似的保护来自内核不经意写操作的功能。Aerie 为解决这些挑战，采取的方案是将文件系统分为不可信部分（libFS）和可信部分（TFS），libFS 提供文件系统接口，包括命名和数据访问；TFS 通过确保元数据的完整性和同步来支持互不信任程序之间的协作。读元数据和读写普通数据可以直接通过 libFS，而对于元数据以及受保护的普通数据的写操作必须经过 TFS。

在容错方面，为了使文件系统能够抵御介质错误和软件缺陷带来的风险，常常采取快照、校验等方法。针对 NVM 文件系统中连续快照之间的一致性，Zheng 等人[59]提出 HMVFS，一种构建在空间高效的内存分层文件系统树（SFST）上的基于混合内存的版本文件系统。整个文件系统的每个快照都是源于文件系统树根的一个分支，并使用高度受限的 COW 友好的 B-tree 来重用不同版本之间重叠的数据块。此外，通过挖掘 NVM 的字节寻址特性来避免原数据更新的写方法问题。 Xu 等人[60]也在文件系统的容错方面做了有益的探索，设计了 NOVA-Fortis，它在之前 NOVA 版本的基础上通过引入快照、复制、校验和以及 RAID-4 奇偶保护等方法全面提升文件系统的容错能力。NOVA-Fortis 首次支持在应用程序执行 DAX 操作时还可以拍具有数据一致性的快照，且两者互不影响，快照数量可以无限，删除快照的顺序也可以随意。为了保证快照数据的一致性，在将所有的页标记为“只读”模式以前防止缺页中断的发生。NOVA-Fortis 也是利用现有的处理器提供的机制来检测内存错误，为了保护文件系统元数据，它为每个元数据结构存有两个版本（primary 和 replica 版本），并进行 CRC32 校验。对于普通文件数据，NOVA-Fortis 在 NVM 的一个预留区域为每个文件页保存了一个奇偶校验带，每个校验带也使用 CRC32 进行奇偶校验，分别存有两个副本且分开存放，从 而对应用程序通过 DAX 方式访问数据提供了最大的保护。

4 总结和展望

存储与计算、网络共同组成计算机系统的三要素，数据密集型应用的大量涌现，使存储面临前所未有的压力。NVM 的出现给存储系统带来巨大机遇，同时也带来诸多挑战，简单的器件替换无法挖掘 NVM 的性能优势，软件栈的重构也需要同步跟上。采取虚拟内存的方式管理持久性外存涉及到整个软件生态系统的革新，是一项庞大且耗时的工程。短期来看，随着 3D XPoint 芯片推向市场，仍然采取文件系统管理持久性外存的方式是现实可行的，过去十余年学术界开展了大量研究，一些技术已趋于成熟。本文梳理了近年来针对 NVM 文件系统的相关研究成果，发现这些工作主要集中在以下几个方面：数据一致性保证；降低软件栈开销；内存管理与文件系统融合；分布式文件系统；文件系统的安全与可靠机制等。

基于收集到的文献资料，本文罗列了几个基于持久内存的文件系统未来会发展的趋势：

a）扩展性问题。扩展性包括两个方面，一是文件系统对于小文件和大文件的支持都要好，二是文件系统对于单核和众核的支持都要好。持久化存储的数据一致性要求数据写入是有序的，要求频繁 flush 缓存中的数据，严重影响了文件系统的性能。因此，充分利用现有的硬件资源，包括指令（如 CLWB、CLFLUSHOPT）和特性（如 HTM）进行软硬协同、细粒度的文件系统设计可缓解文件系统的扩展性问题。

b）内外存融合。NVM 同时具有字节寻址特性和非易失特性，学术界和工业界都认为内外存融合是未来发展的必然趋势[37]。内外存合二为一后，板卡的集成度和可靠性提高，静态功耗减小，对计算机系统尤其是物联网设备和移动终端有很强的吸引力。尽管此前学术界进行了一些探索，但仍有很多问题没有答案，包括内外存融合后，如何合并原来的内存管理和文件系统两大操作系统模块的功能，如何统一编址、统一命名、统一保护，是否还需要超级块，怎样的数据结构能同时发挥出传统磁盘高吞吐和内存低延迟的优势，如何对大容量 NVM 进行高效管理等。因此，单级存储架构下工作内存和持久性外存的协同管理仍是需要探索的问题。

c）分布式文件系统。在电子商务、证券交易等时延敏感型应用中，基于 NVM 和 RDMA 的分布式系统有着显著的优势，高效的容错机制和事务处理仍然是设计的重点。由于 NVM 和 RDMA 都有着极低的访问延迟，且 NVM 可字节寻址，数据的复制和一致性联系更加紧密。比如，利用多副本作为写时拷贝日志，可以降低一致性保证开销。

参 考 文 献

[1] Wong H S P, Raoux S, Kim S. Phase change memory. Proceedings of the IEEE, 2010, 98(12): 2201-2227.

