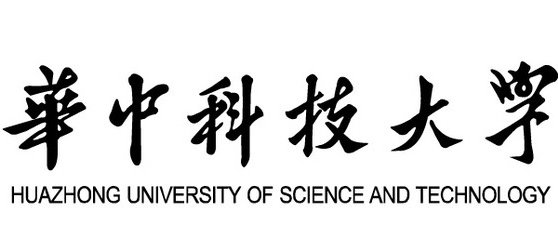
****

**操作系统分析与设计课程论文**

**（NOVA文件系统概述）**

|  |  |
| --- | --- |
| **学院** | 武汉光电国家研究中心 |
| **班级** | 硕1704班 |
| **小组** |  |
| **姓名** |  |
| **学号** |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 2018 | 年 | 1 | 月 | 23 | 日 |

**摘要**

快速非易失性存储器（NVM）将很快将会和DRAM一起出现在处理器内存总线上。由此产生的混合存储器系统将为软件提供亚微秒，高带宽的性能来访问持久性数据，但管理，访问和维护存储在NVM中的数据的一致性带来了许多挑战。 为旋转磁盘或固态硬盘构建的现有针对NVM的文件系统引入了软件开销，这会掩盖NVM应该提供的性能，但是现在提出的NVM文件系统要么引发类似的开销，要么不能为应用程序提供所需的强一致性保证。

我们提供NOVA，一个文件系统，旨在最大限度地提高混合存储系统的性能，同时提供强大的一致性保证。 NOVA采用传统的日志结构文件系统技术来利用NVM提供的快速随机访问的特点。 特别是，它为每个inode维护独立的日志以提高并发性，并将文件数据存储在日志以外，以最大限度地减少日志大小并减少垃圾收集成本。 NOVA的日志提供了元数据，数据和mmap原子性，并将注意力集中在简单性和可靠性上，将复杂的元数据结构保留在DRAM中以加速查找操作。 实验结果表明，在写入密集型工作负载中，相比于最先进的文件系统，NOVA提供了22％至216倍的吞吐量提升，并且与提供同样强大的数据一致性保证的文件系统相比，提高了3.1至13.5倍。

# 介绍

新兴的非易失性存储器（NVM）技术，如自旋转矩传输，相位变化，电阻式存储器，英特尔和美光的3D XPoint 技术等都将使I / O性能发生革命性的变化。 研究人员提出了将NVM集成到计算机系统中的几种方法，最令人兴奋的提议是将NVM与传统的DRAM一起放置在处理器-主存总线上，从而形成混合的易失性/ 非易失性主存储器系统。将速度更快，易失性的DRAM与稍微慢一些，但是密度更高的非易失性主存储器（NVMM）结合起来，提供了结合两种技术的最佳特性的存储系统的可能性。

混合DRAM / NVMM存储系统给系统设计者带来了许多机遇和挑战。这些系统需要最小化软件开销，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，并且同时它们必须提供应用程序需要的强大的一致性保证，并尊重新兴存储器的局限性（例如有限的程序周期）。

传统的文件系统不适用于混合存储系统，因为它们是为磁盘的性能特性（旋转或固态）而构建的，并且依靠磁盘的一致性保证（例如，扇区更新是原子的）来保证正确性[47]。 混合存储器系统与传统存储系统不同的地方在于：NVMM提供比磁盘更大的性能提升，而DRAM提供比NVMM更好的性能，虽然DRAM没有持久性。 内存提供了不同的一致性保证（例如，64位原子存储）。

提供强大的一致性保证对于基于内存的文件系统来说尤其具有挑战性，因为维护NVMM中的数据一致性可能代价高昂。现代的CPU和内存系统可能会将存储重新排序到内存，以提高性能，在系统故障的情况下打破一致性。 为了弥补，文件系统需要显式刷新CPU缓存中的数据以执行排序，增加了大量开销，并浪费了NVMM提供的改进的性能。

克服这些问题是至关重要的，因为许多应用程序依靠原子文件系统操作来确保自己的正确性。 现有的主流文件系统使用日志，shadow pageing或日志结构化技术来提供原子性。但是，日志记录通过将存储设备的写入次数加倍来浪费带宽，shadow paging的文件系统需要从受影响的叶节点到根的级联更新。实施这两种技术都要求严格的排序要求，以降低性能。

日志结构文件系统（LFS）将小型随机写入请求分组为一个更大的顺序写入，即硬盘和基于NAND Flash的固态硬盘（SSD）可以高效处理。 然而，传统的LFS依赖于连续的空闲区域的可用性，并且维护这些区域需要昂贵的垃圾收集操作。 因此，最近的研究表明，在NVMM上，LFS的性能比日志文件系统差。

为了克服所有这些限制，我们提出了Non-Volatile内存加速日志结构文件系统（NOVA）。NOVA采用传统的日志结构文件系统技术来利用混合存储器系统提供的快速随机访问。 这允许NOVA支持大规模并发性，减少日志大小，并最大限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/存储访问提供强大的一致性保证。

NOVA的以下几个方面将它与以前的结构化文件系统区分开来。NOVA为每个inode分配一个单独的日志，以在正常操作和恢复期间最大限度地提高并发性。 NOVA将日志存储为链接列表，因此它们不需要在内存中连续，并且使用对日志尾指针的原子更新来提供原子日志追加。对于跨多个inode的操作，NOVA使用轻量级日志记录。

NOVA不log数据，因此恢复过程只需扫描一小部分NVMM。 这也允许NOVA在页面过期时立即回收页面，显着减少垃圾收集开销，并允许NOVA即使在文件系统快满时也能保持良好的性能。

在描述NOVA时，本文做出如下贡献：

• 它扩展了现有的日志结构文件系统技术，以利用混合存储系统的特性。

• 它描述了原子mmap，一个简化的界面，用于将NVMM直接暴露给具有强大一致性保证的应用程序。

• 这表明，NOVA优于在混合内存系统上运行的现有Journaling，shadow paging和日志结构文件系统。

我们在基于硬件的NVMM仿真器上使用一系列的micro-和macro-的benchmarks来评估NOVA。 我们发现，在广泛的应用中，NOVA比现有文件系统快得多，并且在写入密集型工作负载中，与相同数据一致性保证的文件系统相比，性能是它们的3.1倍到13.5倍。我们还测量垃圾收集和回收开销，我们发现NOVA在高NVMM利用率水平下提供稳定的性能，在系统故障的情况下快速恢复。

# 背景

NOVA的目标是内存系统，包括新兴的非易失性存储器技术以及DRAM。 本节首先简要介绍了非易失性存储器技术以及它们给系统设计人员带来的机遇和挑战。 然后，我们讨论其他文件系统如何提供原子操作和一致性保证。 最后，我们讨论以前在NVMM文件系统上的工作。

## ****非易失内存技术****

当今新型的非易失内存技术如旋转移力矩随机存取内存（STT-RAM）、相变存储器（PCM）、 电阻式随机存储器（RRAM）以及3D XPoint内存技术等，它们能够提供快速、非易失并且是字节寻址的内存。这些内存在内存层次中具有不同的优势和弱势，例如，STT-RAM的延迟能够满足或超过DRAM并且它可能最终作为处理器最后一级高速缓存使用，由于其大的单元尺寸限制了它的容量而不能够替代DRAM；PCM和ReRAM比DRAM的密度更大，可能成为大的非易失主内存，然而由于它们的相对较长的延迟使它们不能够完全替代DRAM成为主存。英特尔和美光公司主推的3D XPoint内存技术据说能够提供比NAND Flash 1000倍的更高的性能。

## ****非易失内存软件的挑战****

NVMM（Non-Volatile Main Memory）技术给文件系统设计者带来了挑战。最关键的关注点在于平衡内存性能和软件开销，强制更新的顺序性以便确保一致性并且提供原子性更新。

