分布式文件系统

姚建宇 阳修康 徐志伟 杨子江 陈华斌

文件系统作为操作系统的一个重要的部分，引起了越来越多学者的关注。传统的文件系统均是基于传统的硬件设备，例如HDD或者SSD进行设计。当新型介质到来后，原始的文件系统模式并不能最大化的发挥新型介质的优势。快速非易失性存储器（NVM），作为未来新一代的存储设备，将很快出现在DRAM的处理器存储器总线上。 最终的混合存储系统将为软件提供更高效的高带宽访问持久性数据，但是管理，访问和维护NVM中存储的数据的一致性带来了许多挑战。 现有的用于磁盘或固态磁盘的文件系统会引入软件开销，这些开销会掩盖NVM应该提供的性能。

自旋扭矩传输技术，相变技术，电阻性存储器以及英特尔和3D XPoint技术等新兴的非易失性存储器（NVM）技术有望彻底改变传统设备的I / O性能。目前，越来越多的研究使用混合DRAM / NVMM，该方案使得存储系统更为高效，但是同样为系统设计人员带来了许多挑战。如果要充分利用NVMM的高性能并有效地支持更灵活的访问模式，这些系统需要将软件开销降至最低，同时，它们还必须提供强大的一致性保证，确保应用程序需要并需要满足新兴内存的限制。由于常规文件系统是为磁盘的性能而构建的，并且依赖于磁盘的一致性保证以确保恢复后的正确性，所以其并不适用于混合存储系统。混合存储系统在两个方面都不同于传统的存储系统：NVMM与磁盘相比，性能大大提高，并且具有非易失的特性。而DRAM提供了更好的性能。从而内存从磁盘结合提供了不同的一致性保证。

引入NVM后，最大的优势在于使用了其持久性的特制。然而对于基于内存的文件系统，提供强大的一致性会付出高昂的代价。现代的CPU和内存系统可能会将存储重新排序以提高性能，从而在系统出现故障时破坏一致性。 作为补偿，文件系统需要从CPU的缓存中显式刷新数据以执行排序，这会增加大量开销从而降低系统的性能。

为了克服所有这些挑战，非易失性存储器加速（NOVA）日志结构的文件系统被提出。 NOVA采用传统的日志结构文件系统技术来利用混合存储系统提供的快速随机访问。这使NOVA支持大规模并发，减少日志大小并最大程度地减少垃圾收集成本，同时为常规文件操作和基于mmap的加载与存储访问提供了强大的一致性保证。

NOVA不记录数据，因此仅恢复过程需要扫描一小部分NVMM。 这也使NOVA可以在页面过时时立即对其进行回收，从而显着减少了垃圾回收的开销，并使NOVA即使文件系统几乎已满也可以保持良好的性能。

NVMM技术对文件系统设计人员提出了一些挑战。 其中最关键的一点是平衡内存的性能与软件开销之间的关系，在更新时使用一些强制方案来确保一致性。在性能方面，NVMM的低延迟更改了硬件和软件延迟之间的权衡。在常规存储系统中，慢速存储设备的延迟决定了访问延迟，因此软件效率并不会对系统性能造成非常大的影响。先前的工作表明，使用NVMM后，其软件开销相比较来说占比增高，从而浪费了NVMM可以提供的性能。在写顺序方面，现代处理器及其缓存结构为了提高性能，从而对存储操作进行重新排序。CPU的内存一致性协议可以保证内存更新的顺序，但是现有模型不能保证更新何时到达NVMM，从而使得电源故障可能会使数据处于不一致状态。在原子性方面，POSIX的文件系统语义要求许多操作是原子的。 例如，POSIX重命名要求如果操作失败，则旧名称或新名称的文件都不得更改或创建。重命名文件是仅元数据的操作，但是某些原子更新适用于文件系统元数据和数据。存储设备通常仅提供有关原子性的基本保证。 磁盘提供原子扇区写操作，处理器仅保证8字节的对齐存储是原子的。基于上述特性与挑战，研究者需要针对新介质来设计非新型文件系统。