[2] Chang M, Lee A, Lin C, et al. Read Circuits for Resistive Memory (ReRAM) and Memristor-based Nonvolatile Logics. in: The 20th Asia and South Pacific Design Automation Conference. Chiba, Japan: IEEE Computer Society Press, January 19-22, 2015. 569-574.

[3] Chen E, Apalkov D, Diao Z, et al. Advances and Future Prospects of Spin-Transfer Torque Random Access Memory. IEEE Transactions on Magnetics, 2010, 46(6): 1873-1878

[4] YANG J, KIM J, HOSEINZADEH M, et al. An empirical guide to the behavior and use of scalable persistent memory[C]//18th {USENIX} Conference on File and Storage Technologies({FAST} 20). 2020: 169-182. 3

[5] RUDOFF A. Persistent memory programming[J]. Login: The Usenix Magazine, 2017, 42(2):34-40. 3

[6] Hwang T，Jung J，Won Y. HEAPO: Heap-based persistent object store[J]. ACM Trans on Storage，2014，11(1) : article No. 3. [7] Volos H，Tack A J，Swift M M. Mnemosyne: Lightweight persistent memory[J]. ACM SIGPLAN Notices，2011，46(3) : 91-104. [8] Coburn J，Caulfield A M，Akel A，et al. NV-Heaps: making persistent objects fast and safe with next-generation，non-volatile memories[J]. ACM SIGPLAN Notices，2011，46(3) : 105-118.

[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 Symposium on Operating Systems Principles. New York: ACM Press，2009: 133-146.

[9] Dulloor S R，Kumar S，Keshavamurthy A，et al. System software for persistent memory[C]/ / Proc of the 9th European Conference on Computer Systems. New York: ACM Press，2014: article No. 15. [10]

Wu Xiaojian，Qiu Sheng，Reddy A L N. SCMFS: a file system for storage class memory and its extensions[J]. ACM Trans on Storage, 2013, 9(3) : article No. 7. [11] Sha E H M，Chen Xianzhang，Zhuge Qingfeng，et al. A new design of in-memory file system based on file virtual address framework［J］. IEEE Trans on Computers，2016，65(10) : 2959-2972. [12] Xu Jian，Swanson S. NOVA: a log-structured file system for hybrid volatile / non-volatile main memories[C]/ / Proc of the 14th USENIX Conference on File and Storage Technologies. Berkeley，CA: USENIX Association，2016: 323-338. [13] Sha E H M，Jia Yang，Chen Xianzhang，et al. The design and implementation of an efficient user-space in-memory file system［C］/ / Proc of the 5th Non-Volatile Memory Systems and Applications Symposium. Piscataway，NJ: IEEE Press，2016: 1-6. [14] Lu Youyou，Shu Jiwu，Chen Youmin，et al. Octopus: an RDMA-enabled distributed persistent memory file system [C]/ / Proc of USENIX Annual Technical Conference. Berkeley，CA: USENIX Association，2017: 773-785.

[15] 石伟, 汪东升. 基于非易失存储器的事务存储系统综述[J]. 计算机研究与发展，2016，53 (2) : 399-415. (Shi Wei，Wang Dongsheng. Survey on transactional storage systems based on non-volatile memory[J]. Journal of Computer Research and Development，2016，53(2) : 399-415.)

[16] 吴章玲, 金培权, 岳丽华, 等. 基于 PCM 的大数据存储与管理研究综述[J]. 计算机研究与发展，2015，52 (2) : 343-361. (Wu Zhangling，Jin Peiquan，Yue Lihua，et al. A survey on PCM-based big data storage and management[J]. Journal of Computer Research and Development，2015，52(2) : 343-361.)

[17] Meza J，Luo Yixin，Khan S，et al. A case for efficient hardware/software cooperative management of storage and memory[C]/ / Proc of the 5th Workshop on Energy-Efficient Design. Piscataway，NJ: IEEE Press, 2013: 1-7.

[18] Zhou P, Zhao B, Yang J, et al. A Durable and Energy Efficient Main Memory Using Phase Change Memory Technology. ACM SIGARCH Computer Architecture News, 2009, 37(3): 24-33.

[19] Tao C, Dejiao N, Yao H, et al. NVMCFS: Complex File System for Hybrid NVM. in: International Conference on Parallel and Distributed Systems. Wuhan, China: IEEE Computer Society Press, December 13-16, 2016. 577-584.

[20] Kwon J B. Exploiting Storage Class Memory for Future Computer Systems: A Review. IETE Technical Review, 2015, 32(3): 218-226

[21] Lee B C, Ipek E, Mutlu O, et al. Architecting Phase Change Memory as a Scalable DRAM Alternative. ACM SIGARCH Computer Architecture News, 2009, 37(3): 2-13.