**性能**  在传统的存储系统中，低速存储设备如磁盘的延迟占了访问延迟的主要部分，所以软件效率不是关键的。然而对于快速的NVMM，软件开销很快地成为内存延迟的主导，这挥霍（squander）了NVMM能够提供的性能。 由于NVMM内存提供低延迟并驻留在处理器内存总线上，软件可以通过loads和stores来直接访问它们。当今基于NVMM的文件系统通过使用直接访问（Direct Access，简称DAX）或原地执行（eXecute In Place，简称XIP）技术来绕过DRAM页高速缓存（page cache）来直接地访问NVMM，避免了存储栈中NVMM和DRAM间额外的拷贝。NOVA就是一个DAX文件系统。

**写重排序（Write Reordering）** 当今处理器和它们的高速缓存层次可能对更新操作进行重排序以便提升性能。现有的模型不能够确保更新是否写入NVMM，因此当系统断电的时候会使得数据不一致。那么，能够识别NVMM的软件通过明确地刷新高速缓存（cache）并且发送内存障碍（memory barriers）以便确保写顺序性。x86体系结构提供clflush指令以便刷新CPU cacheline，但是clflush是强一致性的，刷新成本很大并且clflush仅仅发送数据到内存控制器，并不保证数据到达非易失内存。英特尔mfence指令确保障碍前指令必须在障碍后的指令前完成。这并没有约束数据写回Memory的顺序。因此Intel公司提出了新的指令来修补这些问题，这些新指令包括clflushopt(clflust的更高效版本)，clwb(明确地写回一个缓存行而不使之无效)，以及PCOMMIT(使数据持久地存储到非易失内存中)。

**原子性**  POSIX类型的文件系统语义要求许多操作是原子性的，即要么全部执行要不都不执行。例如该POSIX rename操作就是原子性的，而不能出现执行和不执行的中间状态。重命名一个文件是仅对元数据进行操作，但是某些原子更新同时操作文件系统的元数据和数据。存储设备典型地只是提供初步的原子性保证，如磁盘提供原子的8字节（或更小）的扇区写。

## 构建****复杂的原子操作****

现有的文件系统使用各种各样的技术如journaling，shadow paging或log-structured技术来提供原子性保证。这些工作以不同的方式工作并招致了不同的开销。

**Journaling**  Journaling又名写优先日志（write-ahead logging）被广泛地使用于journaling文件系统中和数据库中。一个journaling文件系统记录所有的更新到journal（一个固定大小的循环buffer）中，然后再将更新写入文件系统目标位置。在系统断电时，通过重新执行journal以便恢复系统到一致性状态。但是Journaling需要写两遍数据：一个到log一个到目标位置。为了提高性能，journaling系统经常采用只journal元数据的模式提升性能。最近有工作提出反向指针（back-pointer）和去耦合持久顺序以便减少journaling的开销。

**Shadow paging**  Shadow Paging属写时拷贝（Copy On Wrie，简称COW）机制。Shadow paging文件系统主要依赖于它们的树结构以便提供原子性。不是通过原地更新数据的方式，shadow paging通过写一份受影响的页的新的拷贝到存储设备新的空闲空间中。就如我们用不同的PDF阅读器打开同一个PDF时，读取时它们共享同一份数据拷贝，只有当某个阅读器编辑修改后才会形成自己新的数据拷贝，即阅读器会让你在新的文件夹位置保存该PDF文件。但是shadow paging会有一个问题，就是迭代更新的问题。

**Log-structureing** 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。研究表明，SSDs在顺序工作负载中表现最佳。因此日志结构设计也适用于SSD。SFS、F2FS和RAMCloud均采用了日志结构存储机制。NOVA也采用了日志结构方式。

虽然LFS是一个优雅的想法，但是高效地实现它却是复杂的，因为LFS依赖于顺序写入磁盘的连续空闲区域。 为了确保这些地区的供应持续稳定，LFS不断清理和压缩日志以回收陈旧数据占用的空间。

日志清理会增加开销并降低LFS的性能。 为了减少清理代价，一些LFS设计将冷热数据分开，并应用不同的清理策略。SSD在顺序工作负载下也表现最佳，所以LFS技术也被应用于SSD文件系统。 SFS 根据文件块的更新可能性对文件块进行分类，并将具有相似“热度”的块写入同一个日志段，以减少清理开销。 F2FS使用多头日志记录，将元数据和数据写入独立的日志，并以高磁盘利用率将新数据直接写入脏段中的空闲空间，以避免频繁的垃圾收集。

RAMCloud是一个基于DRAM的存储系统，它将所有的数据保存在DRAM中，用于读取服务并在硬盘上保留一个永久版本。 RAMCloud将日志结构应用于DRAM和磁盘：它以日志结构的方式分配DRAM，实现比其他内存分配器更高的DRAM利用率，并将备份数据存储在磁盘上的日志中。

## 针对****NVMM的文件系统****

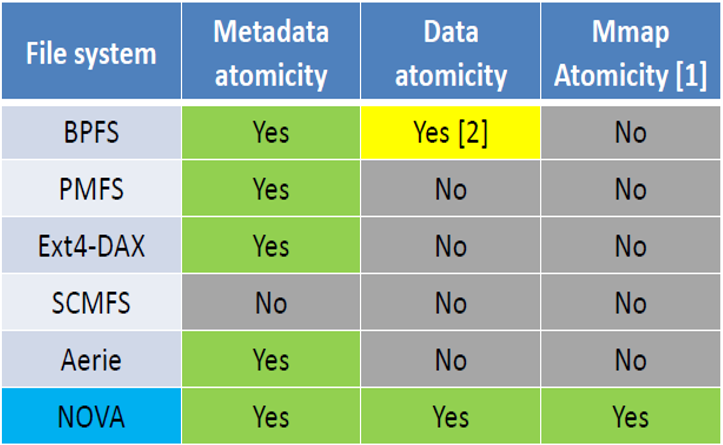


图 ‑1 针对NVMM的文件系统的比较

如图 2‑1所示，显示了6种文件系统在元数据、数据和mmap原子性的特性说明情况。

BPFS是一个shadow paging文件系统提供了元数据和数据的原子性。BPFS提出了硬件机制以便确保存储的持久性和顺序性，并采用了short-circuit shadow paging（SCSP）减少了shadow paging的开销。

PMFS是一个轻量级的DAX文件系统绕过块层和文件系统页高速缓存（page cache）以便提高性能。PMFS使用journaling来保证元数据的更新，因此元数据原子性得到了保证。

Ext4-DAX使用DAX功能扩展Ext4文件系统来直接访问NVMM，并且使用journaling技术保证元数据更新的原子性。

SCMFS使用操作系统虚拟内存管理模块并且映射文件到大量连续的虚拟地址区域，使得文件访问简单，但它不提供任何元数据和数据的原子性保证。

Aerie在用户空间实现了文件系统接口和功能以便提供NVMM数据的低延迟访问。而Aerie只journal 元数据而不提供数据或mmap的操作的原子性。

NOVA是一个日志结构、POSIX混合易失和非易失内存文件系统，提供上述三种操作的原子性保证。

# Nova 设计概述

NOVA是一个日志结构的POSIX文件系统，它建立在LFS的优势之上，并使它们适应混合内存系统的优势。 因为它的目标是不同的存储技术，NOVA看起来与为了最大化磁盘带宽而构建的传统的日志结构文件系统非常不同。

设计NOVA基于以下三个观察。第一，日志支持原子更新容易在NVMM上实现，但是对于查找操作不高效；第二，日志清理的复杂性主要来自需要提供连续空闲的存储空间，但是在NVMM中没有必要，因为NVMM中随机访问是便宜的；第三，对于磁盘而言，使用单个日志使合理的，但是这个限制了并发性。由于NVMMs支持快速高并发随机访问，使用多个日志并不影响性能。

**将日志保存在 NVMM 中索引保存在DRAM 中。**NOVA 将日志和文件数据保存在 NVMM 中, 并在 DRAM 中生成radix树以快速执行搜索操作, 从而使 NVMM 的数据结构简单高效。论文中使用radix树, 因为在 Linux内核中有一个成熟、经过测试、广泛使用的实现。Radix Tree的叶子指向日志中的条目, 依次指向文件数据。

**给每个 inode 各自的log。**NOVA 中的每个 inode 都有自己的日志, 允许跨文件进行并发更新而不进行同步。此结构允许在文件访问和恢复期间具有高并发性, 因为 NOVA 可以同时重播多个日志。NOVA 还保证有效日志项的数量很小 (按文件中的扩展盘数的顺序), 这样可以确保快速扫描日志.

