论文材料收集及思考

# 摘要

写作应该重点在混合上面，说明白NVM对于SSD的补充作用（小写，持久化性），在解决小写时，目前文件系统存在的问题，cow。HLFS。

# 引言

1. 传统基于NVM设计的文件系统在模拟的NVM上进行测试，没有考虑AEP硬件的真实特性
2. 影子页Cow方式更新数据，小写更新的写放大严重
3. NVM价格昂贵，容量优先。考虑将SSD作为后备存储，DRAM-NVM-DISK多层结构，有望发挥NVM的低延迟特性，同时兼具SSD的大容量。
   1. 根据数据操作模式读/写，数据/元数据，将数据以不同的方式写入NVM/SSD
   2. 利用设备之间的并发带宽，提高整体吞吐。比如根据当先AEP的最近吞吐情况，将一些流量强行导到SSD
4. 崩溃一致性
5. 精心设计数据管理布局，减少NVM的顺序和持久性开销
6. 如何充分利用NVM的字节寻址特性，解决小写的写放大问题
7. 提供数据的强一致性保证，同时最小化NVM的顺序开销。
   1. Log的方式
8. 充分考虑AEP的特性。线程扩展性，256B
   1. Ntstore
9. 混合介质，需要充分考虑不同介质的特性，将不同的数据导入不同的介质中，比如使用NVM缓冲小的随机写，并整合成大块的批量写到SSD，**发挥不同介质的并行带宽**。减少SSD的写入量，提高SSD的寿命

思考的问题：

1. inode结构体需要对齐。考虑将元数据分离，比如inode是两百多个字节，**可以把一些常用的部分分离出来**，按照cacheline对齐，不常用的部分单独存放。
2. 文件空间预分配，提高append的性能（或许可以考虑segment和page相结合的空间管理方式，分别对应原始log和threaded log），后台回收预分配的空间
3. 抽象出一个单独的NAT层，是否可行
4. Cache的方法，LRU还是LIRS
5. SSD的log并发写入问题。可以进行实现一个多级调度队列，有限调度前台IO。而不是固定将SSD的并行性通过一个前台一个后台来实现，有时后台可能并不活跃。
6. 支持小写时，如何构建内存中的文件数据索引
   1. 小写的必要性，参考AFCM论文的负载测试
   2. 暂定slab+log的方式分配小写的空间（page NAT表），现在的问题是如何存储索引。Slab所在的page的头部，存储每个单元所在的inode number和offset（如果实现NAT的话，只用存page num）。这样会造成大量的小随机写，是否可以通过完全写log的方式，可以测试对比一样？
   3. 内存可以用调表，但是如何存储到NVM呢？
7. 参考nova。争取每个操作只要写一次NVM log（log中加数据的指针），减少flush+fence次数
8. 是否需要实现内存的page cache

把inode map全部放内存，修改映射时，通过写log的方式，介绍随机小写，可以考虑这种使用方法，传统文件使用journal更多只是处于崩溃一致性的考虑，但也有吸收小的随机写，并通过checkpoint来batch更新

待细看的论文：

Log-Structured Non-Volatile Main Memory

1. A Hybrid Approach for Managing Data Pages in Persistent Memory File Systems
2. HasFS
3. TridentFS:（更好的淘汰算法，包括元数据、更好的设备并行性算法、由于普通的RAID-0）

