DOI:10.7544/issn1000-1239.2020.20190062
57(1):85-101,2020

面向非易失内存的数据一致性研究综述

肖仁智[1] 冯 丹[1,2] 胡燏翀[2,3] 张晓祎[1] 程良锋[2]
[1](华中科技大学武汉光电国家研究中心 武汉 430074)
[2](华中科技大学计算机科学与技术学院 武汉 430074)
[3](深圳华中科技大学研究院 广东深圳 518061)
(rzxiao@hust.edu.cn)

A Survey of Data Consistency Research for Non-Volatile Memory

Xiao Renzhi[1], Feng Dan[1,2], Hu Yuchong[2,3], Zhang Xiaoyi[1], and Cheng Liangfeng[2]
[1](*Wuhan National Laboratory for Optoelectronics*, *Huazhong University of Science and Technology*, *Wuhan* 430074)
[2](*School of Computer Science and Technology*, *Huazhong University of Science and Technology*, *Wuhan* 430074)
[3](*Shenzhen Research Institute of Huazhong University of Science and Technology*, *Shenzhen*, *Guangdong* 518061)

Abstract As DRAM technology is facing the bottleneck in density scaling and the problem of high power leakage, novel non-volatile memory (NVM) has drawn extensive attention from academia and industry, due to its superiority in non-volatility, high-density, byte addressability, and low static power consumption. Novel non-volatile memory such as phase change memory (PCM) is likely to substitute or complement DRAM as system main memory. However, due to the non-volatility of NVM, when system failed, data stored in NVM may be inconsistent by reason of partial updates or memory controller write reordering. In order to guarantee the consistency of data in NVM, it is essential to ensure the serialization and persistence in NVM write operations. NVM has inherent drawbacks, such as limited write endurance and high write latency, thus reducing the number of writes can help prolong the lifetime of NVM and improve the performance of NVM-based systems as long as data consistency in NVM is guaranteed. This paper focuses on data consistency based on NVM, especially on persistent indexes, file systems and persistent transactions, and to provide better solutions or ideas for achieving low data consistency overhead. Finally, the possible research directions of data consistency based on NVM are pointed out.

Key words novel non-volatile memory; non-volatile memory (NVM); phase change memory (PCM); system failure; consistency

摘 要 随着DRAM技术面临密度扩展瓶颈以及高泄漏功耗问题,新型非易失内存(non-volatile memory, NVM)因其非易失、高密度、字节寻址和低静态功耗等特性,已经得到学术界和工业界的广泛关注.新型非易失内存如相变内存(phase change memory, PCM)很可能替代DRAM或与DRAM混合作为系统主内存.然而,由于NVM的非易失特性,存储在NVM的数据在面临系统故障时可能由于部分更新或内存控制器写重排序而产生不一致性的问题.为了保证NVM中数据的一致性,确保对NVM

收稿日期:2019-01-28;**修回日期**:2019-05-30
基金项目:国家重点研发计划项目(2018YFB1003305);国家自然科学基金项目(61772222);深圳市知识创新计划项目(JCYJ20170307172447622)
This work was supported by the National Key Research and Development Program of China (2018YFB1003305); the National Natural Science Foundation of China (61772222) and the Shenzhen Knowledge Innovation Program (JCYJ20170307172447622).
通信作者:冯丹(dfeng@hust.edu.cn)

写操作的顺序化和持久化是基本要求.NVM 有着内在缺陷如有限的写耐久性以及较高的写延迟,在保证 NVM 数据一致性的前提下,减少 NVM 写次数有助于延长 NVM 的寿命并提高 NVM 系统的性能.重点讨论了基于 NVM 构建的持久索引、文件系统以及持久性事务等数据一致性研究,以便为实现低开销的数据一致性提供更好的解决方案或思路.最后给出了基于 NVM 的数据一致性研究展望.