**NOVA使用logging和轻量级的journaling来执行复杂的原子更新。**NOVA是日志结构的，因为它提供比Journaling和shadow paging更便宜的原子更新。 要将数据自动写入log，NOVA首先将数据附加到log，然后自动更新log尾指针以提交更新，从而避免Journaling文件系统的重复写入开销和shadow paging系统的级联更新成本。

某些目录操作（如目录之间的移动）跨越多个inode，NOVA使用Journaling来自动更新多个log。 NOVA首先在每个inode log的末尾写入数据，然后journal log尾部更新以自动更新它们。 NOVA journaling是轻量级的，因为它只涉及log 尾（而不是文件数据或元数据），并且没有POSIX文件操作对四个以上的inode进行操作。

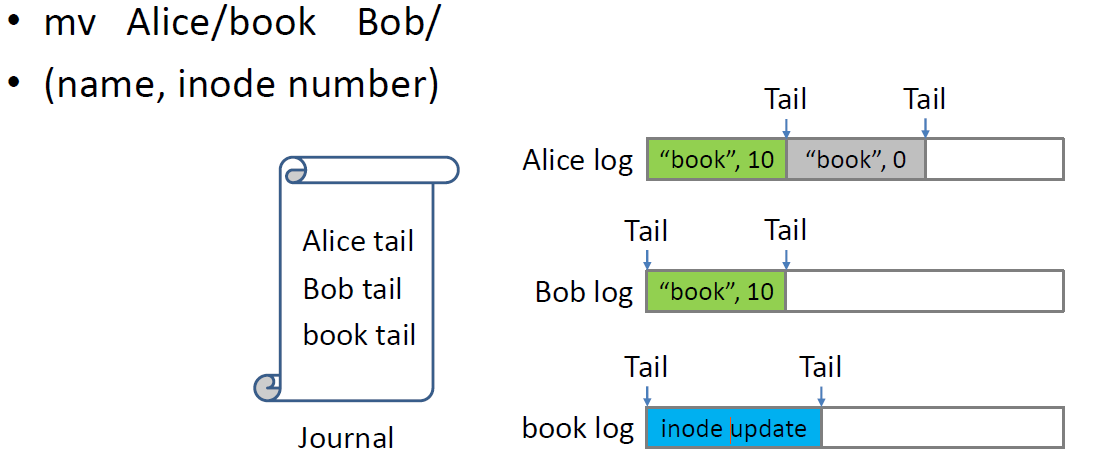


图 ‑1 NOVA的Journal操作

**以单链表的形式实验log。**顺序日志的局部优势在基于NVMM的存储中不那么重要，因此NOVA使用4 KB NVMM页面的链接链表来保存日志，并将下一页指针存储在每个日志页面的末尾。



图 ‑2 NOVA的log的形式

允许非顺序的日志存储提供了三个优点。 首先，分配日志空间很简单，因为NOVA不需要为日志分配大的连续区域。 其次，NOVA可以以精细粒度，页面大小的粒度执行日志清理。 第三，回收仅包含陈旧条目的日志页面只需要几个指针分配。

**不log文件数据。**NOVA中的inode日志不包含文件数据。 相反，NOVA使用写入时复制修改的页面，并将关于写入的元数据追加到日志中。 元数据描述更新并指向数据页面。

对文件数据使用copy-on-write是有用的，原因如下：首先，它会缩短日志，加速恢复过程；其次，它使得垃圾收集变得更简单，更高效，因为NOVA从不需要将文件数据从log中复制出来以回收log page。 第三，回收陈旧的页面和分配新的数据页面都很容易，因为他们只需要添加页面和从DRAM中删除页面。 第四，由于它可以立即回收过时的数据页面，因此即使在写入负载较重和NVMM利用率较高的情况下，NOVA也能保持性能。

# 实现NOVA

文章中在Linux内核版本4.0中实现了NOVA。 NOVA使用内核中现有的NVMM钩子并通过了Linux POSIX文件系统测试套件。源代码在GitHub上可以下载到，地址：https://github.com/NVSL/NOVA。 在本节中，我们首先描述整个文件系统布局及其原子性和写入顺序机制。 然后，我们描述NOVA如何执行原子目录，文件和mmap操作。 最后我们讨论NOVA中的垃圾收集，恢复和内存保护。

## ****NVMM数据结构和空间管理****

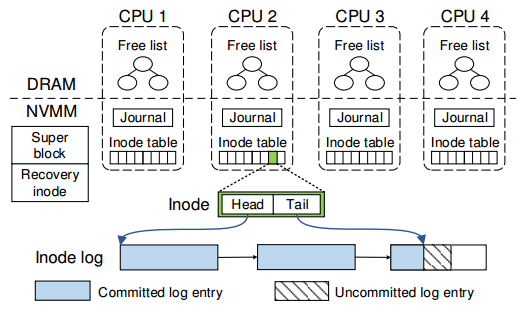


图 ‑1 NOVA数据结构的布局

如图 4‑1所示是NOVA数据结构布局。NOVA将NVMM分成四个部分：超级块（superblock）和恢复结点（recovery node），journals以及日志/数据页（log/data pages)。该超级块包含文件系统的全局信息，恢复结点存储恢复信息以便加速NOVA在正常关机情况下的重新挂载（remount），索引节点表（inode tables）包含索引节点，journals提供目录操作的原子性并且剩余区域包含NVMM日志和数据页。为了获得好的扩展性，NOVA在每个CPU中保持一个索引结点表、journal和NVMM空闲页列表（free page list）以避免全局锁和扩展性瓶颈。

**索引节点表（inode table）** NOVA初始分配每个索引结点表为2MB块组的索引结点，每个索引结点以128字节边界对齐，所以给定一个索引结点号能够很容易定位到目标索引节点。NOVA以round-robin顺序来分配新的索引结点到每个索引节点表中，这样索引结点能够比较平均地分布到众多索引节点表(inode tables)。为了减少索引节点表尺寸，每个NOVA inode包含一个有效位（valid bit）并且NOVA重新使用无效的索引结点用于新的文件或目录。每个CPU一个inode table可避免索引节点的分配争用并允许在故障恢复中并行扫描（parallel scanning）。每个inode包含日志的头尾指针，日志以4KB页的链表存在，并且日志尾指针总是指向最新的提交的日志项（log entry）。

**Journal** 一个NOVA的journal是一个4KB大小的循环缓存（circular buffer),并且NOVA用一个(enqueue,dequeue)指针对来管理每个journal。对于创建操作，NOVA journal目录的日志尾指针和新的索引节点有效位。NOVA允许每个核一次打开一个事务并且每个CPU允许并发事务执行。每次目录操作，内核虚拟文件系统层锁住所有受影响的inodes，保证并发事务不会操作同一个索引节点。

**NVMM空间管理** 为了使得NVMM分配和回收快速，NOVA将NVMM划分成pools,每个CPU一个，并且在DRAM中保持NVMM空闲页列表。如果在当前的CPU pool中没有可用的页时，NOVA将从最新的pool中分配页并且使用pool锁来提供保护。为了减少分配尺寸，NOVA采用红黑树（red-black tree）以便保持空闲列表按地址排序，允许高效合并以及提供O(log n）回收。

## 原子性和强制写顺序

NOVA提供元数据、数据和mmap更新的原子性，通过使用结合日志结构和journaling两种方法的技术实现。该技术使用如下三种机制:

