# Efficient and Consistent NVMM Cache for SSD-Based File System

基于SSD的文件系统的高效和一致的NVMM缓存

## 摘要

**缓冲区缓存是提高系统性能和延长SSD寿命的有效方法。**然而，大多数实际应用中的频繁同步操作限制了这种优势。本文提出采用新兴的非易失性主存储器（NVMMs）来缓解上述问题，同时实现高效和一致的缓存管理。为此，提出了一个自适应细粒度缓存（AFCM）方案，其动机是我们观察到许多同步页中的文件数据对于广泛的工作负载来说是**部分更新**的，这意味着细粒度缓存管理可以节省未更新部分浪费的NVMM缓存空间。为了减少细粒度缓存管理引入的缓存**索引开销**，AFCM采用了基于DRAM和NVMM的混合缓存，用它来提供正常的读写操作而不影响性能。我们还提出了**事务性的写时复制机制**，以保证NVMM缓存空间和文件系统镜像的崩溃一致性。我们的实验结果表明，与传统的粗粒度缓存管理方案相比，AFCM平均提供了高达84%的性能改进和63%的SSD写入量。

## INTRODUCTION

基于FLASH的固态硬盘（SSD）是电子存储设备。它们不包含任何机械部件，因此在低访问延迟、低功耗、紧凑尺寸和抗震方面具有很大的优势。然而，它们面临着写入性能相对较差和寿命有限的问题。以前的研究[1], [2], [3]表明，过多的写操作不仅会显著降低基于SSD的系统性能，而且还会大大减少SSD设备的寿命。为了缓解这个问题，缓冲区缓存是一种有效和常用的方法，可以减少SSD的写入流量[4], [5], [6], [7], [8], [9], [10], [11]，从而提高系统性能，延长SSD的寿命。

传统的文件系统通常使用DRAM作为缓存层，以提高系统性能，因为它具有高耐久性和低访问延迟。不幸的是，DRAM的易失性在系统崩溃时可能导致数据丢失。因此，应用程序要么采用基于日志的文件系统，要么发布额外的同步操作（例如fsync或fdatasync），以使文件数据持久化[12], [13]。例如，数据库系统会在提交事务之前同步日志文件[14]。虽然上述方法提高了系统的可靠性，但由于频繁访问存储空间，它们大大降低了DRAM缓存的有效性[15], [16]。（实际上DRAM缓存只对读有效，写没有太多的提升，每次写入操作为了保证数据的持久性，必须访问SSD）

[16] E. Lee, H. Kang, H. Bahn, and K. G. Shin, “Eliminating periodic

flflush overhead of fifile I/O with non-volatile buffer cache,” IEEE

Trans. Comput., vol. 65, no. 4, pp. 1145–1157, Apr. 2016.

随着新兴非易失性存储器（NVM）技术的快速发展，如3D-XPoint、相变存储器（PCM）、电阻式RAM（ReRAM）和记忆体，它们有望提供类似DRAM的性能和比闪存更好的耐久性。将NVM直接连接到主存储器总线上将产生可由字节寻址的非易失性主存储器（NVMM）。与DRAM相比，NVMM具有非易失性、高密度和低功耗的独特特性[17], [18]。因此，使用NVMM作为持久性缓存来提高SSD的性能和耐用性是切实有效的[7], [8], [9], [10], [11], [19], [20] 。

相变存储器件：

[17] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, “Architecting phase

change memory as a scalable DRAM alternative,” in Proc. 36th

Annu. Int. Symp. Comput. Archit., 2009, pp. 2–13.

[18] P. Zhou, B. Zhao, J. Yang, and Y. Zhang, “A durable and energy

effificient main memory using phase change memory technology,”

in Proc. 36th Annu. Int. Symp. Comput. Archit., 2009, pp. 14–23.

[19] E. Lee, H. Bahn, and S. H. Noh, “Unioning of the buffer cache and

journaling layers with non-volatile memory,” in Proc. 11th USE

NIX Conf. File Storage Technol., 2013, pp. 73–80.

[20] Q. Wei, C. Wang, C. Chen, Y. Yang, J. Yang, and M. Xue,

“Transactional NVM cache with high performance and crash con

sistency,” in Proc. Int. Conf. High Perform. Comput. Netw. Storage

Anal., 2017, Art. no. 56.

由于目前的NVMM设备价格高、**容量小**，为SSD设计NVMM缓存的关键挑战之一是如何提高缓存利用率。然而，现有的基于NVMM的固态硬盘缓存策略是以闪存页粒度[7]、闪存块粒度[8]、[9]、[10]或混合**页和块粒度**[11]管理NVMM缓存空间。这些算法主要集中在利用空间局部性（locality）来提高回写操作的顺序性（顺序回写）。然而，由于粗粒度的管理，**它们受到缓冲区污染（浪费）和过早驱逐问题的影响**，导致缓冲区利用率低和命中率低，从而降低了系统性能。

在本文中，我们观察到大量的工作负载包含大量的同步操作（见2.2节）。为了消除这些同步操作的影响，我们建议引入NVMM缓存来缓冲这些操作，以减少对SSD的写入流量。为了提高缓存的利用率，我们进一步为NVMM缓存提出了一个自适应细粒度缓存管理（AFCM）策略。AFCM不是对整个同步页面进行缓冲，而是有选择地对同步页面的脏部分进行缓冲，以节省被干净页面浪费的NVMM空间。通过充分利用NVMM的字节寻址能力，AFCM提高了缓存利用率和系统性能。据我们所知，这是第一个以细粒度管理NVMM中的文件数据的工作。然而，细粒度的缓存管理有三个关键的挑战：

1）在细粒度管理的情况下，**索引缓存数据的开销会变大**，由于缓存命中/缺失判断的延迟增加，这可能会导致读写性能低。

2）细粒度的缓存管理将**在NVMM中引入额外的元数据**，这使得系统能够在系统崩溃后的恢复过程中跟踪数据（在第3.3节）。然而，当同步页面的脏污率很高时（例如，接近100%），上述方法可能会降低缓存的利用率。

3）利用NVMM的字节寻址能力，细粒度的缓存管理使得保证NVMM缓存的一致性更加复杂。这是因为文件页在NVMM缓存中被分割和分散。同步页的任何部分更新都会在系统崩溃时造成文件系统的不一致。