关键词 新型非易失内存;非易失内存;相变存储器;系统故障;一致性

中图法分类号 TP303

随着新型非易失内存技术的发展,涌现出了众多的非易失内存存储器,如相变内存 PCM[1]、自旋转移力矩磁随机存取存储器 STT-MRAM[2]、电阻式随机存储器 ReRAM[3] 以及忆阻器(memristor)[4]等,这些非易失内存(non-volatile memory, NVM)以其非易失、低静态功耗、高密度、字节寻址以及就地更新等特性受到了学术界和工业界的广泛研究.NVM 也被称作存储级内存(storage-class memory, SCM)、持久内存(persistent memory, PM)或非易失随机访问内存(non-volatile random access memory, NVRAM),本文统一称为 NVM.由于这些优良特性,NVM 可以连接到内存总线上,应用程序可以通过 load/store 指令来访问 NVM 上的数据.因此,在未来的计算机系统中,采用 NVM 技术代替 DRAM 作主存或 NVM 与 DRAM 共同作为主存,这已被广泛地讨论并研究[5-15].如图 1 所示,NVM 构建主存主要有 3 种类型[16]:仅 NVM 的单层主存、NVM 和 DRAM 的混合主存以及 DRAM 作为 NVM 的缓存.NVM 最独特的特性是非易失性,即写到 NVM 的数据在断电时不丢失.之前的工作如 BPFS[17], PMFS[18], NV-heaps[19], Mnemosyne[20], Quartz[21]等,假定 NVM 能够保证 8 B 的原子写,NVM 的更新粒度为缓存行大小,通常为 64 B.当前的处理器架构没有提供有序的内存写原语[8].由于 CPU 缓存的易失性和 NVM 之间更新粒度不匹配,并且当前处理器或内存控制器可能重排序内存写以便进一步优化程序性能,所以当系统故障时的部分更新可能导致不一致性情况的发生.因此 NVM 系统面临的最棘手的问题之一是数据一致性[22],即系统在意外崩溃或断电重启之后数据的正确性.

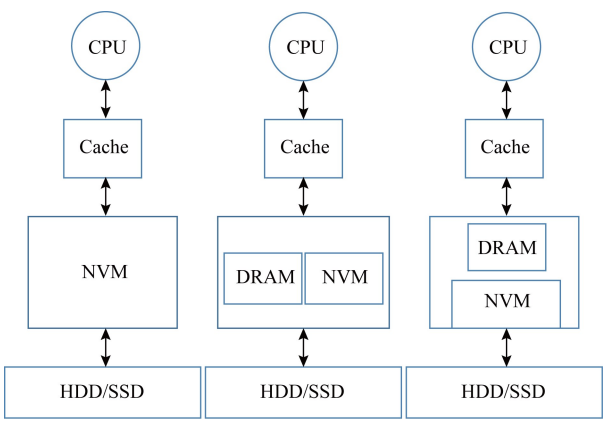

Fig. 1 NVM builds three architectures of main memory[16]

图 1 NVM 构建主存的 3 种结构[16]

非易失内存中数据一致性问题已被广泛地研究[8-14,17-18,23-28].由于 NVM 具有非易失特性,在系统故障时,保证易失性的 CPU 缓存和非易失内存之间的数据一致性是必须的.NVM 的一致性基本需求是确保内存写的持久性和顺序性.然而,当前处理器没有保证有序的内存写操作原语,只提供了缓存行刷新(CLFLUSH)和内存栅栏(MFENCE)指令.缓存行刷新指令 CLFLUSH 功能:给定一个内存地址,CLFLUSH 指令使得包含该内存地址的缓存行失效,并且在系统中进行广播到所有的 CPU 核中;如果特定的缓存行是脏的,它将在失效前写回内存.内存栅栏 MFENCE 功能:MFENCE 指令保证在程序顺序中的 MFENCE 之前发出的所有内存读取和内存写入比在 MFENCE 指令之后的任何读取或写入之前变为全局可见(即优先写入到非易失内存中).但是,仅靠 MFENCE 指令不会写回任何脏的缓存行.多个 CLFLUSH 指令可能并行执行.然而,脏缓存行(dirty cache lines)可能以任意顺序写回.为了确保 2 个脏缓存行的写回顺序,需要 MFENCE 指令来使得 CLFLUSH 刷新有序.因此,可以在 CLFLUSH 指令之间插入 MFENCE 指令以便确保脏的缓存行写回顺序[9].

然而,CLFLUSH 和 MFENCE 的指令的序列化操作导致了大的开销.另一个重要的方面是维持一致性的写大小,假设 NVM 的故障原子写(failure-atomic write)粒度为 8B,写粒度小于 8B 的更新,可以直接应用原子 8 B 写;但是当 NVM 写单元大于 8B 时,需要依赖于传统方法如日志或写时拷贝(copy-on-write, COW)等机制维持数据一致性.然

而,日志或写时拷贝(COW)机制存在双倍写问题,这增加了缓存行刷新的开销和大量的 NVM 写操作[21].因此,我们需要在保证一致性的前提下尽量减少维护 NVM 中数据一致性带来的开销.