**64位原子更新**：当前处理器支持易失性内存的64位原子写。NOVA使用64位原地写以便直接修改元数据，如文件的读访问时间等某些操作，并且使用64位原子写通过更新索引节点的日志尾指针来提交更新到日志中。

**Logging：**NOVA采用索引结点日志以便记录修改单个索引结点的所有操作。这些操作包括write，msync和chmod。每个索引结点的日志是相互独立存在的。

**轻量级Journaling：**对于目录操作要求改变多个索引结点的操作如create，unlink和rename，NOVA采用轻量级的日志提供原子性。任何时间内，任何journal的数据是小的，不会超过64字节。

**强制写排序** NOVA依赖三个写顺序规则确保一致性。首先，它在更新日志尾指针前提交数据和日志条目到NVMM中；第二，它在传输更新前提交journal的数据到NVMM；最后，在回收过时版本之前提交数据页的新版本。如果NOVA运行在一个支持clflushopt,clwb和PCOMMIT指令的系统上，将使用如下面的伪代码确保写顺序性。

new\_tail = append\_to\_log(inode->tail, entry);

// writes back the log entry cachelines

clwb(inode->tail, entry->length);

sfence(); // orders subsequent PCOMMIT

PCOMMIT(); // commits entry to NVMM

sfence(); // orders subsequent store

inode->tail = new\_tail;

如果平台不支持上述新指令，NOVA采用movntq非时序移动指令绕过

CPU高速缓存层以便直接写NVMM，并且结合clflush和sfence确保写顺序性。

## 目录操作

NOVA对目录操作进行了优化，包括link，symlink以及rename。它将目录分成两个部分：一个是存在NVMM中的目录索引结点日志，一个是存在DRAM中的radix tree。图 4‑2表明了这些部件的关系。目录日志（directory log）包括了两个条目：目录条目（directory entries，简称dentry）和索引结点更新条目（inode update entries）。Dentries包括子文件/目录的名字，它的索引结点号（inode number）以及时间戳（timestamp）。为了加速目录条目的检索，NOVA将每个目录索引结点保存在DRAM中的radix tree中。NOVA使用当前CPU的Journal去原子地更新目录日志的尾指针并且设置新的索引结点的有效位。

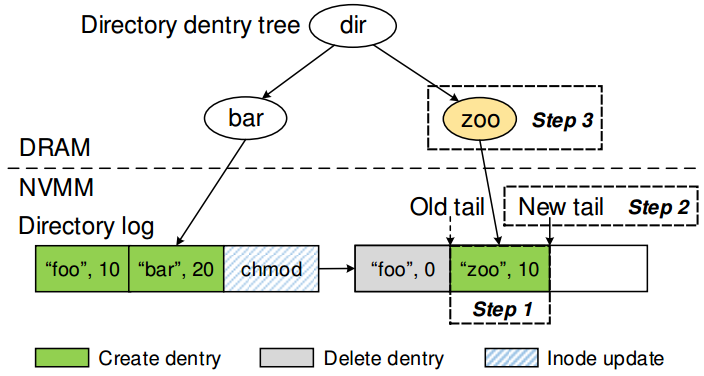


图 ‑2 NOVA目录结构

**创建文件** 1.在 inode table 里面为 zoo 选择并初始化一个未使用的 inode

2.将 zoo 的 dentry 添加到目录的 log 里面 3.使用当前 CPU journal 更新 dir 目录的 log tail 和给新的 inode 设置一个 valid 位。4将 zoo 添加到 radix tee 上面。

**删除文件** 在Linux中，删除文件需要两次更新：第一次减少文件inode的

链接数，第二次从封闭目录中删除文件。NOVA首先将delete dentry log entry附加到目录inode log中，然后，inode会更新文件inode log 的条目，然后使用journaling机制以原子方式更新两个log tails。 最后，它将更改传播到DRAM中目录的radix tree。

## 原子的文件操作

NOVA 的文件 log 包含两种，一个是 inode update entries，另一个 write entries，write entry 里面会描述 write operation 以及指向实际的 data page。如果一次写入太大，NOVA 会使用多个 write entries，并将它们全部追加到 log 后面，然后最后更新 log tail。

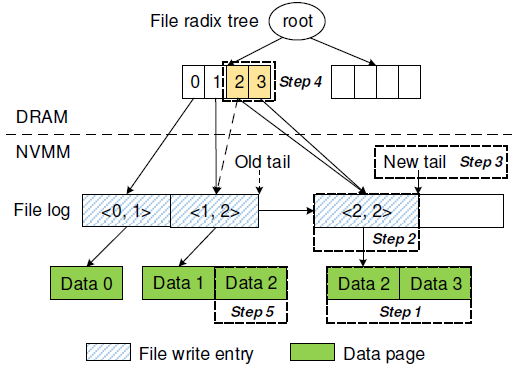


图 ‑3 NOVA的文件结构

图 4‑3展示的是一个文件写入例子，上面的 <0, 1> 这种的表示 <filepageoff, num pages>，也就是 page 的 offset 和有多少个 page。譬如 <0, 1> 就表示这个 page 的 offset 是 0， 有一个 page，也就是 4 KB。当我们要进行一次 <2, 2> 写入（也就是在 offset 为 2 的地方重新写入 2 pages），流程如下：

1. 使用 COW 技术将数据写入到 data page 上面

2.将 <2, 2> 追加到 inode log 上面

3.原子更新 log tail，提交这次写入

4.更新 DRAM 里面的 radix tree，这样 offset 2 就指向了新的 page

5.将之前旧的两个 page 放到 free list 里面

上面需要注意，虽然我们更新 log tail 是原子的，但并没有保证原子更新 log tail 和 radix tree，这里使用了一个 read-write lock，其实感觉并不高效，可以考虑考虑优化。

## 原子的mmap

DAX(Direct Access）文件系统允许应用程序通过load和store指令去直接访问NVMM，通过将物理的NVMM文件数据页映射到应用程序物理地址空间中。DAX-mmap绕过文件系统页高速缓存（page cache）并且避免了paging开销，程序员现有可用的原子性机制包括64位写、fences以及高速缓存刷新指令（clflush），仅用这些原语构建强健的非易失数据结构是非常困难的。为了解决这个问题，NOVA提出了atomic-mmap，它是一个具有强一致性的直接NVMM访问模型。由于NOVA对于文件数据使用写时拷贝技术并且立即回收过时的数据页（stale data pages），它不支持DAX-mmap。Atomic-mmap比DAX-mmap有更高的开销但是提供更强的一致性保证。

## 垃圾回收

NOVA的日志采用链表（linked list）形式并且仅包含元数据，是的垃圾回收简单高效。NOVA垃圾回收机制分Fast GC和Thorough GC。分别如下介绍。

**Fast GC** Fast GC不要求任何的拷贝（copying）。如果在一个日志页中的所有条目过时（dead），那么Fast GC通过从日志链表中删除该页来回收日志空间。如图 4‑4（a）展示一个快速日志垃圾回收的例子。

**Thorough GC** 在Fast GC日志扫描期间，NOVA记录每个日志空间存活日志（log）条目所占的空间比例。如果存活的条目所占的空间少于日志空间的50%，NOVA就利用Fast GC结束后的Thorough GC来拷贝获得条目到一个新的更加紧实的日志版本中，更新DRAM数据结构指向新的日志，然后使用新的日志原子地替换旧的日志，最后并回收该旧日志。具体例子图 4‑4（b）所示。

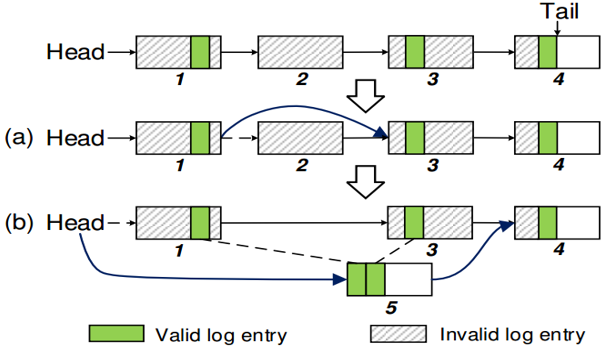


图 ‑4 NOVA log cleaning

## 关机和恢复

当NOVA重新挂载文件系统时，它将重构需要的DRAM数据结构。由于应用程序可能仅访问文件系统的一部分索引节点（inodes），所以NOVA采用一个称为延迟重建（lazy rebuild）的策略以便减少恢复时间。它推迟重建radix树和索引结点，直到文件系统第一次访问该索引结点时才重建。这个策略加速了恢复过程并且减少了DRAM的消耗。