[22] Qureshi M K, Srinivasan V, Rivers J A. Scalable High Performance Main Memory System Using Phase-Change Memory Technology. in: Proceedings of the 36th annual international symposium on Computer architecture. Austin, USA: ACM Association Press, June 20-24, 2009. 24-33.

[23] Kultursay E, Kandemir M, Sivasubramaniam A, et al. Evaluating STT-RAM as an energy-efficient main memory alternative. in: IEEE International Symposium on Performance Analysis of Systems and Software. Austin, USA: IEEE Computer Society Press, April 21-23, 2013. 256-267 [24] Zhang Y Y, Swanson S. A Study of Application Performance with Non-Volatile Main Memory. in: 31st Symposium on Mass Storage Systems and Technologies. Santa Clara, USA: IEEE Computer Society Press, May 30 - 5 June, 2015. 1-10.

[25] Coburn J, Caulfield A M, Akel A, et al. NV-Heaps: making persistent objects fast and safe with next-generation, non-volatile memories. in: Proceedings of the sixteenth international conference on Architectural support for programming languages and operating systems. Newport Beach, USA: ACM Association Press, March 5-11, 2011. 105-118.

[26] Pelley S, Chen P M, Wenisch T F. Memory persistency. in: ACM/IEEE 41th International Symposium on Computer Architecture. Minneapolis, USA: IEEE Computer Society Press, July 14-18, 2014. 265-276. [27] Lu Y, Shu J, Sun L, et al. Loose-Ordering Consistency for Persistent Memory. in: 32nd IEEE International Conference on Computer Design. Seoul, South Korea: IEEE Computer Society Press, October 19-22, 2014. 209-216. [28] Liu R S, Shen D Y, Yang C L, et al. NVM duet: unified working memory and persistent store architecture. ACM SIGARCH Computer Architecture News, 2014, 42(1): 455-470.

[29] Volos H, Nalli S, Panneerselvam S, et al. Aerie: Flexible File-system Interfaces to Storage-class Memory. in: Proceedings of the Ninth European Conference on Computer Systems. Amsterdam, The Netherlands: ACM Association Press, April 14-16, 2014. 10-14.

[30] Yang J, Minturn D B, Hady F. When poll is better than interrupt. in: Proceedings of the 10th USENIX conference on File and Storage Technologies. San Jose, USA: USENIX Association Press, February 14-17, 2012. 3-3.

[31] Badam A, Pai V S. SSDAlloc: hybrid SSD/RAM memory management made easy. in: Proceedings of the 8th USENIX conference on Networked

systems design and implementation. Boston, USA: USENIX Association Press, March 30 - April 1, 2011. 211-224.

[32] Dulloor S R, Sanjay K, Keshavamurthy A, et al. System Software for Persistent Memory. in: Proceedings of the Ninth European Conference on Computer Systems. Amsterdam, The Netherlands: ACM Association Press, April 14-16, 2014. 1-15 [33] Richter Jeffrey, Nasarre Christophe. Windows 核心编程. 第五版. 葛子昂, 周婧. 北京: 清华大学出版社, 2008. 397-435 [34] Huang J, Badam A, Qureshi M K, et al. Unified Address Translation for Memory-Mapped SSDs with FlashMap. ACM SIGARCH Computer Architecture News, 2015, 43(3): 580-591.

[35] Caulfield A M, Mollov T I, Eisner L A, et al. Providing safe, user space access to fast, solid state disks. ACM SIGARCH Computer Architecture News, 2012, 40(1): 387-400.

[36] Wang L, Wang Q, Chen L, et al. Fine-Grained Data Management for DRAM/SSD Hybrid Main Memory Architecture. IEICE Transactions on Information & Systems, 2016, 99(12): 3172-3176.

[37] Laura M G, John D D, Steven S. The bleak future of NAND flash memory. in: Proceedings of the 10th USENIX conference on File and Storage Technologies. San Jose, USA: USENIX Association Press, February 14-17, 2012. 2-2 [38] M S, M S M. FlashVM: Virtual Memory Management on Flash. in: Proceedings of the 2010 USENIX conference on USENIX annual technical conference. Boston, USA: USENIX Association Press, June 23-25, 2010. 14-14

[39] Chen X, Sha H M, Jiang W, et al. The design of an efficient swap mechanism for hybrid DRAM-NVM systems. in: International Conference on Embedded Software. Pittsburgh, USA: IEEE Computer Society Press, October 2-7, 2016. 1-10.

[40] Badam A, Pai V S, Nellans D W. Better flash access via shape-shifting virtual memory pages. in: Proceedings of the First ACM SIGOPS Conference on Timely Results in Operating Systems. Farmington, USA: ACM Association Press, November 3-6, 2013. 1-14.