表 1 列举了 DRAM 和 3 种新型非易失内存器件的主要指标.从表 1 可以看出,NVM 技术如 PCM 在密度和读速度方面具有很好的优势.然而非易失内存 NVM 具有 2 方面局限性:1)有限的写耐久性,比如 PCM 的写耐久性为 $10^8 \sim 10^9$ 擦除次数[29],在反复写同一个存储单元情况下,存储器会很快失效;2)读写不对称性,比如 PCM 写延迟大于 DRAM,PCM 的写延迟大约是 DRAM 的 2～15 倍,而读延迟是 DRAM 的 2～6 倍[29].因此,除了减少 NVM 一致性开销,还需要尽可能减少 NVM 的有限的写入耐久性和读写不对称的影响[5-7,30-31],从而提高 NVM 系统的寿命和性能.

Table 1 Comparison of Features Between New Non-volatile Memory and Traditional Memory Devices[29]

表 1 新型非易失内存与传统易失性内存器件的特征比较[29]

Storage Device	Cell Size/F^2	Write Endurance	Read Latency/ns	Write Latency/ns	Non-volatile	Leakage Power
DRAM	60-100	$>10^{15}$	10	10	N	Medium
PCM	4-12	10^8-10^9	20-60	20-150	Y	Low
STT-MRAM	6-50	10^{12}-10^{15}	2-35	3-50	Y	Low
ReRAM	4-10	10^8-10^{11}	10	50	Y	Low

近年来,保证 NVM 数据一致性成为了 NVM 的研究热点,无论是从基于 NVM 的持久索引、文件系统还是从持久性事务等方面均存在数据一致性问题,并且已经得到不少的研究和关注.鉴于 PCM 技术最成熟、容量也最大,而 ReRAM 实际容量小且处于实验室阶段,因此,本文的主要关注点是如何使得基于以 PCM 为主的 NVM 系统面临故障时,确保数据一致性的同时尽可能降低一致性维护带来的额外开销.

1 基于 NVM 的内存索引的数据一致性研究

近年来,基于 NVM 的内存索引的数据一致性工作得到了广泛的关注和研究.B+Tree 以及变体由于高效的范围扫描操作和相对好的对数级查询能力被广泛应用[9,11,15],而散列索引[23-25,32-37]因为常量级的查询时间而被广泛应用,基数树[13,38]和红黑树[14]等树结构与 NVM 的结合也引发了关注和研究,不同内存索引的混合以便充分发挥各自索引的优势而屏蔽各自索引的劣势,比如 HiKV[26]的数据一致性研究.接下来从 Tree 索引、散列索引以及混合索引等方面分别介绍相关的数据一致性工作.

1.1 基于 Tree 索引结构的一致性研究

NVM 和 Tree 索引结构相结合的持久索引的一致性研究得到了广泛关注[8-14].然而以 PCM 为主 NVM 具有有限的写耐久性和读写不对称性,在保证基于 NVM 的 Tree 索引结构如 B-Tree 或 B+Tree 及其变体[8-12]、基数树[13]以及红黑树[14]等的数据一致性的基本前提下,减少一致性开销的同时需要尽量避免 NVM 写以便延长 NVM 寿命和提高索引结构的整体读写性能.

针对当前处理器或内存控制器能够重排序写并且处理器不能提供有序的内存写原语操作问题,伊利诺伊大学 Venkataraman 等人[8]提出了基于 NVM 的一致且持久的 B-Tree 变体 CDDS B-Tree,它通过采用 64 b 无符号整数的版本化机制,当 CDDS B-Tree 的一个树节点更新时,它创建了更新条目的一个有版本信息的副本而不是原地重写该条目,这保证了可恢复性和一致性.通过 8 B 原子更新而无需日志或写时拷贝操作实现一致性保障.CDDS B-Tree 的节点条目保持有序性以便支持折半查找提高读性能.然而,有序的节点在插入或删除阶段带来了许多条目移动开销,这导致了许多额外的 NVM 写,从而降低了 NVM 的寿命.插入或删除过程也遭受了许多死节点和死条目的影响.由于 CDDS B-Tree 调用了昂贵的 MFENCE 和 CLFLUSH 指令数量同节点内需要重排序的条目数量相等,因此 CDDS B-Tree 的插入和搜索性能相对较差.最后需要注意的是 CDDS B-Tree 中的 B-Tree 指代的是将值仅存储在叶子节点中的 B+Tree.