NOVA是一个日志结构的POSIX文件系统，该文件系统基于LFS文件系统的优势，结合NVM新型设备进行设计。由于NOVA的目标是不同的存储技术，因此它与传统的基于磁盘的文件系统有着极大的不同点。首先，支持原子更新的日志很容易在NVMM中正确实现，但对于搜索操作（例如目录查找和文件内的随机访问）效率不高。然而支持快速搜索的数据结构在NVMM中很难正确有效地实现。其次，由于NVM的随机访问特性很好，所以清理日志的复杂性主要是由于需要提供连续的空闲存储区域，但是在NVMM中这不是必需的。 第三，对磁盘使用单个日志是有意义的（磁盘只有一个磁盘头，提高空间局部性是提高性能的关键因素），但它限制了并发性。 由于NVMM支持快速，高度并发的随机访问，因此使用多个日志不会对性能产生负面影响。首先，NOVA将日志和文件数据保存在NVMM中，并在DRAM中构建基数树以快速执行搜索操作，从而使NVMM中的数据结构简单有效。 NOVA中的每个inode都有自己的日志，允许跨文件并发更新而无需同步。 由于NOVA可以同时重播多个日志，因此这种结构在文件访问和恢复期间都允许高并发性。 NOVA还保证有效日志条目的数量很少，从而确保系统能够快速扫描日志。根据上文所致，NOVA是日志结构的新型文件系统。为了将数据原子写入日志，NOVA首先将数据追加到日志，然后原子更新日志尾部以提交更新，从而避免了日志文件系统的重复写入开销。 某些目录操作（例如目录之间的移动）将跨越多个inode，并且NOVA使用日记来原子更新多个日志。NOVA首先在每个inode日志的末尾写入数据，然后记录日志尾部更新以自动更新它们。除此之外，NOVA日志记录是轻量级的，仅涉及日志尾部的操作。在基于NVMM的存储中，顺序日志的局部性优势不太重要，因此NOVA使用4 KB NVMM页面的链接列表来保存日志，并将下一个页面指针存储在每个日志页面的末尾。而此种模式具有三个优点。 首先，分配日志空间很容易，因为NOVA不需要为日志分配大的连续区域。其次，NOVA可以以细粒度的页面大小粒度执行日志清理。第三，回收过时日志页只需要分配几个指针即可，从而大大降低系统开销。NOVA中的inode日志不包含文件数据。相反，NOVA使用修改后的页面写时复制，并将有关写入的元数据附加到日志中。元数据描述更新并指向数据页面。

图1显示了NOVA数据结构在其管理的NVMM区域中的模式分布图。 NOVA将NVMM分为四个部分：超级块、恢复inode、inode表，日志以及数据页。超级块包含全局文件系统信息，恢复索引节点存储恢复信息，索引节点表包含索引节点，日志提供目录操作的原子性，其余区域包含NVMM日志和数据页。我们在设计NOVA时考虑了可伸缩性：NOVA在每个CPU上维护一个索引节点表，日志和NVMM可用页列表，以避免全局锁定和可伸缩性瓶颈。