**正常关机的恢复** 对于正常关机操作，NOVA将NVMM页分配状态保存到恢复索引结点日志（recovery inode’s log）中并且在接下来的remount操作中恢复该分配状态。这个过程中，NOVA不需要扫描任何的索引节点日志，所以恢复过程很快。

**失败恢复**在系统故障发生是，NOVA通过扫描索引结点日志来重建NVMM分配器信息。NOVA的日志扫描过程是快速的，原因有两点，其一是每个CPU的索引结点表和每个索引结点均有一个log允许日志恢复的巨大的并行性；第二点是因为NOVA的日志仅包含元数据而不包含数据页，日志很短。NOVA的失败恢复过程包括两步：

1. NOVA检查每个journal并且回滚任何未提交的事务以便恢复文件系统到一致性状态。
2. NOVA在每个CPU中运行一个恢复进程并且并行扫描索引节点表，为索引节点表中的每一个有效的索引节点执行日志扫描。

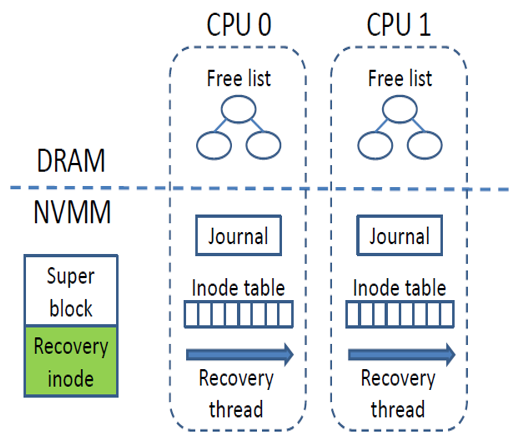


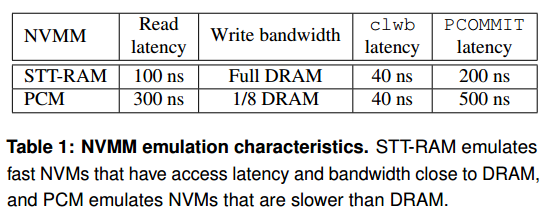
图 ‑5 NOVA关机恢复

## ****NVMM保护****

为了保护文件系统并且防止NVMM免受流浪写的持久腐蚀影响，NOVA必须确保只有系统软件才能够访问NVMM。NOVA采用PMFS相同的保护机制。在文件系统挂载时，整个NVMM区域被映射为只读。当NOVA需要写NVMM页时，它通过禁止（disable）处理器的写保护控制（CR0.WP）以便打开一个**写窗口**（**write window**）。当CR0.WP被清理时，内核软件以ring 0权限运行则可以写内核空间中标记为只读的页。写完后，NOVA又重置CR0.WP以便关闭该写窗口。

# 评价

NOVA采用了因特尔持久内存模拟平台（Intel PM Emulation Platform，简称PMEP）。1.用PMEP模拟了不同的NVM特征；2.模拟了clwb/PCOMMIT的延迟。



**NOVA提供了低延迟的原子性**。如图 5‑1所示：

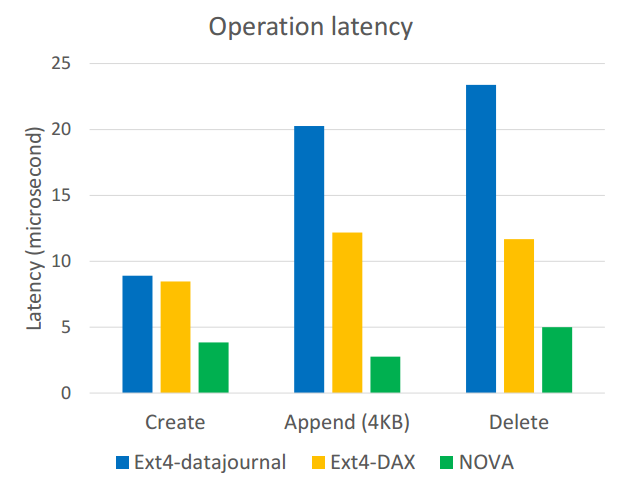
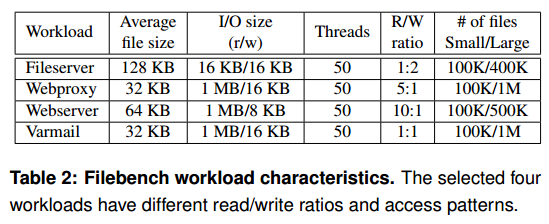


图 ‑1 NOVA的低延迟的原子性

NOVA采用了Filebench工作负载包括fileserver、webproxy、webserver和varmail去评估NOVA的应用程序级性能。Filebench工作负载特征见表2所示。



**NOVA获得了高性能并且有强数据一致性保证**。

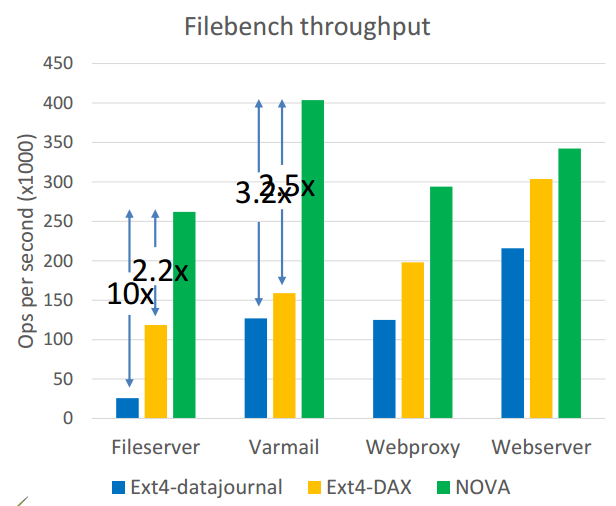


图 ‑2 Filebench的吞吐率

**垃圾回收效率** NOVA的性能随着运行时间的增加保持稳定，原因是在长期的运行过程中，Fast GC回收了主要的过时页（stale pages）。如所示：

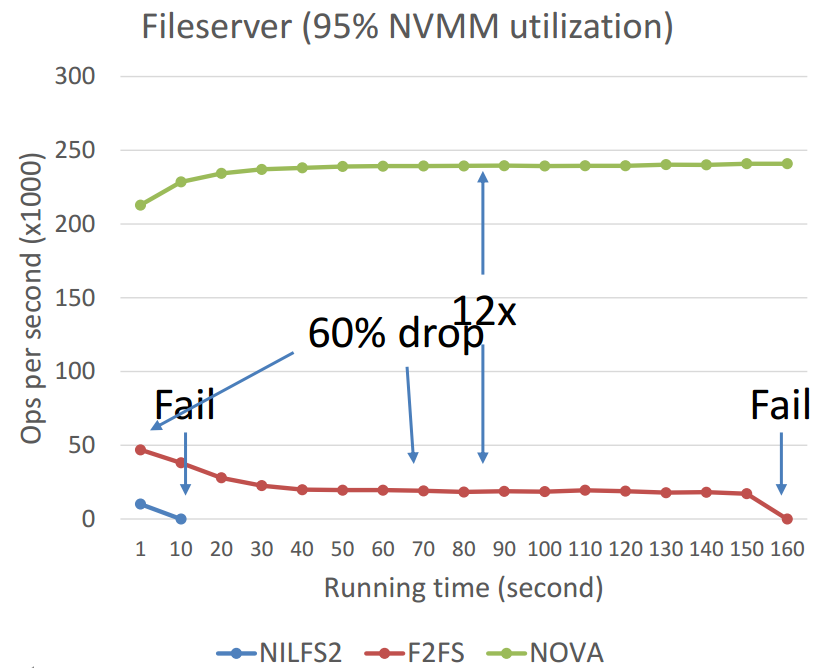
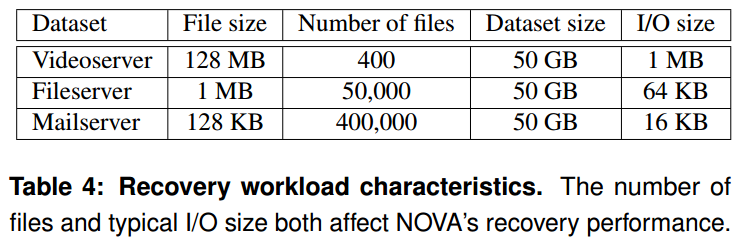
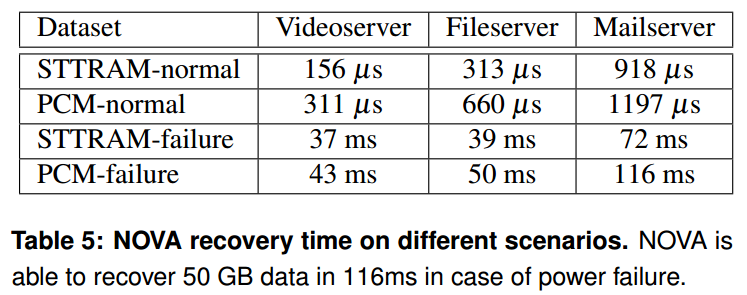