针对撤销-重做日志和影子技术确保一致性带来的双倍写问题,中国科学院计算技术研究所的 Chen 等人[9]在先前 PCM 友好的 B+Trees 的数据库算法工作[16]基础上,进一步提出了基于 NVM 写

原子的 B+Trees 变体 wB+-Trees,它通过 bitmap 的原子更新来提交绝大部分的改变,除了有时候当叶子节点分裂时需要使用重做日志机制确保一致性,采用了仅追加更新的策略.wB+-Trees 通过采用叶子节点条目的无序性来减少插入过程中写次数,而删除过程仅仅更新 bitmap 区域.为了确保好的搜索性能,wB+-Trees 工作通过添加 slot array 来维持节点中索引条目的有序性,带有 slot array 的非叶子节点支持折半查找,从而能够提供 $O(\log n)$ 的查找性能.slot array 存储了键的有序索引,由于索引比实际的键和指针都小,几个键索引可以通过 8 B 的原子写操作进行原子更新.wB+-Trees 提出使用 8 B 的 bitmap 来更新节点容量,当使用了 bitmap 时,wB+-Trees 需要至少使用 4 个缓存行刷新,如果采用了 slot array,那么缓存行的刷新数量可以降到 2 个.虽然相对于 CDDS B-Tree 的缓存行刷新数急剧下降,但是 B+tree 带来了间接开销.然而,当叶子节点分裂操作时依旧需要昂贵的日志机制或写时拷贝来确保一致性,这带来额外的 NVM 写开销.

针对 CPU 和内存控制器可能会重排序内存写以及 MFENCE&CLFLUSH 指令带来的显著开销等问题,新加坡数据存储研究所 A-STAR 的 Yang 等人[10]提出了基于 NVM 的一致且缓存优化的 B+Tree 变体 NV-Tree,它通过采用仅追加写策略减少了昂贵的 MFENCE 和 CLFLUSH 指令数量.NV-Tree 通过选择执行数据一致性来减少一致性开销,即通过维持存放于 NVM 中的叶子节点等关键数据(critical data)的一致性保障,而对存放于 DRAM 中的中间节点(IN)的可重构数据(reconstructable data)无需做一致性保证,因为内部节点可以通过一致性的叶子节点(LN)来重构.由于追加更新叶子节点策略以及只有叶子节点保存在 NVM 中,NV-Tree 只需要 2 个缓存行刷新,其中一个用于条目(entry)更新,另一个用于条目数(nElements)的更新,结果性能得到了提升.同时也带来了 2 个问题:1)叶子节点依旧保持无序;2)内部节点在系统故障时可能会丢失.通过保持叶子节点条目的无序性并进行原子性的更新,而对内部节点保持有序能够支持好的搜索性能.最后是对内部节点以缓存优化的组织方式来提高缓存局部性,即具体采用连续的内存空间,通过偏移(offset)而非指针来定位内部节点,所有的节点都以缓存行对齐方式存储.然而,由于 NV-Tree 需要所有的内部节点存储在连续的内存块,那么每一个叶子节点的父节点的分裂操作将导致整个内部节点的重构操作.并且,NV-Tree 在 DRAM 中保存内部节点的方式在系统故障时需要一定的时间来恢复.

针对 NVM 读写不对称性问题(主要是延迟方面)以及现有的 B+Tree 变体工作存在持久内存泄露问题以及数据恢复弱点,德国德累斯顿工业大学的 Oukid 等人[11]提出了基于 NVM 和 DRAM 混合内存的持久并发 B-Tree 变体 FPTree,它将内部节点保存在易失性的 DRAM 中而叶子节点保存在非易失性的 NVM 中;FPTree 通过一个字节指纹(fingerprints)来减少缓存行的刷新,指纹是每个叶子节点中键的 1 B 散列.在查找搜索键之前先扫描指纹来限制需要查找键的数量,结果降低了缓存行失效率;FPTree 使得叶子节点中的条目无序,内部节点中的条目保持有序.同 NV-Tree 一样,FPTree 也采用选择持久化(selective persistence)机制降低数据一致性开销,即将叶子节点等主要数据存储在 NVM 中并保证叶子节点一致性,而将内部节点等非主要数据存储在 DRAM 中且不保证内部节点一致性.通过选择并发性机制即对内部节点采用硬件事务内存(HTM)而叶子节点采用细粒度锁确保并发;最后通过持久性指针来解决内存泄漏问题.虽然 FPTree 显示比 NVTree 更好的性能,但是当系统故障时也需要重构存放在 DRAM 中的内部节点.