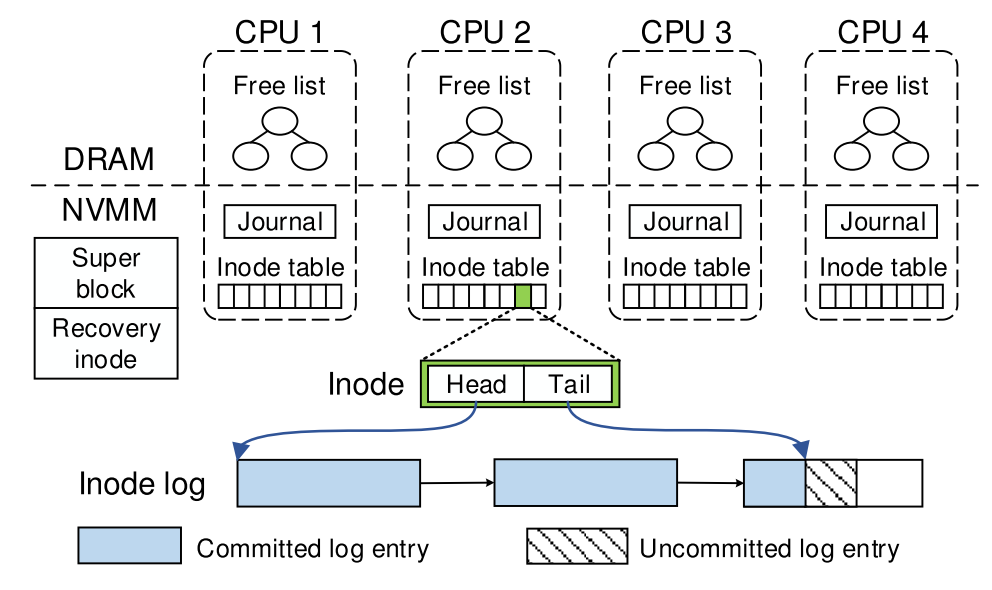


图1 NOVA数据结构模式分布图

NOVA作为NVM文件系统的一种，已经受到越来越多的研究者关注。然而，当多个线程对Manycore服务器上的单个共享文件执行I / O时，NOVA没有显示可伸缩性。这里存在两个两个问题：首先，由于对文件系统层中文件的粗粒度锁定，当多个线程向单个文件写入时限制了并行写入。其次，当多个线程读取一个文件时，每个读取器锁的获取都会使等待线程和块持有者的缓存行无效。

尽管NOVA是针对NVM的经过优化的文件系统，具有可伸缩的数据结构和强大的一致性保证，但它的每文件粗粒度锁降低了服务器环境中共享文件中的I / O性能。对于写操作来说，只有一个I / O线程通过调用down\_write获得文件的写锁。在保持写锁的状态下执行写操作，在此期间，其他写线程将被阻塞，等待释放该锁。当锁被锁保持器释放时，等待的写线程之一成功获取锁并执行写操作。粗粒度的写锁仅允许单个写程序执行写操作，从而防止了写程序之间的并发。对于读操作，系统使用down\_read函数的读取锁。与写锁不同，读锁可以由多个读取器持有，这意味着可以并行执行多个读取操作。但是，单读取锁定变量可能是多核环境中的性能瓶颈。当读取线程通过调用down\_read获取读取锁时，读取锁变量将增加1。此增量会使与其他等待线程的读锁变量以及锁持有者相对应的高速缓存行无效。在拥有数百多个内核的多核服务器中，此缓存行失效的开销可能很关键。

为了解决NOVA并发写的问题，于是June-Hyung Kim等研究者提出了基于范围的读写器同步机制的pNOVA，它选择性地仅在具有锁定持有者的重叠范围内阻止I / O操作。在具有基于范围的同步的pNOVA中，给定时间可以有多个操作器同时工作，从而实现了写并发。在读取方面，我们通过部署多个范围锁来减轻由于单个读取计数器而导致的缓存行无效开销。为了有效地找到重叠的范围，我们尝试在NOVA中申请基于间隔树的范围锁定。间隔树是用于检测重叠范围的众所周知的数据结构。但是，间隔树中的范围检查操作通过保持树的互斥锁开始，然后执行树插入和遍历。我们发现，间隔树的粗粒度锁定在多核服务器上的范围检查过程中成为瓶颈。接下来，我们建议基于原子操作的范围锁要比基于间隔树的范围锁执行得更好。基于原子操作的范围锁使用硬件支持的原子操作，以确保内存顺序。当NOVA采用上述基于范围的同步并且多个编写器并行执行写操作时，文件系统的一致性可能会被破坏。多个写入器必须同时在日志上写入，竞争单个日志指针。我们通过提交标记来保证文件系统的一致性。 在每次写入操作结束时，pNOVA都会通过在条目上放置提交标记以及内存围栏和高速缓存行刷新指令来提交日志条目。然后，在恢复阶段，只有带有提交标记的条目才能够成为恢复候选对象。这种基于提交标记的方案完成了同时写入多个文件的事务，并在断电后使文件系统保持一致。