图 ‑3 NOVA的GC效率

**恢复开销** NOVA使用DRAM去维持NVMM空闲页列表（free page lists），所以在文件系统挂载时必须重建。NOVA通过延迟地重建索引节点信息，保持短的日志并且执行并行的日志扫描来加速恢复过程。为了测量恢复开销，采用了表4中的三种工作负载。每种工作负载代表了文件系统的一种不同的使用情况：Videoserver包含少量的大尺寸请求的大文件访问，mailserver包含了大量的小文件并且请求大小比较小，而Filesever介于两者之间。详情见表4

   
 表5总结了结果。正常关机，NOVA在1.2ms时间恢复文件系统，原因是NOVA不需要扫描索引结点日志。在系统故障后，NOVA的恢复时间随着索引结点的数量的增大而增长，文件越多，文件日志越多并且随着IO操作的进行文件越来越碎片化以致文件日志变得越来越长。基于STT-RAM上比基于PCM的恢复快是因为NOVA需要读日志以便重构NVMM空闲页列表，而PCM有比STT-RAM更高的读延迟。在PCM和STT-RAM中，NOVA能够在116ms时间

内恢复50GB数据，恢复带宽超过400GB/s。



# 总结

本文实现并描述了NOVA，一种为混合易失性/非易失性主存储器设计的日志结构文件系统。 NOVA扩展了LFS的思想，以利用NVMM，产生更简单，高性能的文件系统，支持快速高效的垃圾收集和系统故障的快速恢复。 文中的测量结果显示，NOVA在广泛的应用上大幅优于现有的NVMM文件系统，同时提供更强的一致性和原子性保证。

# 听报告心得

## NOVA-Fortis: 容错的非易失主存文件系统

非易失存储介质兴起，使得能够将常规DRAM与大量非易失性主存储器（NVMM）相结合，使系统存储性能得到巨大提升。充分实现NVMM的潜力需要根本改变系统软件管理，保护NVVM中的数据并提供访问。在此基础上Xu J博士等人研究并开发了NOVA文件系统，但是其在容错方面存在不足，因此Xu J博士等人进而改良NOVA，并发布了一个新的版本——NOVA-Fortis。NOVA-Fortis这个文件系统在面对由于软件错误导致的数据损坏时有既快又有弹性的处理。针对NVMM文件系统添加容错，将最先进的可靠性技术应用于NVMM文件系统以及量化这些技术的性能和存储开销方面的独特挑战，确定并提出解决方案。

NOVA-Fortis在NOVA文件系统的基础上通过快照和数据保护机制，提升容错性。NOVA-Fortis快照支持允许系统管理员在应用程序运行时获取文件系统的一致快照。 系统可以将快照安装为只读文件系统，或者将文件系统回滚到以前的快照。 NOVA-Fortis支持无限数量的快照，快照可以按任意顺序删除。

NOVA-Fortis通过将每个结构的两个副本保留为主要副本和副本并向两者添加CRC32校验和来保护其元数据。为了更新元数据结构，NOVA-Fortis首先将数据结构的内容复制到主数据结构中，并发出一个锁以确保数据写入NVMM。然后它对副本（tock）也是一样的。该方案确保在任何时候，两个副本中的至少一个被正确更新并具有一致的校验和。为了可靠地访问元数据结构，NORA-Fortis使用memcpy\_mcsafe（）将主副本复制到DRAM缓冲区中，以检测媒体错误。如果它找不到，它将验证两个副本的校验和。如果由于介质错误或校验和不匹配而检测到一个副本损坏，则通过复制另一个来恢复它。如果两个副本没有错误但不相同，则系统在上一次更新的ticktock和回传阶段之间fail，NOVA-Fortis将主要副本复制到副本，从而有效地完成中断更新。如果两个副本都损坏，则元数据将丢失，NOVA-Fortis将返回错误。

## FIRM：公平和高性能的永久内存系统的内存控制

持久化内存系统指的是采用的最近新型的非易失存储器的存储系统，它的特点是掉电数据不丢失。

持久化和非持久化的应用程序的区别在于：持久化程序具有一致性保障机制可以保证在掉电之后恢复原来的数据；其特点是高写入强度，低内存并行度。而非持久化程序就是其他普通的程序，在掉电之后丢失数据；其特点是以读为主。持久化程序包括了：数据库，文件系统，密钥值存储和持久性文件缓存。

如果现在系统上要运行持久化和非持久化的程序，这会带来以下问题：1、持续写入导致频繁进入”写队列释放”2、频繁的总线周期轮转导致额外的消耗

3、低并行度4、有失公平性

FIRM的整体框架如下图所示：

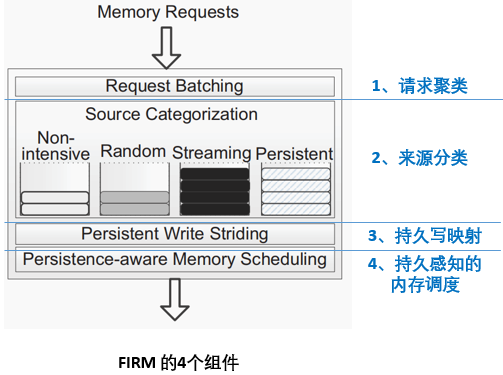


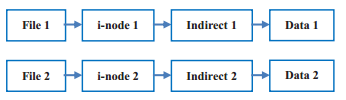
图 ‑1 FIRM架构

其中，来源分类就是是根据，读取的强度，行缓冲局部性，等标准进行分类。来源可以分为：非密集型，随机型，流式，和持久性的。非持久的来源可以分为：非密集型，随机型和流式的；非密集型的可以分为：低的读取强度，流式的就是连续读，随机的就是随机读。

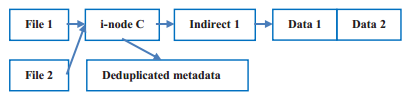
## 复合文件的文件系统：解耦文件和元数据的一对一映射以获得更好的性能

传统文件系统的文件与元数据采用一对一的映射方式（每个文件享有一个i-node节点）。但是其会存在冗余元数据信息、频繁访问小文件、预取限制等的缺点。

CFFS（Composite-file File System）将多个子文件(subfile)合并为一个组合文件（composite file）；同一组合文件下的小文件共用一个i-node、间接寻址块。



1. 传统文件系统的元数据与数据之间的关系



1. CFFS的元数据与数据之间的关系

图 ‑2 传统文件系统和CFFS的元数据组织方式

composite file的创建的过程如下：1.分配i-nodeC 2.复制、拼接子文件内容3.记录子文件大小、偏移位置、子文件的原始i-node信息等到扩展属性。为使用户可以通过原始文件名找到文件：子文件名重新映射到i-nodeC，给子文件创建subfile ID。CUID（CFFS unique IDs i-node的命名空间）中的高位作为组合文件的i-node number，低位作为子文件的subfile ID。