针对易失性 CPU 缓存与 NVM 之间以缓存行粒度而非页粒度作为数据传输单元,但是故障原子写单元为 8 B 而非 64 B 的缓存行大小,该粒度不匹配问题已经引发了广泛的研究,即基于块的数据结构如 B+Tree 重新设计以便匹配字节寻址的 NVM.然而,各种 B+Tree 变体降低了 B+Tree 在 NVM 中的效率.韩国蔚山国家科学技术研究所(UNIST)Hwang 等人[12]提出了基于 NVM 的可忍受瞬态不一致性的持久 B+Tree 变体 FAST&FAIR,它通过采用插入过程故障原子移动(failure-atomic shift, FAST)解决了粒度不匹配问题,在 FAST 的 8 B 存储(store)指令和故障原子的原地再平衡(failure-atomic in-place rebalance, FAIR)机制可以确保 B+Tree 从一致性状态转换到另一种一致性状态;或者在写线程更改树节点不成功时,FAST 中的 8 B 的存储操作序列和 FAIR 算法可保证任何读线程不会访问到不一致的树节点.通过读操作容忍临时的不一致性可以有效避免昂贵的写时拷贝、日志机制,甚至读取锁存器的必要性,以便读取事务可以是非阻塞的即无锁查询.

当前基于 NVM 的 Tree 索引结构一致性研究主要基于 B-Tree 或 B+Tree 变体,如 CDDS B-Tree[8], wB + Trees[9], NV-Tree[10], FPTree[11], FAST&FAIR[12] 等.然而也有研究开展了基于 NVM 的基数树(radix tree)变体 WORT[13] 和红黑树 CRB-Tree[14] 的一致性工作研究.

针对当前的处理器经常以缓存行粒度重排序内存写操作并且需要采用 MFENCE 和 CLFLUSH 指令执行内存写的有序性,但是 MFENCE 和 CLFLUSH 指令是引发性能下降的主要因素[9-11,39],韩国蔚山国家科学技术研究所(UNIST)的 Lee 等人[13]指出了现有 NVM 的 B-Tree 变体通过削弱原始 B-Tree 的某个关键特征如保持节点中键的有序性.此外,8 B 原子更新写不足以处理节点分裂涉及多个节点更新粒度,该分裂操作仍然需要日志机制保持一致性.而 radix tree 不需要键比较,并且插入或删除操作仅需单个 8 B 更新操作而不再需要树结构的再平衡操作如 B-Tree 分裂或合并操作所引发的节点粒度更新,表明了基数树(radix tree)比 B-Tree 更适用于 NVM.由于原始基数树已知糟糕的内存利用率和缓存效率使其不能直接适应 NVM.为了克服这个局限,基数树采用了路径压缩优化,路径压缩将多个树节点合并成单个节点以便形成唯一的搜索路径.虽然路径压缩显著地提升了基数树的性能,但是它涉及节点分裂和合并操作是对 NVM 有害的.因此,韩国蔚山国立科学技术研究所(UNIST)的 Lee 等人[13]提出了基于 NVM 写优化的基数树变体 WORT,它通过采用 8 B 故障原子写更新(failure-atomic write per update)而无需通过日志或写时拷贝机制来确保一致性,并且所开发的故障原子路径压缩机制的基数树不仅能够确保故障一致性,而且同现有的路径压缩机制的基数树有一样的内存节省效果.对于 WORT 中的节点分裂和合并操作,通过添加 MFENCE 和 CLFLUSH 持久操作确保故障原子写的同时最小化 NVM 写次数以及 MFENCE 和 CLFLUSH 指令操作次数.文献[13]表明了 radix tree 是适合于 NVM 的数据结构,并且提出了优化的 radix tree 变体 WORT 和 WOART 以及 ART+COW,后 2 种是基于自适应的基数树(adaptive radix tree, ART),ART[38]解决了搜索性能和节点利用率之间的平衡,通过采用自适应的节点类型转换能够根据节点利用率来动态地改变树节点的大小,能够进一步提高基数树(radix tree)的空间利用率.并且指出了 ART 中写时拷贝开销显著地小于 B-Tree 及其变体中的写时拷贝开销.