因此，我们做出了以下贡献：

1）通过对不同文件系统工作负载的分析，我们发现：（1）需要明确同步到稳定存储的写操作占总写操作的很大一部分；（2）**大多数同步页中脏缓存行的平均百分比远低于100%**。

2）为了充分利用NVMM缓存空间，我们提出了一个自适应细粒度缓存管理方案，该方案根据脏缓存行的比例动态地选择细粒度或粗粒度的缓存模式来缓冲同步页面。

3）为了减轻细粒度管理造成的索引开销，通过结合DRAM和NVMM，引入了一种新型的混合缓存。DRAM缓存以页为单位进行管理，为正常的读写操作提供服务，而NVMM缓存仅用于缓存同步数据以保证一致性。我们进一步提出了事务性的 **"写时复制"（TCoW）机制**。它引入了一个循环日志来有效地保持NVMM缓存和文件系统镜像的一致性。

4）我们的实验结果表明，AFCM比传统的粗粒度缓存管理策略要好84%。同时，与传统方法相比，它还实现了平均63%的SSD写入量，从而大大延长了SSD的使用寿命。

本文的其余部分组织如下。第2节介绍了一些背景并讨论了动机。我们在第3节和第4节分别介绍了AFCM的设计和实现。然后我们在第5节中介绍了AFCM的评估结果。最后，我们在第6节中讨论了相关工作，并在第7节中得出结论。

## BACKGROUND AND MOTIVATION

在本节中，我们首先分析了文件系统中现有的缓存策略。然后，我们收集了不同工作负载的fsync字节的百分比，以揭示同步的开销。我们还分析了同步页面中脏缓存线百分比的分布，以显示AFCM的潜在优势。本节还说明了文件系统中崩溃一致性的开销，这促使我们设计TCoW。

### 文件系统中的缓存管理

现有的计算机系统配备了不同的存储设备，如寄存器、DRAM、闪存和硬盘。它们在访问延迟和带宽方面都表现出不同的特性。因此，缓冲区缓存是一种广泛使用的方法来填补不同存储层之间的性能差距。缓存效率是由平均缓存访问时间（表示为T）决定的。它取决于缓存命中访问时间（表示为Thit），缓存命中率（表示为Phit），以及缓存错过访问时间（表示为Tmiss）。因此，T的值可以用以下公式表示。

T = Thit \* Phit + Tmiss \* (1 – Phit)

为了提高缓存效率，系统设计者可以对这三个因素中的每一个进行优化。

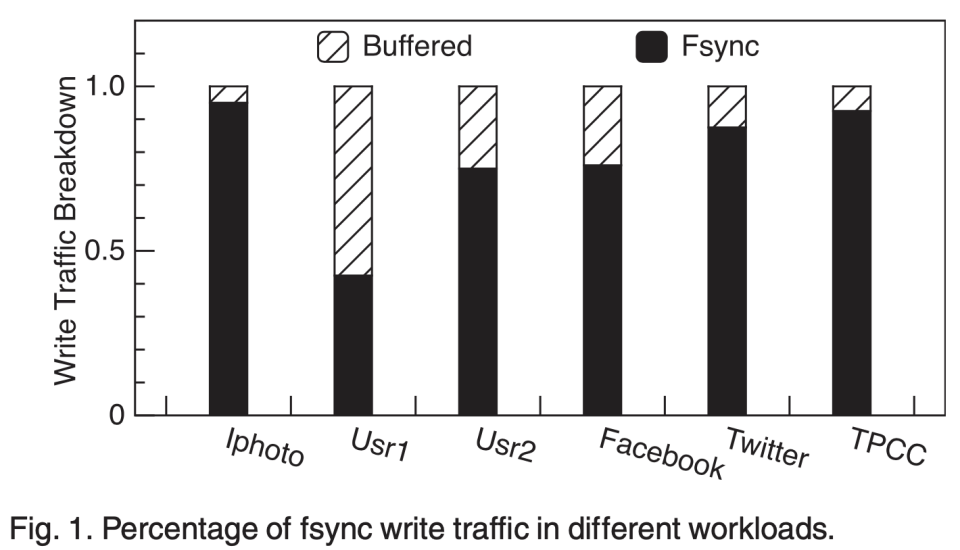
由于目前NVMM技术的高价格和低容量，现有的缓存优化主要集中在从两个方面设计基于NVMM的缓存策略。一个是提高缓存命中率（Phit）。由于Thit的值远低于Tmiss的值，较高的缓存命中率可以使T的值降低，表明缓存效率较高。同时，随着缓存命中率的增加，缓存的回写流量也会减少，从而导致SSD设备的寿命延长。现有的方法[21], [22], [23], [24]利用时间局部性和访问频率来设计有效的缓存替换算法，**以便尽可能长时间地将最有可能被再次访问的数据保留在缓存中**。另一方面，为了减少缓存缺失惩罚（即Tmiss）。一些研究人员考虑到SSD的高顺序写入性能，设计了高效的缓存回写算法[4], [8], [9]。这些算法主要是利用空间局部性和DRAM/NVMM缓存中的常驻页，以便尽可能按顺序将脏的缓存数据冲到SSD设备上。然而，由于粗粒度的管理，它们受到缓冲区污染和早期驱逐问题的影响，导致低缓存利用率和低命中率。

值得注意的是，上述两种方法的有效性取决于工作负载的访问特性，因为它们都利用了访问的位置性或频率来进行优化。因此，现有的缓存管理方案的能力是有限的，因为大多数I/O密集型工作负载是随机和顺序I/O请求混合在一起的。为了不受工作负载特征的影响，我们从另一个独特的角度（即缓存管理粒度）来优化基于NVMM的缓存效率。由于高速缓存命中率不仅受高速缓存替换算法的影响，而且还由总的可用高速缓存空间决定。**提高缓存的利用率会带来更多的可用缓存空间，从而提供更高的缓存命中率和系统性能**。

### Workload Analysis

以前的工作[13]表明，在大多数应用中，很大一部分写入的数据应该明确地同步到磁盘上，以确保其所要求的数据持久性的语义。这将导致较低的缓存命中率（第2.1节），即使缓存空间没有用完。因此，同步操作的频率和同步的数据量会直接影响系统的性能和DRAM缓存的效率。