## AEP有哪些独特的特性

仅仅考虑带宽、延迟、访问粒度和访问模式，并不能充分发挥AEP的潜能，还需要其他一些AEP的特性，如线程扩展性，同一地址的重复刷写等。

已有的工作都是基于模拟的NVM来进行实验，没有切确考虑真实NVM硬件的特性。

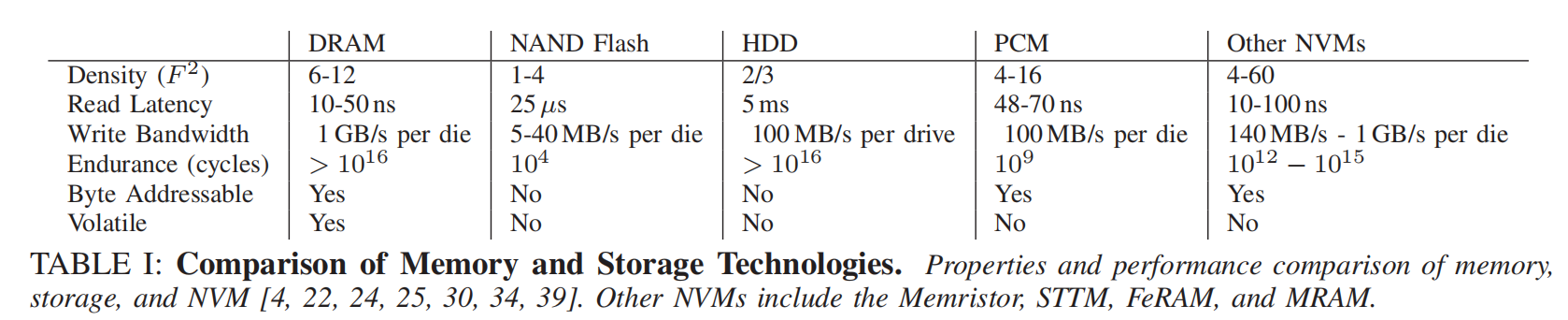
和背景知识中的AEP部分有重叠

这里没有关注的特性，可以附上测试图：1）线程扩展性；2）随机小写 vs 顺序写；3）同一地址重复刷写的高延迟问题（一个slab分配器的论文有描述）

## 现有NVM-Aware文件系统小数据的写放大问题

## NVM和SSD混合的优势和挑战

现有AEP的使用方式，是和传统SSD、HDD并列的方式，具体如何充分发挥它们的性能交给了用户来决定，用户需要花费更多的精力去决策。设计一个混合介质的文件系统，可以节约用户的使用成本。



来自 A Study of Application Performance with Non-Volatile Main Memory

分层：DRAM、NVM、disk，使得需要考虑三方面数据的划分：

* 元数据（大部分设计将元数据集中存放在NVM当中）
* 新写入的数据
* 读取的数据

另外需要考虑到AEP带宽并不高，与先进的SSD相比，带宽并没有很明显的优势，并且具有很明显的写线程扩展问题。考虑利用设备之间的并发带宽，提高整体吞吐。比如根据当先AEP的最近吞吐情况，将一些流量强行导到SSD。

## 现有NVM-aware文件系统的问题

**现有的文件系统专门用于一种存储技术**。

现有文件系统做出适合于特定类型存储设备的权衡；没有一个文件系统适合于不同的存储介质。例如，NOVA [52]和PMFS [25]需要字节可寻址性，并将它们限制为NVM；F2FS [32]使用多头日志记录和缓冲区缓存，这在NVM上是不必要的。Strata是为了利用每个存储设备的优势和弥补弱点。相比之下，在不同的媒体上分层独立的文件系统会不必要的重复机制，如块和inode分配，并且缺乏表达层之间的api。例如，块的使用频率和碎片信息不容易在独立的文件系统中传递（5.3）。[3:2]