[41] Yang J, Wei Q, Chen C, et al. NV-Tree: reducing consistency cost for NVM-based single level systems. in: Proceedings of the 13th USENIX Conference on File and Storage Technologies. Santa Clara, USA: USENIX Association Press, February 16-19, 2015. 167-181. [42] Peter S, Li J, Woos D, et al. Towards high-performance application-level storage management. in: Proceedings of the 6th USENIX conference on Hot Topics in Storage and File Systems. Philadelphia, USA: USENIX Association Press, June 17-18, 2014. 7-7. [43] Oikawa S. Non-volatile main memory management methods based on a file system. SpringerPlus, 2014, 1(3): 1-17.

[44] J. Condit et al.,“Better I/O through byte-addressable, persistent memory,” in Proc. 22nd ACM Symp. Oper. Syst. Princ., 2009, pp. 133–146.

[45] S. R. Dulloor et al.,“System software for persistent memory,” in Proc. 9th Eur. Conf. Comput. Syst., 2014, pp. 1–15.

[46] J. Xu and S. Swanson,“NOVA: A log-structuredfile system for hybrid volatile/non-volatile main memories,” in Proc. 14th USENIX Conf. File Storage Technol., 2016, pp. 323–338.

[47] Sha E H M, Jia Yang, Chen Xianzhang, et al. The design and implementation of an efficient user-space in-memory file system[C]/ / Proc of the 5th Non-Volatile Memory Systems and Applications Symposium. Piscataway, NJ: IEEE Press，2016: 1-6.

[48] Zhang Zheng, Feng Dan, Chen Jianxi, et al. The design and implementation of a lightweight management framework for non-volatile memory［C］/ / Proc of IEEE International Conference on Trust，Security and Privacy in Computing and Communications. Piscataway, NJ: IEEE Press, 2016: 1551-1558.

[49] Sehgal P, Basu S, Srinivasan K, et al. An empirical study of file systems on NVM[C]/ / Proc of the 31st Symposium on Mass Storage Systems and Technologies. Washington DC: IEEE Computer Society, 2015: 1-14.

[50] Kannan S，Gavrilovska A，Schwan K． Reducing the cost of persistence for nonvolatile heaps in end user devices[C]/ / Proc of the 20th IEEE International Symposium on High Performance Computer Architecture． Washington DC: IEEE Computer Society，2014: 512-523．

[51] Ou J, Shu J, Lu Y. A high performance file system for non-volatile main memory. in: Proceedings of the Eleventh European Conference on Computer Systems. London, United Kingdom: ACM Association Press, April 18-21, 2016. 12-25

[52] Sha H M, Chen X, Zhuge Q, et al. A New Design of In-Memory File System Based on File Virtual Address Framework. IEEE Transactions on Computers, 2016, 65(10): 2959-2972

[53] Rohan Kadekodi, Se Kwon Lee, et al.SplitFS: Reducing Software Overhead in File Systems for Persistent Memory, 2019: 494-508

[54] Ian Neal, Gefei Zuo, et al. Rethinking File Mapping for Persistent Memory, 2021: 97-111

[55] Kannan S，Gavrilovska A，Schwan K． pVM: persistent virtual memoryfor efficient capacity scaling and object storage[C]/ / Proc of the 11th European Conference on Computer Systems． New York: ACM Press，2016: article No. 13.

[56] Islam N S, Wasi-ur-Rahman M, Lu Xiaoyi, et al. High performance design for HDFS with byte-addressability of NVM and ＲDMA[C]/ / Proc of International Conference on Supercomputing. New York: ACM Press, 2016: article No. 8.

[57] Yang Jian, Izraelevitz J, Swanson S. Orion: a distributed file system for non-volatile main memory and RDMA-capable networks[C]/ / Proc of the 17th USENIX Conference on File and Storage Technologies. Berkeley, CA: USENIX Association, 2019: 221-234.

[58] Protected and persistent RAM filesystem (PRAMFS) [EB /OL]. [2018-04-05]. http: / /pramfs.sourceforge.net/.

[59] Zheng Shengan, Huang Linpeng, Liu Hao, et al. HMVFS: a hybrid memory versioning file system[C]/ / Proc of the 32nd Symposium on Mass Storage Systems and Technologies. Piscataway, NJ: IEEE Press, 2016: 1-14.

[60] Xu Jian, Zhang Lu, Memaripour A, et al. NOVA-Fortis: a faulttolerant non-volatile main memory file system[C]/ / Proc of the 26th Symposium on Operating Systems Principles. New York: ACM Press, 2017: 478-496.

[61] 徐远超，面向新型非易失存储的文件系统研究综, 计算机应用, 2021: Vol. 38 No. 6

[62] Yan Lei, Research on Persistent Memory File System Optimization, 华中科技大学硕士学位论文