为了了解同步操作对DRAM缓存效率和相关系统性能的影响，我们做了一个实验来跟踪各种工作负载中同步操作的使用情况。图1显示了这些工作负载中Fsync Write和Buffered Write的写流量细分。我们可以观察到，在所选的6个工作负载中，有5个工作负载的总写入数据的70%以上需要明确同步到存储层。此外，其余的工作负载（即Usr1）也有近50%的总写入数据需要被持久化。总而言之，大量的应用都有相当一部分写入的数据有持久性要求。因此，采用NVMM作为持久性缓存工作既是必要的也是有效的。



为了了解细粒度缓存管理的潜在性能优势，我们进一步分析了所有同步页面的页内脏污率的分布。页内脏污率是指一个页面的脏污数据大小占总页面大小的比例。结果显示在图2中。从图中，我们发现在不同的工作负载中，同步化页面的脏污率分布有很大的不同。例如，在TPCC工作负载中，超过99%的同步页面具有100%的脏污率（即这些同步页面中的所有数据都是脏污）。与此相反，Iphoto、Usr1和Usr2工作负载有很大比例的总同步页面，其脏污率远远低于100%（即这些同步页面中只有部分数据是脏的）。这表明具有粗粒度管理（即页粒度）的传统缓存方案往往会使NVMM的缓存空间被部分更新的同步页所浪费，从而降低了缓存空间的利用率。这一观察促使我们设计了一种基于NVMM的新型细粒度缓存管理方案。它通过减少部分更新页面造成的缓存空间浪费来实现这一目标，从而提高缓存利用率和系统性能。

PS：没有考虑整页的情况，考虑到现在很多先进的SSD，吞吐量和NVM相当，4KB的整页写可以直接写到SSD，而无需NVM缓冲。这样也能减少NVM的磨损。提高NVM用于缓冲小写的的空间

### 文件系统中崩溃的一致性开销

文件系统的崩溃一致性保证了上层应用程序的数据完整性。现有的文件系统[25], [26], [27], [28]已经开发了不同的技术来提供崩溃一致性。

M. Rosenblum and J. K. Ousterhout, “The design and implemen

tation of a log-structured fifile system,” ACM Trans. Comput. Syst.,

vol. 10, no. 1, pp. 26–52, Feb. 1992.

C. Lee, D. Sim, J. Y. Hwang, and S. Cho, “F2FS: A new fifile system

for flflash storage,” in Proc. 13th USENIX Conf. File Storage Technol.,

2015, pp. 273–286.

最流行的持续更新文件数据的解决方案是日志（又称 " Write-Ahead-Logging"）[29], [30], [31] 。在将文件数据更新到存储设备上的地方之前，相应的日志条目首先会在日志区中被持久化。然而，它给磁盘引入了双重更新，从而限制了系统性能。另一种方法是Copy-on-Write（CoW）[26], [32]，它被许多文件系统采用，如Btrfs和ZFS[33]。它不是在原地覆盖文件，而是将更新的数据重定向到新的位置，并通过更新文件元数据的指针到新的位置来验证它。这种技术避免了数据的重复写入，然而，任何文件数据的更新都会传播到根节点，导致次优的性能。

[29] V. Chidambaram, T. S. Pillai, A. C. Arpaci-Dusseau, and R. H.

Arpaci-Dusseau, “Optimistic crash consistency,” in Proc. 24th

ACM Symp. Operating Syst. Principles, 2013, pp. 228–243.

[30] R. Hagmann,“Reimplementing the Cedar fifile system using log

ging and group commit,” ACM SIGOPS Operating Syst. Rev.,

vol. 21, no. 5, pp. 155–162, 1987.

[31] A. Sweeney, D. Doucette, W. Hu, C. Anderson, M. Nishimoto, and

G. Peck, “Scalability in the XFS fifile system,” in Proc. Annu. Conf.

USENIX Annu. Tech. Conf., 1996, pp. 1–1.

还有一些基于NVMM的缓存方案[19], [20]，是为文件系统设计的崩溃一致性。UBJ[19]采用NVMM来作为日志设备和缓冲区缓存工作。为了在基于NVMM的缓冲区缓存中提交数据块，它们被就地冻结以作为日志记录，并在随后被检查点到磁盘以释放NVMM空间。然而，UBJ在更新冻结数据时需要一个额外的内存拷贝。它还依赖于传统的检查点技术来回收NVMM空间，这是很低效的。Tinca[20]通过使NVMM中的数据块在提交和检查点的角色之间切换来避免重复写入。**然而，由于Ext4中的提交和检查点机制的紧密耦合设计，这两种方法的通用性都很有限。此外，它们还存在磨损问题，因为所有的文件I/O都会流经NVMM的缓存**。

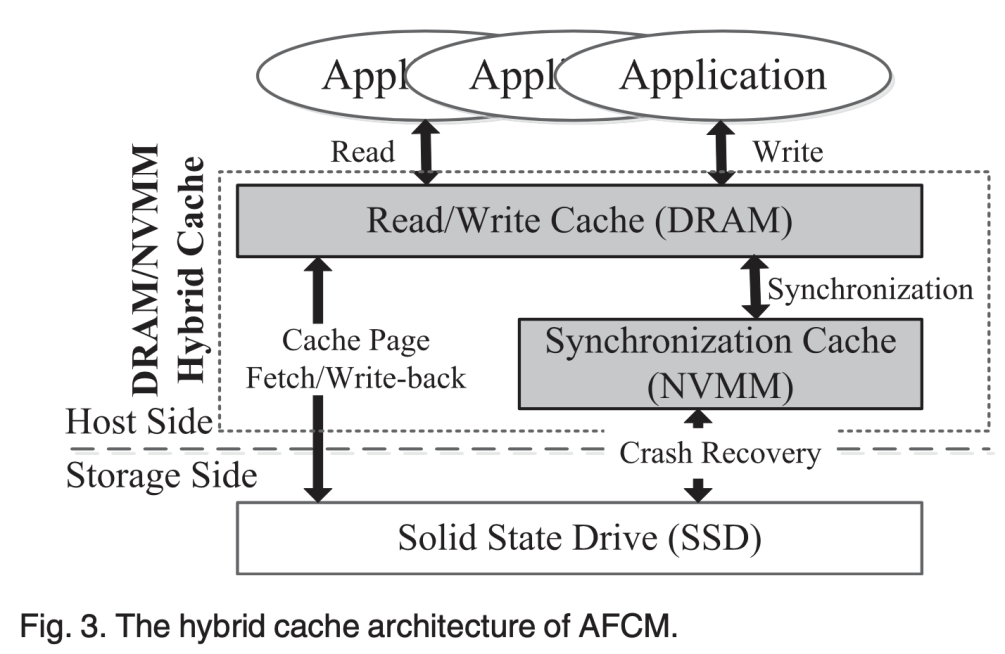
总的来说，现有的文件系统中的崩溃一致性方法要么产生高的开销，要么具有有限的通用性。在本文中，我们的目标是通过提供原子更新接口来设计高效和一致的缓存方案。这使得下面的文件系统自然而然地继承了高一致性的特性，不管它们的实现如何。