**文件系统写入放大**。[3:3]

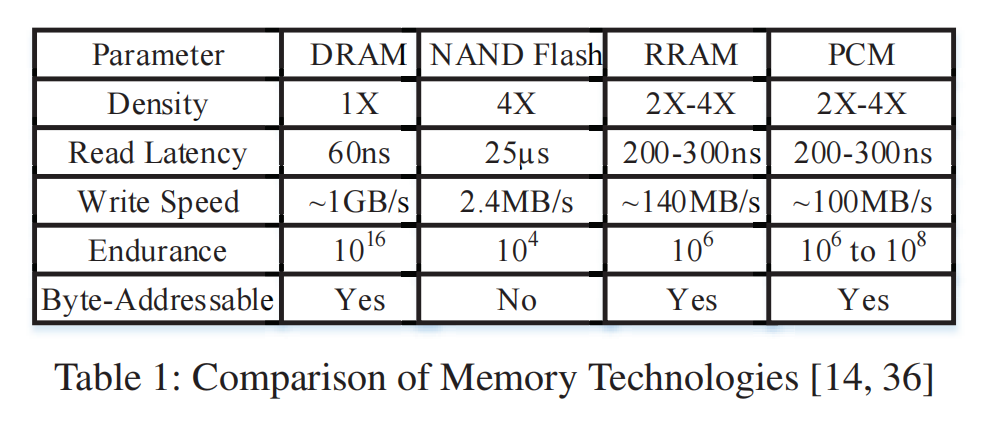
如5所示，许多文件系统将更新到统一的块大小（例如，在正在使用的块位图中设置一个位将写入整个块），并且文件系统通常需要进行元数据写入来完成更新（例如，数据写入需要更新inode中的文件大小）。与设备级写放大一样，文件系统写放大通常是影响应用程序性能的一个主要因素，特别是对于支持高效小写的NVM设备。使用NVM层的操作日志，然后进行块更新，Strata能够有效地聚合重复数据和元数据更新，显著降低文件系统的写入放大。

（log的方式更新元数据，可以利用顺序写的性能，缺点是写放大；原地更改的方式，引入小于256B的随机写，但可以减少写放大，减轻对NVM的磨损）

Q: 如何保证崩溃一致性，先将数据块链如，再修改大小，两个操作不能同时完成。通过log容易保证

各种介质的性能、容量和价格对比表。

* 上亚马逊查看价格
* 上intel官网查看性能延迟，容量
* SSD看三星的就行
* 还有DRAM



## 崩溃一致性

我们介绍了Strata，一个跨媒体文件系统，它利用一个存储媒体的优势来弥补另一个存储媒体的弱点。

和NOVA一样，具备崩溃一致性。

参考背景知识

# 背景

## 多层存储体系结构

近年来，NAND闪存帮助降低了存储和内存[30]之间历史上的高性能差距。随着存储速度的加快，其趋势是使它更靠近CPU[1]。例如，非易失性 DIMM (NVDIMM) 将存储直接连接到可扩展内存 (DDR) 接口[15、38]。 NVDIMM 因其在高额定 IOPS 下提供低延迟可预测性能的能力而越来越受欢迎 [29, 38] 但是，尽管连接到 CPU，由于 NAND [36] 的固有属性，基于 NAND 的大容量 NVDIMM 仍然作为块设备 [38] 在一个单独的地址空间。

从**容量、性能和价格**三方面着重介绍当前的DRAM-NVM-SSD混合体系结构。新兴的NVM虽然直接连接到CPU，但其访问性能和DRAM仍有差距，因为将NVM加入到传统的存储体系结构中，形成DRAM-NVM-SSD/Disk混合体系结构是一个合理的使用方式。

总结一个表，参考上面的表

## AEP & SSD

介绍一下AEP如何使用，clfush，fence，说明挑战点。

顺序和崩溃一致性。

参考 File System for Non-volatile Main Memories Performance Testing and Analysis 序言

AEP本身的特性，传统文件系统都是针对模拟NVM上进行的测试，或者没有关注AEP的特性：An Empirical Evaluation of NVM-Aware File Systems on Intel Optane DC Persistent Memory Modules

AEP的配置流程，参考：Persistent Memory I/O Primitives

参考markdown，aep和ssd

## 用户态和内核态

但是, 其缺陷是软件层次开销大, 无法充分利用非易失内存的优势. 具体的原因包括以下几个方面：[2]