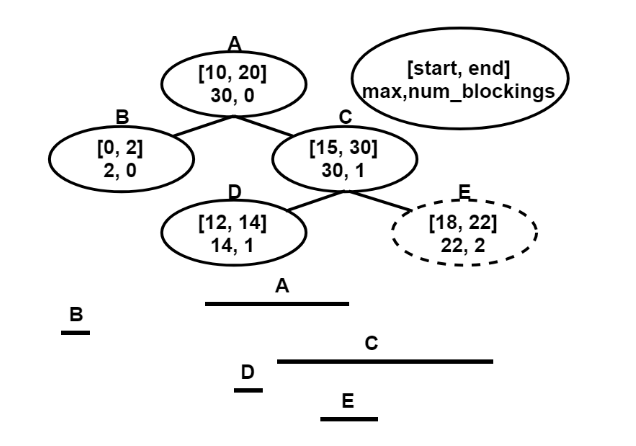


图2 间隔树方案图

间隔树是一种常见的数据结构，旨在查找与任何给定间隔重叠的所有间隔，这也是Linux内核常用的数据结构。如图2所示。其中每个间隔表示为树中的一个节点。一个节点由间隔的开始和结束组成。当插入一个节点后，系统根据其键进行排序排序。插入时，它将按顺序遍历以查找所有重叠间隔。范围锁定是仅锁定对象一部分的同步机制。存在基于Linux内核补丁形式的间隔树的范围锁定实现。在范围锁定机制中，每个范围锁定都表示为一个间隔树节点，并且有一个互斥锁来保护间隔树。要在单个文件上允许并行I / O，我们可以将范围锁定应用于文件系统。我们将基于间隔树的范围锁定原语结合pNOVA。由于不需要永久存储锁定状态，所以每个文件均定义了一个间隔树，该间隔树位于DRAM中。在文件系统I / O操作的开始，pNOVA使用写入请求的开始和结束页号作为节点的开始和结束值来生成间隔树节点。 然后，pNOVA遍历树以查找与给定间隔重叠的间隔。 在遍历期间，pNOVA将重叠间隔的数量存储到其节点。遍历后，pNOVA使用起始值作为排序键将节点插入树中。但是，对范围锁定的锁定和解锁操作是在持有间隔树的互斥锁的同时完成的。 间隔树的粗粒度互斥可能会严重破坏多核服务器中的并行性，特别是在具有快速存储设备的NVM文件系统中，开销会更大。

为了解决上述问题，我们提出了一种基于硬件原子操作的范围锁定，它比基于间隔树的范围锁定更有效。具体来说，我们定义段，即文件的一组连续页面。 段大小是page大小的倍数。对于文件的每个段，定义了一个32位变量，该变量用于该段的读写器锁定。 最左边的位是写锁。如果正在写入段，则该位为1，否则为零。 剩余的31位用作读取器计数器，其结构如图3所示，

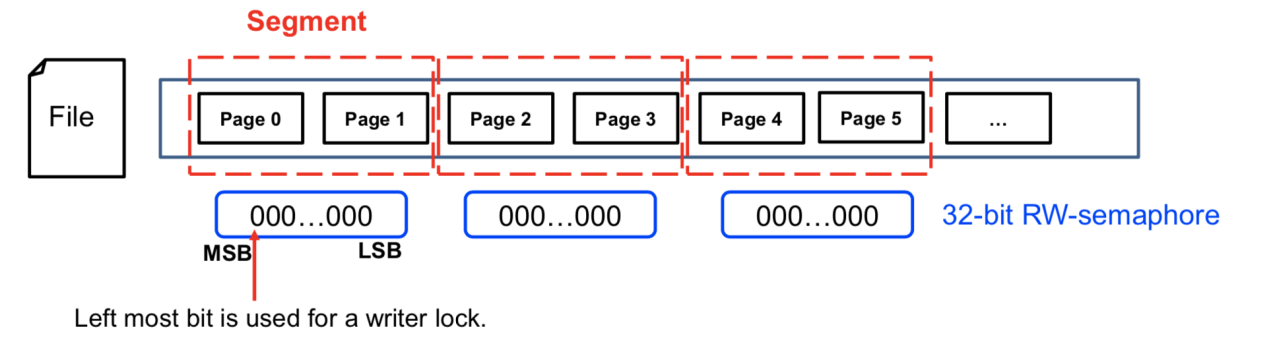


图3 基于段的范围锁原子操作

要解锁写入者和读取者锁定，系统不需要重复尝试原子操作。对于写程序解锁，我们只需使用clear\_bit原子操作清除最左边的位。当有线程在进行写操作时，写者或读者锁的获取都无法成功，所以清除最左边的位始终是安全的，在读取器解锁中，我们仅使用atomic\_dec将读取器计数器减少1。为了保证范围锁定的正确性，我们在每个原子操作的前后分别调用smp\_mb\_before\_atomic和smp\_mb\_after\_atomic。该函数为操作系统函数，用于保证操作严格按照顺序执行。根据测试可知，基于原子操作的范围锁定粒度更小，其比基于间隔树的范围锁定具有更少的开销。