## AFCM DESIGN

在本节中，我们将首先讨论如何设计一个基于NVMM的高效的细粒度缓存管理方案，以提高系统性能并减少对SSD的回写流量。然后，我们介绍了事务性的写时拷贝机制的设计，该机制通过提供原子性的同步化操作来保证系统的一致性。

### Hybrid Cache Architecture

虽然细粒度的缓存管理策略可以提高缓存的利用率（在第2.2节中提到），**但它也增加了缓存索引的开销，导致读写性能降低**。换句话说，缓存颗粒度越细，索引缓存数据的开销就越大。为了克服这个问题，**我们提出了一个基于DRAM和NVMM的混合高速缓存**。这个架构的设计如图3所示。



SSD的缓存层包括一个易失性缓存（即DRAM）和一个非易失性缓存（即NVMM），它们都在主机侧。NVMM与DRAM一起直接连接到内存总线上，并通过内存接口（即加载/存储指令）进行访问，以便支持字节寻址。**此外，NVMM缓存空间通过细粒度的方式管理，利用其非挥发性缓冲同步sync操作。**为了避免细粒度缓存管理对正常读写操作性能的影响，额外的DRAM缓存是以页粒度管理的，以缓冲读写操作，从而使索引开销很小。

PS：实际上内存的空间要比NVM小很多，DRAM不能缓冲所有到NVM的更新，还需额外的优化。实际上这个方案和传统的写log很类似，log用来保证崩溃一致性，内存中的接口用来加快访问。

当服务于正常的读写操作时，页面获取和回写操作在DRAM缓存和后端SSD设备之间直接发出。**不同的是，NVMM高速缓存主要用于确保应用程序所需的数据持久性**。当文件系统收到同步操作时，AFCM将以细粒度的方式将需要同步的相关DRAM页冲到NVMM缓存中（详见3.2节）。这种设计消除了在传统的纯DRAM缓存环境中由同步操作引起的对SSD的写入流量，从而提高了系统性能并延长了SSD的使用寿命。此外，AFCM只允许同步数据流向NVMM设备，因此与现有的使所有文件数据流经NVMM的缓存方案[19], [20]相比，减少了NVMM的磨损。

NVMM和DRAM缓存的缓存驱逐只有在没有空闲块时才会发生。为了将DRAM缓存中的脏页替换到SSD设备上，NVMM缓存中的相关数据将被丢弃，因为这些数据已经被持久化到SSD设备上。当NVMM缓存中发生替换操作时，被替换的缓存数据在从NVMM缓存中驱逐之前也应被写回SSD设备，以保证数据的持久性。为简单起见，它的替换与DRAM缓存一样是面向页面的（在3.2.1节）。为了保留现有高速缓存替换算法对正常读写操作的好处，**NVMM高速缓存中的替换不会改变DRAM高速缓存中相关页面的优先级，但会将其状态从脏变为干净**。原因是这个DRAM页可能仍有较高的缓存优先级，即使相关的数据被从NVMM缓存中驱逐。因此，我们设计中新增加的NVMM缓存层不会改变现有的DRAM缓存策略，而是主要用于尽可能地减少同步操作引起的对SSD的写入流量。

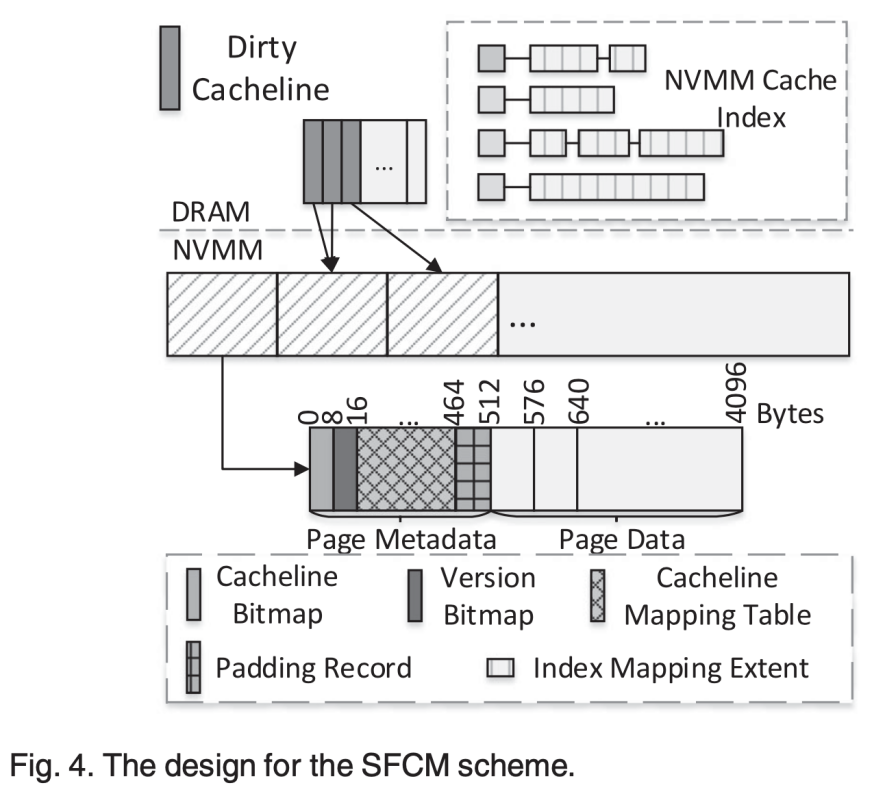
讨论。以前的工作指出，NVMM也有耐久性问题。然而，这在AFCM中不是一个问题，因为1）AFCM分离了异步/同步数据的I/O路径，所以NVMM和SSD设备之间的写流量被重新平衡；2）同一文件中的多个脏数据块将在NVMM中被合并和压缩，所以对SSD设备的写流量将大大减少；3）我们只需要在NVMM中修改数据块的脏部分，这进一步限制对NVMM的写放大。总之，与只配备SSD设备的 "单级架构 "相比，我们提出的混合架构能够减少对NVMM和SSD的写入流量。