* 操作系统开销：由于传统外存的延迟极高, 带宽有限, VFS 管理了一部分内存空间, 形成位于文件系统之上的缓存系统, 其将频繁访问的文件数据块、元数据块等直接放入内存, 以提升性能. 然而, 非易失内存的访问性能与 DRAM 非常接近, 因此, 在基于非易失内存的文件系统中, DRAM 缓存将不再高效. 同时, VFS 本身软件逻辑复杂, 软件开销高.系统调用开销。
* 非易失内存字节可寻址特性无法充分发挥。按照块来读写

VFS 除了管理数据页缓存, 还会将频繁访问的元数据放在 DRAM, 以提升在传统外存设备下的访问性能. 然而, 中科院计算所的研究人员发现, 基于持久性内存的文件系统在 VFS 层次的执行时间占比超过 50%, 而这些时间主要花费在元数据的路径查询上. 基于上述发现, 他们设计了 byVFS 文件系统 [58], 并提出元数据直接访问模式: VFS 不再缓存文件系统元数据, 而是直接在物理的文件系统镜像上进行元数据索引与更新. 通过这种方法, byVFS 能够将路径查询性能提升 7.1% 到 47.9%, 应用程序执行时间缩短 26.9%.[2]

## 崩溃一致性和性能之间的权衡

许多现代应用程序都需要实现其文件的崩溃一致性。在过去的十年中，实现文件用户级崩溃一致性的性能成本和复杂性有所增长，但没有看到任何缓解的迹象。[3]

在许多文件系统上，这些应用程序的高效崩溃一致性是困难的，而且速度很慢，因此许多应用程序为性能[4]牺牲了正确性。

同步持久性为应用程序[38]提供了清晰的语义（例如，排序保证和崩溃恢复），但在存储设备速度较慢的假设下，它并没有被广泛使用。现代NVM存储技术允许Strata在不牺牲性能的情况下提供同步IO语义。事实上，同步语义可以加速NVM的整体IO性能。Strata会将数据写入NVM一次，而不会将其复制到DRAM缓冲区缓存中。内存复制延迟与NVM写成本相当，因此消除内存复制大约有一半的写延迟。[3]

几十年来，计算机系统一直面临着易失性和非易失性存储之间的权衡。所有持久数据最终必须存储在非易失性介质上，如磁盘或flash，但由于这些设备只支持慢的、批量数据传输，持久数据必须临时缓冲在快速的、可字节寻址的DRAM中。不幸的是，仅驻留在易失性内存中的数据可能在崩溃或电源故障时丢失，这意味着现有存储系统在平衡这两种存储介质的使用时经常牺牲耐用性、一致性或性能[5]

BPFS利用了新的体系结构特性来提供强大的文件系统正确性保证，同时仍然受益于内存总线上的PCM的可用性。我们相信，这些架构特性的好处并不是特定于这个文件系统的——事实上，任何希望提供基于非易失性内存的保证的应用程序都需要这些特性。

## 写放大

许多文件系统将更新对齐到统一的块大小。

与设备级写放大一样，文件系统写放大通常是影响应用程序性能的一个主要因素，特别是对于支持高效小写的NVM设备。在NVM层使用操作日志，然后将其消化为块更新，Strata能够有效地聚合重复数据和元数据更新，显著降低文件系统的写入放大。