Jian Yang 等人提出了Orion，一种用于基于NVMM的存储的分布式文件系统。通过采用简洁的设计，并利用NVMM和基于RDMA的高速网络的特性，Orion在保持NVMM字节寻址能力的同时，提供了高性能的元数据和数据访问。他们的评估表明，Orion的性能可与本地NVMM文件系统相媲美，并且在很大程度上优于现有的分布式文件系统。

在专为基于块的设备设计的分布式文件系统中，媒体性能几乎是数据路径上性能的唯一决定因素。与其余存储堆栈相比，磁盘的冰河性能（硬磁盘和固态磁盘）激励了磁盘访问周围的复杂优化（例如排队，分条和批处理）。这也使设计人员无需对网络效率，CPU利用率和本地性进行类似的积极优化，同时将其推向易于开发和维护的软件体系结构。

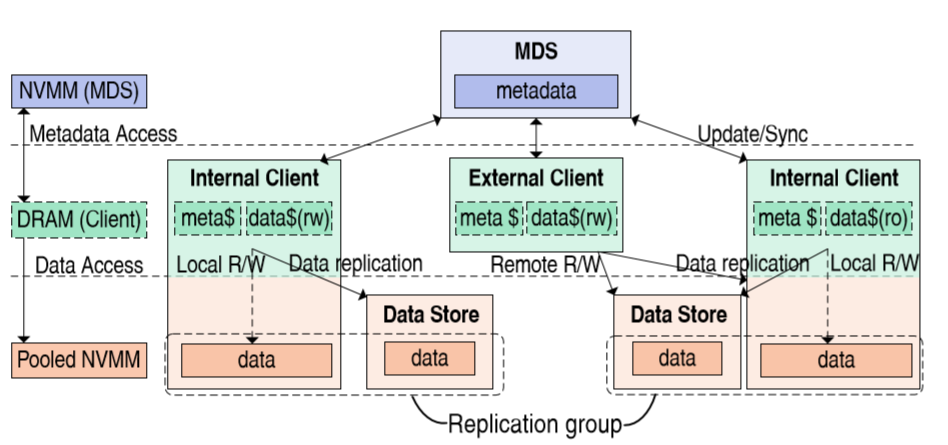


图4 Orion集群组织 Orion集群由元数据服务器，客户端和数据存储组成

Orion的目标是与具有RDMA功能的网络连接的系统。它尽可能使用RDMA来加速元数据和数据访问。一些现有的分布式存储系统使用RDMA作为数据访问的快速传输层，但并未将其深深地集成到他们的设计中。其他系统更广泛地适应了RDMA，但是为对象存储提供了与文件系统功能不兼容的自定义接口，例如不受限制的目录和文件范围，符号链接和文件属性。 Orion是第一个将RDMA深度集成到其设计的各个方面的全功能文件系统。大量使用RDMA意味着CPU不参与许多传输，从而降低了CPU负载并提高了处理传入请求的可伸缩性。特别是，将RDMA与NVMM配对可以使节点直接访问远程存储，而无需任何目标端软件开销。

另一方面，Orion预计可能还需要解决其他可伸缩性问题，例如RDMA连接管理和RNIC资源争用，以允许进一步扩展Orion。这也是未来探索的一个重要方向。

1. 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.
2. Xu J, Zhang L, Memaripour A, et al. NOVA-Fortis: A fault-tolerant non-volatile main memory file system[C]//Proceedings of the 26th Symposium on Operating Systems Principles. ACM, 2017: 478-496.
3. Kim J H, Kim J, Kang H, et al. pNOVA: Optimizing Shared File I/O Operations of NVM File System on Manycore Servers[C]//Proceedings of the 10th ACM SIGOPS Asia-Pacific Workshop on Systems. ACM, 2019: 1-7.
4. Lu Y, Shu J, Chen Y, et al. Octopus: an RDMA-enabled distributed persistent memory file system[C]//2017 {USENIX} Annual Technical Conference ({USENIX}{ATC} 17). 2017: 773-785.
5. Yang J, Izraelevitz J, Swanson S. Orion: a distributed file system for non-volatile main memory and RDMA-capable networks[C]//17th {USENIX} Conference on File and Storage Technologies ({FAST} 19). 2019: 221-234.