### 自适应的细粒度缓存管理

我们将首先介绍简单的细粒度缓存方案（SFCM，第3.2.1节），用于提高缓存的利用率。在此基础上，我们提出了一个自适应缓存管理（AFCM，第3.2.2节），根据每个同步页面的脏污率动态地选择缓存模式（缓存线或页面颗粒度）。

#### Design of SFCM

SFCM方案的设计如图4所示。由于CPU和内存之间的最小交互粒度是CPU缓存线的大小（即64B），SFCM不会在NVMM中缓冲整个同步页面，而只是将脏的缓存线写到NVMM的缓存中，以提高NVMM的缓存利用率。此外，在同步操作完成后，相应的DRAM页不会被丢弃或替换。**因此，所有最新的数据仍然留在DRAM缓存中，为正常的读写操作服务。**



为了便于系统在同步化操作中判断脏缓存线是否已经位于NVMM缓存中，SFCM为每个逻辑文件页维护一个基于链接列表的索引，其头部由一个全局内存哈希表索引（见图4）。为了节省索引的存储开销并提高索引性能，链表中的每个条目代表一个连续的映射范围。每个映射范围是一个三元组，表示为<Logic\_File\_Addr, Physical\_NVMM\_Addr, Mapping\_Len>，表示从Logic\_File\_Addr到Logic\_File\_Addr + Mapping\_Len的连续逻辑文件范围被映射到NVMM缓存中从Physical\_NVMM\_Addr到Physical\_NVMM\_Addr + Mapping\_Len的连续物理范围。**由于SFCM根据缓存线的粒度将同步页写入NVMM缓存，三元组中每个项目的值都与缓存线大小一致**。为了便于搜索和合并映射范围，所有属于同一逻辑页的映射外延都按逻辑文件地址排序。对于每个需要同步到NVMM缓存的脏缓存线，SFCM首先参考NVMM缓存索引。如果是命中，SFCM直接将这个缓存线写到相应的NVMM位置（指定是索引原地修改）。否则，SFCM应从NVMM分配新的缓存空间，以缓存线为粒度，并更新相应的缓存索引。

在系统崩溃恢复过程中，为了确定NVMM中每个缓冲缓存线的相应SSD位置，SFCM需要为NVMM中每个缓冲缓存线维护一个额外的映射表。**如图4所示，SFCM将NVMM缓存空间划分为连续的页面，每个页面内的空间根据缓存线的颗粒度进行分配**。页面元数据位于每个NVMM页面的头部，由四部分组成。第一个8字节的缓存线位图用于识别页面数据区的高速缓存线是否处于有效或空闲状态。后面的版本位图用于崩溃一致性，这将在第3.3节中介绍。**缓存线映射表用于记录页面数据区中每个缓存线对应的SSD地址**（算了一下约8bytes）。填充记录用于使页面元数据区与缓存线的大小一致。

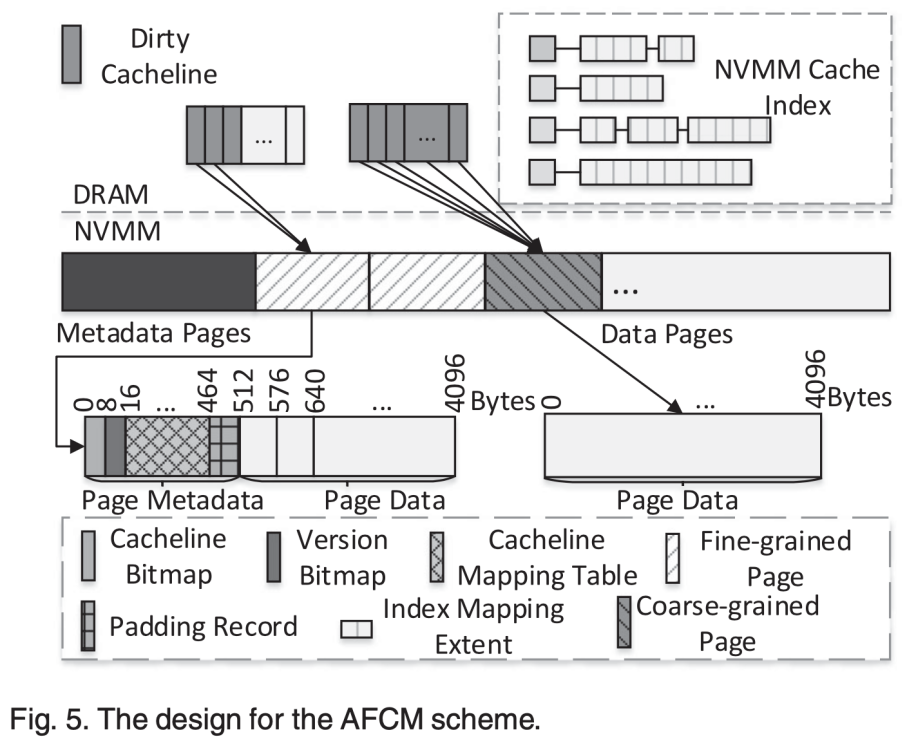
尽管NVMM缓存的空间分配是基于缓存线粒度的，但NVMM缓存的替换是基于页粒度的。这是因为被替换的数据在被驱逐之前应该被写回固态硬盘，而固态硬盘的写入单位是逻辑文件页。当一个逻辑页被选择从NVMM缓存中驱逐时，与该逻辑页相关的所有碎片数据需要被写回SSD，然后它们的空间将被回收用于未来的缓存。SFCM不需要通过合并NVMM中所有相关的脏缓存线来生成一个新的逻辑页。相反，在DRAM中维护的相关页面被选择写到SSD中。