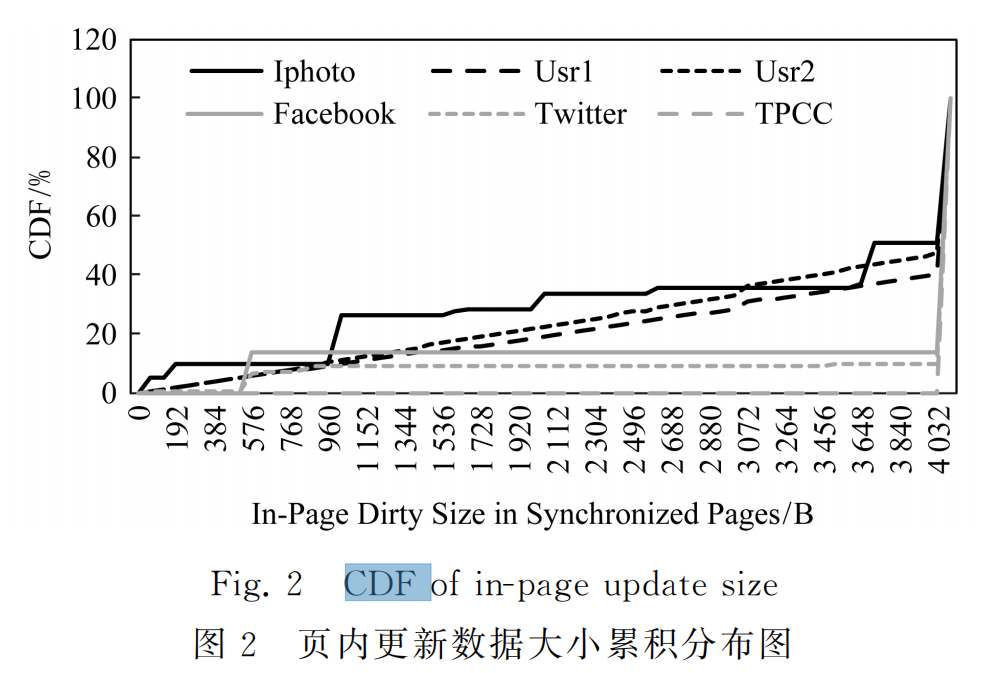
现有的文件系统并没有充分利用NVM的可字节寻址，对于数据的管理任然按照Page为单位，为了解决原子性问题，按页进行copy-on-write。对于小写任然不友好。

block 存储并不是唯一的答案。

# 挑战/动机

## 小负载情况

* 小写的必要性，参考AFCM论文的负载测试
* A Hybrid Approach for Managing Data Pages in Persistent Memory File Systems的动机中，也测试了存在大量未对齐的io负载



# 设计

## 数据流

按照数据流，总结不同负载情况下的IO方式

## Page小写问题

参考AFCM，标记flush page的脏cacheline的个数，脏的个数小时，细粒度管理，大时粗粒度管理。

细粒度：1）log的方式，只需要一次访盘，内存构建索引，每个页有两个指针，一个指向完整的数据页，另一个指向log；2）只log元数据，数据任然按照page-based分配，但支持部分写

粗粒度：Cow

内存索引——两层索引结构：先按照文件的范围划分区域，每个区域内用跳表。（可以额外考虑一下cacheline的优化）

如果索引很复杂，需不需要做thread-local的cache？类似论文LSNVMM

A Hybrid Approach for Managing Data Pages in Persistent Memory File Systems文章中的方法，但是它：1）没有将log entry cache对齐、没有避免同一个cacheline的重复刷写问题、读慢，每个page一个log，空间浪费且引入大量随机访问。实际上减少随机访问和减少垃圾回收的开销是一个trade-off。

初步的想法是，一个线程一个log，但需要想一个好的垃圾回收方案。

思考，如果实现NAT，page在实现单独的log时，如何保证崩溃一致性呢？

细粒度的另一个好处是，（1是减少对AEP的写放大）对于处于SSD中的页，避免了读改写操作，大大降低写入时延。

HDPM中，回收log时仍需要Cow，可以原地写原来的page。

另外HDPM没有解决多page写的崩溃一致性问题。

小粒度方面努力的相关论文：

- noseFS

- HDPM

## Log技术

- 经典：内容+标记（数据还显示，对于超过512次的访问，非时间存储比存储+ clwb的延迟更低 B. 非时间存储对于超过256次的访问也有最高的带宽）两个flush+sfence

- header：内容+log-header，两个flush+sfence。缺点是重复cacheline的flush，优点是恢复时不需要扫描log

- Zero/校验和：日志初始化为0，popcnt，类似校验和，和数据一起刷下去，只需要一次flush+fence（**考虑将长度和校验和放在同一个8字节当中，这样就不会出现错误了**）