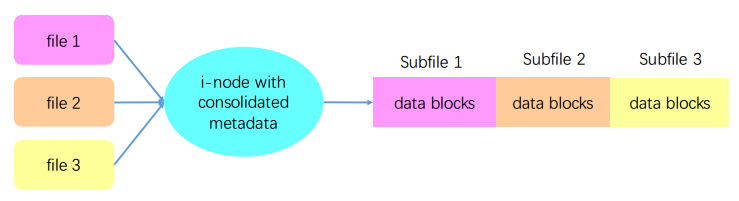


图 ‑3 复合文件的创建过程

## F2FS：Flash Friendly File System

F2FS (Flash Friendly File System) 是专门为基于 NAND 的存储设备设计的新型开源 flash 文件系统。特别针对NAND 闪存存储介质做了友好设计。F2FS 于2012年12月进入Linux 3.8 内核。F2FS 采用 log-structured的方案。修复了旧式log-structured文件系统的一些已知问题， (1) wandering tree (2)高clean开销。根据闪存内部结构和管理机制，F2FS的设计者增加了多种参数，不仅用于配置磁盘布局，还可以选择分配和清理算法，优化性能(并行IO提高性能)。

像NOVA一样，F2FS也对垃圾收集进行了设计，以便能够快速GC。F2FS使用多头日志记录，将元数据和数据写入独立的日志，并以高磁盘利用率将新数据直接写入脏段中的空闲空间，以避免频繁的垃圾收集。

## Falcon:在多SSD卷上扩展IO性能

大数据应用需要很高的吞吐量来处理大量的数据，因此对高性能的存储系统 产生了需求.基于闪存的固态硬盘（SSD）与磁盘相比具有很好的随机和顺序读写性能，因此是一种很有吸引力的选择。最近的趋势是使用多个SSD来获得更高的吞吐量1.图分析2.机器学习3.键值存储

Linux块层的不足：1.缺乏并行性。由于多个IO服务任务被迫在单个plug-list中执行，因此并行化这些任务的机会是有限的。 在Linux架构下，合并，标签分配和排序任务缺乏并行性。 且调度是以轮询的方式进行。2.合并和排序的效率很低。3.不可预测的阻塞。如果标签分配失败，则IO线程将阻塞等待空闲标签 。导致流水线中活跃的IO个数的不确定性

Falcon体系架构：1. **Falcon IO Management Layer：**将到达的IO批量的放入plug-list。只是将IO放入Plug-list当中，不进行处理。将pluglist分类，单次传输操作不进行排序。可以并行地对分类后的进行批处理，为每一个FBL创建一个Falcon线程。2. **Falcon Block Layer：**排序阶段：每驱动排序处理阶段：邻居合并、动态标签分配、调度完成阶段3. **Dynamic Tag Management:** 只在调度阶段才分配标签 .主要好处是提高队列利用率，因为允许更多的IO驻留在IO管道中而不需要获取标签.

## 用WOM代码来节约SSD擦出的成本

一次写记忆(WOM，write-once memory ) 代码首先在1982年被莱维斯特和沙米尔提出，其目的是为在一次写的存贮媒介介质(如穿孔卡和光盘)上记录信息多次。但是当施加了类似的约束，它们也可以应用于flash 存储器。只要保证每个单元的位值只能增加, 不能减少即可。

Reusable SSD采用Polar WOM代码。第一次写的时候几乎不进行修改，没有额外的开销，第二次写的时候一个逻辑页写入两个物理页。Reusable SSD有如下优点：1.在擦除之前增加写入操作2.扩展SSD的生命周期3.提升性能4.保留原有的容量和接口。

这篇文章推出了可重复使用的固态硬盘，这是一种实用的设计，用于应用第二次写入来延长SSD使用寿命，同时显着提高性能。这种设计是通用的，适用于当前的闪存架构，只需在FTL内进行微调即可，无需额外的硬件或接口修改。

尽管如此，随着技术在几个预期的方向上的进步，可重复使用的固态硬盘还能获得更多。允许更高程度的并行性的闪存体系结构可以容纳第三次或者第四次写入，分别将每个逻辑页面的4个和8个物理页面组合起来。 随着Polar WOM代码硬件实现的成熟，其编码开销将减少，从而实现更快的重试次数，并有可能使用成功概率更高的结构。 同样，更强的ECC可以补偿增加的BER，增加块在擦除之前可以被回收的寿命的百分比。

最后，大多数先前提出的延长SSD寿命的方案与可重用SSD的设计是正交的，并且可以与第二次写入相结合。 可重复使用的SSD所实现的性能改进可以掩盖那些引起额外延迟的方案的一些开销。

## TensorFlow：大规模机器学习系统

Tensorflow是一个能够满足模型在不同平台上不同性能以及资源需求而建立一个独立的系统，训练好的模型能够以在线服务或者在移动设备上运行等各种各样的形式投入产品。

TensorFlow的设计原则：在满足谷歌机器学习产品workload的同时让tensorflow更加灵活；提供一个简单的基于数据流图的编程抽象让用户可以在分布式集群，本地工作站，移动设备等上使用；在用户级代码中构建数据流图，让用户不修改核心系统的前提下能够比较自由地尝试不同的网络结构。

TF使用一个数据流模型代表了一个机器学习算法中的所有计算和状态，包括单独的数学运算，参数和他们的更新规则，输入的预处理。数据流图显式地表达了子运算之间的通信，从而使得并行地执行独立计算和在不同设备上的计算分配十分简单。它和batch dataflow systems的区别在以下两个方面：

* 支持在重叠子图（overlapping subgraphs）多并行执行
* 单个顶点可以有在图的不同执行之间共享的可变状态。

通过对PS的观察可以知道可变的状态在训练分非常大的模型的时候至关重要，因为这能让模型在非常大的参数规模下完成就地更新（in-place updates），并尽快地传播这些更新给并行训练步（steps）。有可变状态的数据流让tf能够模仿ps的功能，并且能够满足灵活性要求，因为这让host共享模型参数的机器上能够执行任意的数据流子图。所以，用户能够在不同的优化算法，一致性策略和并行策略上实验。

目前，机器学习和深度学习如日中天，方兴未艾。TensorFlow是非常红的一个深度学习框架。我认为非常有必要了解和学习一下，并且最近也有存储方向论文用CNN或者RNN进行数据压缩。有机会可以看一下。

## 使用PathDump简化数据中心网络调试

随着数据中心网络的不断发展，需要网络调试工具解决网络问题，现有的网络调试工具需要结合网内技术（如捕获每个数据包的每个交换日志，动态更新交换机规则等），实现越来越困难。因此这篇文章提出了一个截然不同的方法（Path Dump）：分配边缘设备和网络交换机之间的功能，尽可能地避免使用复杂的网内技术。

Path Dump是一个网络调试器，用于监测数据中心网络可能出现的问题。

其用于解决与链路相关的网络问题。Path Dump大约会占用4%网络吞吐量以及占用少量服务器CPU，内存和硬盘资源。

Path Dump设计基础：①交换机仅进行简单的操作。②边缘设备保存交换机标识符，并对引擎中的条目进行相应的更新。③边缘设备可以根据条目来触发和调试异常的网络行为。

Path Dump设计的核心思想是跟踪数据包的轨迹。①使用Cherry Pick的链路采样思想来跟踪数据包的轨迹。②边缘设备从接收到的分组报头中的提取交换机ID列表并记录在本地存储和查询引擎上。③每个边缘设备公开Host API,用于识别、触发和调试异常网络行为。