SFCM的主要优点是它只将同步页的脏缓存线写入NVMM缓存，这就节省了由干净缓存线造成的缓存空间浪费。然而，当一个同步页的脏污率接近或等于100%时，SFCM将降低缓存的利用率。**这是因为细粒度的高速缓存管理引入了额外的页面元数据。更详细地说，每个4,096字节的NVMM页面有512字节的元数据开销。**因此，在SFCM方案下，节省的缓存空间不足以摊销引入的页面元数据开销，从而降低了缓存利用率和系统性能。

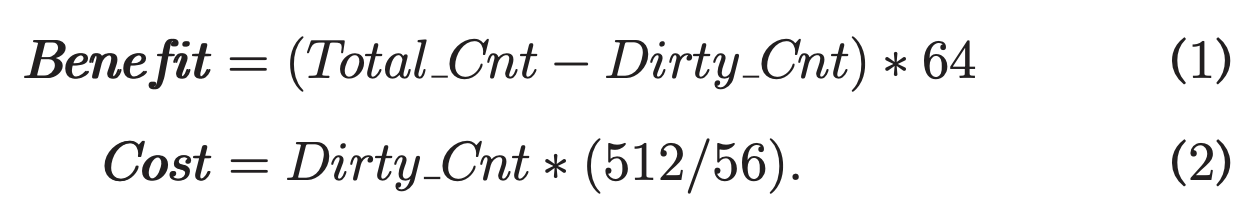
#### Design of AFCM

为了避免SFCM的缺点，我们进一步提出了一个自适应细粒度缓存管理方案。AFCM的关键思想是**根据每个同步化页面的脏污率自适应地选择细粒度或粗粒度的缓存管理模式**，从而避免了简单的细粒度方法的缺点。

AFCM方案的设计如图5所示。在AFCM中，NVMM高速缓存空间由一些元数据页和一些动态分配的数据页组成。为了满足自适应分配的要求，AFCM包含两种不同类型的数据页，分别为细粒度页和粗粒度页。细粒度页的数据结构与SFCM中的NVMM页的结构相同。相反，粗粒度的管理是基于页的粒度的。如图所示，粗粒度页不包含任何元数据，因为这些粗粒度页的映射表（即相应的SSD地址）被存储在NVMM缓存的全局元数据页中，用于崩溃恢复。此外，全局元数据页为每个数据页存储2比特的状态位图，它代表三种状态（即细粒度、粗粒度或自由状态）。粗粒度数据页的版本位图也被存储在其中。



AFCM的关键挑战是如何自适应地确定每个同步页面的缓存模式（即粗粒度模式或细粒度模式）。**为此，我们提出了一个基于细粒度缓存管理策略的成本-效益模型**。在这个模型中，我们假设一个同步页面中的所有缓存线和脏缓存线的总数分别为Total Cnt和Dirty Cnt。那么，细粒度缓存模式所节省的缓存空间为：（Total Cnt - Dirty Cnt）\* 64字节。 然而，正如3.2.1节所分析的，每个4,096字节的页面只存储了56个缓存行的数据，而其余512字节的空间被用于页面元数据，这表明每个脏缓存行平均产生512/56字节的元数据开销。因此，细粒度缓存模式的好处和成本可以分别用公式（1）和（2）表示。



上述模型只是忽略了粗粒度模式引起的元数据开销，因为它几乎可以忽略不计（小于0.2%）。此外，细粒度索引所带来的额外成本也没有被考虑在内，因为与SSD上（典型的延迟为几十微秒）减少I/O操作的好处相比，它是微不足道的。**在AFCM中，只有当收益大于成本时，它才会为同步化的页面选择细粒度模式。否则，它将使用粗粒度模式来缓冲这个同步页面**。值得注意的是，如果当前同步页面被确定为用粗粒度模式缓冲，但之前是用细粒度模式存储的，**AFCM需要首先将NVMM中所有相关的碎片数据与当前同步的DRAM页面合并。然后，它将DRAM页写入一个新分配的粗粒度NVMM页。**随后，它通过修改位于全局元数据页区域的状态位图将该NVMM页的状态从自由变为粗粒度。最后，这个逻辑页的原始碎片数据所占用的NVMM空间被释放。缓存替换只在当前同步页是粗粒度的，而将要被驱逐的页是细粒度的情况下有所不同。在AFCM中，我们选择容纳最大部分受害者页面的NVMM页面进行回收，所有与该受害者相关的逻辑文件页面也将被驱逐。因此，更多的NVMM的自由空间被回收。（所以log里面还要存储page num用于反向索引）