- 经典优化方案：限定log entry为一个cacheline大小，写内容+sfence+标记+flush+sfence，减少一次flush。(什么时候将flush分解为步骤是有意义？例如，如果一个程序多次调用 memcpy()，它可以复制数据并只执行flush，将最后的drain步骤推迟到最后。这是有效的，因为与刷新步骤不同，drain步骤不采用地址范围 - 它是系统范围的drain操作，因此可以在复制单个数据块的循环结束时发生。所以fence对系统的性能影响应该比较大)

对于size字段重复flush的问题，这个问题可以通过使用一个舞蹈大小字段来解决：我们在头中的不同的缓存行上使用几个大小字段，并且为每个日志条目只写一个（循环）。(几个cacheline最好呢？需要测试)

考虑log+不同的write方式相结合

经典方法需要两次flush+fence，校验和方法需要额外计算，测试他们数据大小的临界值

# 测试

### 背景测试

<https://github.com/NVSL/OptaneStudy>

1. 测试AEP和SSD随机写的粒度，多大时和顺序写带块相同，以确定log的大小，而不是固定的page。
   1. 测试第二种log方式，barrier()和fence的性能差距（由于前者不发出额外的指令，因此速度应该要快一些）
2. 测试ntstore、store+clwb、每64Bstore+clwb（多线程的时候更应如此）的使用临界条件（带宽/延迟）
   1. 数据还显示，对于超过512次的访问，非时间存储比存储+ clwb的延迟更低 B. 非时间存储对于超过256次的访问也有最高的带宽 B.
3. 最初的NOVA设计有两个特点，这降低了Optane的性能。首先，每个元数据更新附加的日志条目NOVA都很小-40-64b，而且由于NOVA使用了许多日志，**日志更新显示出很少的局部性**，特别是当文件系统处于负载状态时。其次，NOVA使用在写入时复制到4 kB的页面来进行文件数据更新，从而导致无用的存储。无论底层内存技术如何，这种低效率都会发生，但Optane糟糕的存储性能加剧了其影响。（optance下，更应该减少写放大）
4. AEP和SSD的多线程访问的性能
5. 测试AEP mmap的区域能否作为spdk的 read write内存缓冲区

我们通过增加日志条目的大小和避免一些写时复制操作来解决这两个问题。我们修改后的NOVA-nova数据日志将子页面的写入数据嵌入到日志中（图10）。与日志的普通写条目不同，它包含一个指向新的4 kB页面的指针及其在文件中的偏移量，嵌入写条目包含一个页面偏移量、页面中的地址，后面是写入的实际内容。这种优化需要对最初的NOVA设计进行几个辅助性的修改。特别是，NOVA必须在进行内存映射或读取文件之前将子页面更新合并到目标页面中。（那读就比较慢，支持小数据更新后，需要测试对读的影响程度）【aep的测试时，2020fast】

# 参考文献

1. PMFS
2. 非易失主存的系统软件研究进展
3. strata
4. Thanumalayan Sankaranarayana Pillai, Vijay Chidambaram, Ramnatthan Alagappan, Samer Al-Kiswany, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. 2014. All File Systems Are Not Created Equal: On the Complexity of Crafting Crash-consistent Applications.
5. BPFS
6. NOVA
7. SCMFS
8. A Study of Application Performance with Non-Volatile Main Memory
9. Ziggurat
10. An Empirical Study of File Systems on NVM
11. F2FS
12. Avantika Mathur, Mingming Cao, Suparna Bhattacharya, Andreas Dilger, Alex Tomas, and Laurent Vivier. 2007. The new ext4 filesystem: current status and future plans. In Proceedings of the Linux Symposium, Vol. 2. Ottawa, ON, Canada.
13. NVMFS
14. The Design and Implementation of a Log-Structured File System
15. kucofs ↩︎