PathDump可以用来做路径的一致性检查、负载不均衡的诊断、无声随机数据包丢失、路由环路诊断和TCP性能异常测试。

## My VM is lighter than your container

与虚拟机相比，容器的需求量非常大，因为它们更加轻量级。但是不利的一面是，容器提供的隔离度比虚拟机的隔离度要低，人们不得已在虚拟机中运行容器以实现适当的隔离。在LightVM这篇paper中，作者们研究隔离(VMs)和效率（容器）之间是否存在严格的权衡。于是他们发现虚拟机可以像容器一样灵活，只要它们很小，而且工具栈足够快。通过使用专用应用程序的unikernel和Tinyx来实现轻量级的虚拟机，Tinyx是一个可以创建定制的，精简的Linux虚拟机的工具。由于虚拟化控制平面（工具堆栈）成为性能瓶颈，因此轻量级虚拟机本身并不足以确保良好的性能。

LightVM的作者们提供了一种基于Xen的新型虚拟化解决方案LightVM，该虚拟化解决方案针对快速启动时间进行了优化，无论活动虚拟机数量多少。 LightVM具有对Xen控制平面的完全重新设计，将其集中式操作转变为分布式操作，与管理程序的交互被降到最低。 LightVM可以在2.3ms内启动VM，与Linux上的fork / exec（1ms）相当，比Docker快两个数量级。 LightVM可以在适中的硬件上打包数千个LightVM guest虚拟机，其内存和CPU使用率与进程相当。

## 高效高可用的内存中的K-V存储

内存中的键/值存储（KV-store）是许多系统（如数据库和大型网站）的关键构建模块。 这样的系统的两个关键要求是效率和可用性，这要求KV存储每秒连续处理数百万个请求。 常用的可用性方法是使用复制，例如主备(PBR)，然而，要求M + 1次内存来容忍M失败。 这使得稀少的内存无法处理有用的用户作业。

内存KV存储这篇文章通过集成纠删编码来实现内存效率，而不是显着降低性能，从而构建高度可用的内存KV-store。一个主要的挑战是内存中的KV-store有很多分散的元数据。由于对元数据的过度小的更新，单个KV放置可能导致过度的编码操作和奇偶校验更新。这篇文章的方法即Cocytus通过使用针对小尺寸和分散数据（例如，元数据和密钥）的PBR的混合方案，而仅对相对大的数据（例如，值）应用擦除编码来解决该挑战。为了缓解像删除编码冗长恢复等众所周知的问题，Cocytus通过利用复制的元数据信息不断地为KV请求提供在线恢复方案。为了进一步证明Cocytus的有用性，作者们通过使用Cocytus作为一个快速可靠的存储层来存储数据库记录和事务日志，从而构建了一个事务层。已经将Cocytus的设计集成到Memcached中，并将其扩展到支持内存交易。使用具有不同KV配置的YCSB的评估表明，Cocytus在延迟和吞吐量方面具有较低的开销，能够容忍节点故障并能快速在线恢复，同时在容忍两次故障时节省33％至46％的内存。使用SmallBank OLTP基准测试的进一步评估表明，内存事务可以在吞吐量高的Cocytus上运行，具有高吞吐量，低延迟和低中止率，并能够快速恢复连续的故障。

# 本课程学习心得体会

通过对本课程的学习，学到了很多知识。

首先，是了解了各种各样的文件系统。例如针对易失和非易失内存的混合存储系统的NOVA和NOVA-Fortis文件系统。NOVA扩展了LFS的思想，以利用NVMM，产生更简单，高性能的文件系统，支持快速高效的垃圾收集和系统故障的快速恢复。 文中的测量结果显示，NOVA在广泛的应用上大幅优于现有的NVMM文件系统，同时提供更强的一致性和原子性保证。NOVA-Fortis实施的解决方案通过对文件系统进行一致的快照来提供备份，并提供重大保护，防止由于软件错误导致的媒体错误和损坏。FIRM，对持久性的内存进行控制。还有CFS（复合文件的文件系统）通过解耦文件和元数据的一对一映射以获得更好的性能。F2FS是一个面向flash的移动设备的文件系统。

还有几种很重要的分布式的文件系统。我了解了GFS。GFS是一个可扩展的[分布式文件系统](https://baike.baidu.com/item/%E5%88%86%E5%B8%83%E5%BC%8F%E6%96%87%E4%BB%B6%E7%B3%BB%E7%BB%9F)，用于大型的、分布式的、对大量数据进行访问的应用。它运行于廉价的普通硬件上，并提供容错功能。它可以给大量的用户提供总体性能较高的服务。虽然这个文件系统不开源，但是Apache根据Google的那三篇paper，可能也不止那三篇开发了大名鼎鼎的hdfs。还有Ceph，也是一种为优秀的性能、可靠性和可扩展性而设计的统一的、[分布式文件系统](https://baike.baidu.com/item/%E5%88%86%E5%B8%83%E5%BC%8F%E6%96%87%E4%BB%B6%E7%B3%BB%E7%BB%9F)。还有FaceBook为存储海量图片文件而开发的一种文件系统，可惜，忘了叫什么名字了，对不起老师了。

其次，了解了一些SSD的优化的方案。例如利用WOM编码的可重用的SSD，还有Falcon这种在多SSD卷上扩展IO性能的方案·······不一而足。

最后还有一些其它的涉猎比较广的东西。比如，最近人工智能、深度学习可谓红得发紫，有几位同学也简单介绍了TensorFlow这个Google的深度学习和机器学习的框架。随着大数据时代的来临，云数据中心的兴建，云计算也可谓方兴未艾，我在课堂上也了解了Xen和一种代替虚拟机的容器Docker，也有一位同学讲了轻量级的VM，还有其他专业同学介绍的一些在我知识盲区的东西······

通过本课程的学习，个人觉得，确实达到了了解操作系统前沿方向的目标，虽然说好多细节也不懂。另外，通过自己看论文和讲论文的过程，受益匪浅。

# 参考文献

1. Xu J, Swanson S. NOVA: A Log-structured File System for Hybrid Volatile/Non-volatile Main Memories[C]//FAST. 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. Yadgar G, Yaakobi E, Schuster A. Write Once, Get 50% Free: Saving SSD Erase Costs Using WOM Codes[C]//FAST. 2015: 257-271.
4. Kumar P, Huang H H. Falcon: Scaling io performance in multissd volumes[C]//2017 USENIX Annual Technical Conference (USENIX ATC 17). USENIX Association. 2017: 41-53.
5. A. Akel, A. M. Caulfield, T. I. Mollov, R. K. Gupta, and S. Swanson. Onyx: A protoype phase change memory storage array. In Proceedings of the 3rd USENIX Conference on Hot Topics in Storage and File Systems, HotStorage’11, pages 2–2,Berkeley, CA, USA, 2011. USENIX Association.
6. J. Arulraj, A. Pavlo, and S. R. Dulloor. Let’s Talk About Storage & Recovery Methods for Non-Volatile Memory Database Systems. In Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, SIGMOD ’15, pages 707–722, New York, NY, USA, 2015. ACM.
7. K. Bhandari, D. R. Chakrabarti, and H.-J. Boehm. Implications of CPU Caching on Byte-addressable Non-volatile Memory Programming. Technical report, HP Technical Report HPL-2012-236, 2012.
8. M. S. Bhaskaran, J. Xu, and S. Swanson. Bankshot: Caching Slow Storage in Fast Non-volatile Memory. In Proceedings of the 1st Workshop on Interactions of NVM/FLASH with Operating Systems and Workloads, INFLOW ’13, pages 1:1–1:9, New York, NY, USA, 2013. ACM.
9. M. J. Breitwisch. Phase change memory. Interconnect Technology Conference, 2008. IITC 2008. International, pages 219–221, June 2008.
10. F. Chen, D. A. Koufaty, and X. Zhang. Understanding intrinsic characteristics and system implications of flash memory based solid state drives. ACM SIGMETRICS Performance Evaluation Review, 37(1):181–192, 2009.
11. Manco F, Lupu C, Schmidt F, et al. My VM is Lighter (and Safer) than your Container[C]// The, Symposium. 2017:218-233.
12. Chen H, Zhang H, Dong M, et al. Efficient and Available In-Memory KV-Store with Hybrid Erasure Coding and Replication[J]. Acm Transactions on Storage, 2017, 13(3):25.