AFCM的NVMM缓存利用率比SFCM高得多，特别是当所有同步页面的平均脏污率接近100%时。因此，AFCM显示出比SFCM更好的系统性能。

### 系统的一致性和崩溃恢复

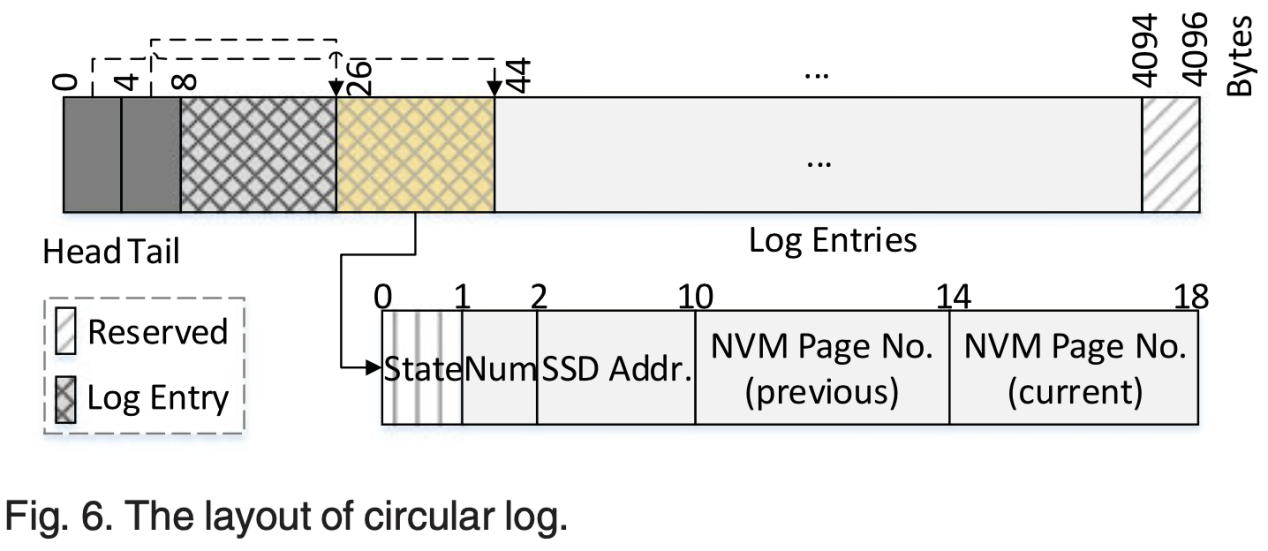
系统一致性和崩溃恢复是文件系统中最重要的部分。利用NVMM的高性能和字节可寻址性，AFCM通过事务性拷贝写机制有效地支持一致的文件数据更新。此外，AFCM通过精心组织NVMM中的元数据实现了系统崩溃后的快速恢复。

> 3术语 "原子同步操作 "被定义为。当系统在执行fsync操作的过程中崩溃时，应用程序在重启后看到的是前一个或当前版本的一致性数据。

#### 事务性的拷贝写

传统的文件系统，如第2.3节所述，要么不能保证数据的一致性（Ext2等），在系统崩溃时让文件数据部分更新，要么由于双重数据写入（Ext4 +JBD）或级联更新（Btrfs）而产生高额开销。我们通过提供原子同步化操作来实现高效和一致的文件数据更新，这些操作是基于写时复制机制的。有了NVMM缓存来接管同步数据流，AFCM可以通过仔细管理NVMM缓存空间来提供一致的文件数据更新，无论底层文件系统的实现如何。

事务性的写时拷贝机制使所有数据同步到NVMM的out-of-place更新。对于涉及多个DRAM页或细粒度缓冲区的同步化操作，其原子性由**循环日志**来保证，循环日志是一个放置在NVMM缓存空间尾部的4KB页面（如图6所示）。



例如，一个包含两个脏数据页（表示为P1和P2）的同步操作将以下列步骤处理。

1) 初始化Head指向的日志条目，其中Num, SSD Addr.和NVM Page No.(previous)代表了P1中脏缓存线的数量，存储该文件页的相关SSD地址，以及缓存P1的前一个NVM页索引（如果在NVMM中没有命中或者之前用细粒度模式缓存的，则为空）。

2) 以循环方式将Head向前移动一个。

3) 如果P1先前在NVMM中被缓存，将P1的版本位图（见第3节）更新为旧的，并将State更新为1（State将被检查以确定在恢复期间哪个阶段已经完成）。

4) 根据成本效益模型为P1分配新的空间，更新NVM Page No（current）以指向新分配的空间（当以缓存线粒度分配时为空），将P1的数据复制到其中并将状态更新为2。

5) 重复步骤1）到4），再次处理P2。

6) 最后，释放所有旧版本的缓存空间。

更新Tail，使其指向与Head相同的条目。

PS: 一次事务涉及多次mfence操作，且次数和更新的page成正比。log只有一个，线程扩展性差。

**为了安全恢复，上述六个步骤的修改要严格按照硬件指令（如clflush和mfence）进行排序**，使数据持久化。此外，在同步操作过程中，页面替换和再生的原子性保证在TCoW中也是成立的。这是因为只有在页面数据被写入SSD设备后，受害者页面才会被释放。此外，在释放NVMM缓存空间时，我们使用CPU提供的64位原子操作来原子化地重置相应的元数据。如果系统在此过程中崩溃，主要数据要么在NVMM缓存中，要么在SSD设备中，或者两者都在，从而避免了系统不一致的问题。

传统上，那些重要的应用程序（如数据库系统）完全绕过本地文件系统进行数据存储，它们将一个大文件映射到应用程序中，并通过自我定义的事务性接口仔细更新数据。有了AFCM中一致的缓存管理，底下的文件系统自然就继承了原子同步语义。为了保证应用数据的一致性，上层应用要做的唯一事情就是正确使用fsync操作。我们的经验表明，在所有评估的工作负载中，由TCoW引入的一致性开销平均限制在0.14%。

#### Crash Recovery

由于TCoW谨慎地按照严格的顺序更新日志条目、缓存数据和相应的元数据，崩溃恢复过程的设计与这种顺序紧密相连。

在系统崩溃重启后，恢复程序将首先检查循环日志中Head和Tail的值。如果它们不相等，NVMM缓存空间的图像需要回滚，因为有一些未完成的事务。详细地说，恢复过程会逐一检查从尾部到头部的每个日志条目的状态。如果所有的日志条目都是2的状态，它只需扫描NVMM页面并释放任何带有旧版本数据的页面。否则，与这些日志条目相关的缓存数据将在日志条目和NVMM元数据区记录的信息帮助下回滚到原始版本：（1）丢弃所有新分配的缓存空间（2）重置NVMM缓存中相关旧版本数据的版本位图，以及（3）通过回退头部指针截断循环日志。

通过上述步骤，NVMM高速缓存空间处于一致状态。随后，扫描位于NVMM缓存全局元数据页区域的状态位图，以确定数据页区域中每个页面的状态（即细粒度、粗粒度或自由）。然后我们读取全局元数据区的页面映射表和每个细粒度数据页头部的缓存线映射表，以重新构建NVMM缓存索引（在第3.2.1节中描述）。同时，细粒度页面中的缓存线数据和粗粒度页面中的页面数据被复制到DRAM缓存中，为正常读写操作服务。

PS：这个故障恢复时间也太久了

请注意，一些异步写入的数据仍然可能丢失，因为AFCM只保证同步操作的原子性。然而，通过正确使用这种语义，应用程序可以有效地保证AFCM中存储数据的一致性，而不必费力地维护自己的日志文件。**此外，AFCM不缓存任何文件系统元数据，只改变文件同步操作的I/O路径，因此所有的文件系统元数据操作都以其原始方式处理**。

## IMPLEMENTATION

在本节中，我们将讨论AFCM的实现细节。由于我们使用跟踪驱动实验来评估AFCM方案，**我们主要开发了一个基于DRAM和NVMM的混合缓存模拟器**，并在这个缓存模拟器的基础上实现AFCM方案。在混合缓存模拟器中，DRAM缓存的默认页面大小被设置为4KB。此外，我们使用最近最少使用（LRU）策略来替换DRAM缓存。也就是说，模拟器维护一个LRU列表来跟踪DRAM缓存中的页面的访问参考（包括读和写参考）的最近性，DRAM缓存中的所有页面都按照它们的最后访问时间进行排序。相比之下，NVMM高速缓存的替换是基于**最近最少写入（LRW）**算法的。为了实现这一点，每个逻辑页的索引列表头被链接到一个全局LRW列表。当一个逻辑页的缓存范围被更新时，这个逻辑页的索引列表头会被移到最近写入（MRW）的位置。虽然AFCM分别选择LRU和LRW算法来替换DRAM和NVMM缓存，但AFCM并不限制DRAM或NVMM缓存使用其他复杂的缓存替换策略。事实上，AFCM可以支持任何缓存替换策略，同时仍然保留其提高缓存命中率的优点。

**PS：简单的LRU线程扩展性肯定差**

为了将我们的模拟器用于跟踪驱动的实验，我们需要将各种形式的跟踪文件预处理成统一格式。这种统一格式包括四种不同类型的字段，即文件操作类型（即读、写、fsync或unlink）、文件编号、文件访问偏移量和文件访问长度。在重放跟踪的过程中，缓存模拟器将首先读取跟踪文件中的每个请求记录，然后根据实现的AFCM方案向DRAM（即读或写请求）或NVMM缓存（即fsync请求）发出相应的请求。当DRAM或NVMM中的空闲缓存空间不足时，**缓存模拟器将在数据写入缓存前基于相应的替换算法清理足够的缓存空间。（PS：缓存算法处于关键路径）**

## 评估

在所有实验中，我们使用DRAM来模拟NVMM设备。除非另有规定，默认的DRAM缓存大小被设置为工作负载大小的一半，而默认的NVMM缓存大小被设置为工作负载大小的20%。（这个设定有点奇怪了）

## RELATED WORK

缓存的替代。随着新兴的非易失性主存储器技术的出现，一些研究考虑了如何使用少量的字节可寻址的NVM来提高基于闪存的SSD的性能和寿命。为了提高缓冲区回写操作的顺序性，闪存感知缓冲区（FAB）管理[9]将同一闪存块中的页面分组，并在缓冲区满时驱逐拥有最大数量页面的组。然而，FAB只考虑组的大小，而忽略了时间性。为了适应时间局部性和群组大小，冷最大群组（CLC）策略[8]结合了FAB和LRU算法。

混合结构。UBJ[19]用NVMM空间联合了页面缓存和日志设备的功能。已**提交的文件数据将在NVM中就地更新，随后检查点到磁盘以释放NVMM空间。**Tinca[20]以CoW的方式将数据持续保存在NVMM缓存中。它采用了日记数据和文件数据之间的角色转换来避免重复写入。UBJ和Tinca在缓存管理（NVMM +SSD/DISK）的结构上与AFCM有一些相似之处，Tinca也采用了循环日志来管理缓存数据。然而，UBJ和Tinca都是以页的粒度来组织NVMM空间，它们的设计在Ext4的架构中是紧密耦合的，而AFCM的设计并不局限于某个特定的文件系统，而是惠及所有的底层文件系统。与上述系统不同，HiNFS是一个基于NVMM的文件系统，旨在通过在DRAM缓存中缓存**懒惰持久的写入数据**来隐藏NVMM的高写入延迟，从而形成DRAM-NVMM混合架构。DRAM缓存中的数据也是以缓存线的粒度来管理的，然而，不需要考虑DRAM缓存的崩溃一致性。

细粒度的缓存管理。为了利用NVMM的字节寻址能力，**PMFS[38]和Chen等人[39]在系统设计中也采用了细粒度的NVMM空间管理。然而，他们都是针对元数据的细粒度日志**，因为元数据的更新往往是小尺寸的。不幸的是，所有现有的工作都忽略了缓存颗粒度对系统性能和文件数据的闪存寿命的影响。由于目前的NVMM技术还不成熟，其容量有限，现有的缓冲缓存方法所使用的粗粒度（即闪存页[11]或闪存块[8]、[9]、[10]的粒度）缓存管理方案大大损害了缓存利用率和缓存命中率，从而降低了系统性能和闪存的耐久性。据我们所知，AFCM是第一个引入自适应细粒度/粗粒度缓存管理来管理文件数据的工作。

[38]  S. Dulloor, S. Kumar, A. Keshavamurthy, P. Lantz, D. Reddy, R. Sankaran, and J. Jackson, “System software for persistent memory,” in Proc. 9th Eur. Conf. Comput. Syst., 2014, Art. no. 15.

[39]  C. Chen, J. Yang, Q. Wei, C. Wang, and M. Xue, “Fine-grained metadata journaling on NVM,” in Proc. 32nd Symp. Mass Storage Syst. Technol., 2016, pp. 1–13.

为了解决这个问题，我们建议使用基于NVMM的高速缓存来减少对SSD设备的同步流量。为了提高NVMM缓存的空间利用率，我们进一步提出了一个自适应的细粒度缓存管理方案，称为AFCM。它采用了一种混合缓存结构，在不降低性能的情况下使用DRAM来服务于正常的读写操作，并让所有同步的数据页流经NVMM缓存空间。为了减轻细粒度管理的元数据开销，AFCM动态地选择细粒度或粗粒度模式来缓存数据，取决于其脏污率。我们还提出了一种名为 "事务性写入时复制 "的崩溃一致性机制，以保证NVMM缓存空间和文件系统镜像的一致性。我们的实验结果表明，与传统的粗粒度缓存管理方案相比，AFCM明显提高了系统性能并减少了对SSD的写入流量。