针对大多数当前的内存数据结构都会出现一致性问题,新加坡数据存储研究所 A-STAR 的 Wang 等人[14]首先将计算机系统中广泛使用的重要内存结构红/黑树(RB 树)引入基于 NVM 的系统,并开展了一致性研究工作.由于红黑树具有漫长而复杂的更新过程,因此基于 NVM 的 RB 树很容易出现不一致的问题.他们提出了一种基于 NVM 的一致性红黑树 CRB-Tree,其中包含一种名为 cascade-versioning 的新技术.所提出的 CRB-Tree 具有 2 个特点:1)总是一致且可扩展的;2)在系统崩溃之后不需要恢复过程.实验结果表明,基于 NVM 的 CRB-Tree 不仅获得较低的空间开销和数据一致性的目的,而且有着与普通易失性红黑树相当的性能.

1.2 基于散列索引结构的一致性研究

散列算法因其常量级的查询性能而被广泛使用[23-25,36-37].然而基于 NVM 的散列索引将面临系统故障时数据一致性问题.因为由散列索引固有的散列冲突问题而造成过多的写操作,所以需要在保证基于散列索引结构的数据一致性的同时减少 NVM 的写.美国 NEC 实验室的 Debnath 等人[36]提出了基于 PCM 写友好的布谷鸟散列变体 PFHT,PFHT 提供了布谷鸟散列的优势,比如高效的内存利用率以及保证的查询时间;并且它通过限制主表剔除次数为 1,显著地减少了传统布谷鸟散列算法因散列冲突导致的大量 NVM 写.华中科技大学的 Zuo 等人[37]提出了一种基于 NVM 的成本高效的写入友好路径散列方案 Path Hashing,它通过新颖的散列冲突解决方法位置共享技术,避免了插入和删除过程中额外的 NVM 写操作.虽然 PFHT 和 Path Hashing 都减少了 NVM 写次数,但是它们都没有保证数据的一致性.接下来将对现有的基于散列的一致性工作展开介绍如组散列(Group Hashing)[23], Level Hashing[24], NVLH[25].

针对先前的基于 NVM 的散列工作如 PFHT 和 Path Hashing 等既不考虑高速缓存行刷新(CLFLUSH)和内存栅栏(MFENCE)的成本,也不维护系统在意外故障时的数据一致性,华中科技大学 Zhang 等人[23]首先提出了基于 NVM 的写入高效且一致的散列方案为 Group Hashing.组散列背后的基本思想是降低一致性成本,同时在出现意外系统故障时保证数据一致性.组散列主要贡献有 2 个:1)使用 8 B 故障原子写入来保证数据的一致性,

从而消除了对 NVM 的双倍写入,因此降低了散列表结构的一致性成本;2)为了提高 CPU 缓存效率,组散列利用了一种新颖的组共享(group sharing)技术,它将散列表分成组,并且在每个组中部署一个连续的内存空间来处理散列冲突,从而减少 CPU 缓存缺失,因此组散列在请求延迟方面获得更高的性能.

针对基于 NVM 的数据一致性和硬件限制的要求,最初为 DRAM 设计的传统索引技术在 NVM 中变得低效.为了有效地索引 NVM 中的数据,华中科技大学 Zuo 等人[24]提出了一种基于 NVM 的写优化和高性能的散列索引方案 Level Hashing,它具有低开销的一致性保证和成本高效的 resize 操作.Level Hashing 提供基于共享的 2 级(two-level)散列表,在最坏的情况下实现恒定的时间复杂度的搜索、插入、删除以及更新操作,并且很少引起额外的 NVM 写入.为了保证低开销的一致性,Level Hashing 利用无日志一致性方案进行插入、删除和 resize 操作,并采用机会无日志方案进行更新操作.为了经济高效地调整此散列表的大小,Level Hashing 利用了原地调整大小的方案,只需要重新传输 1/3 的桶(bucket)而不是整个表,因此减少重新散列的桶数并提高调整大小的性能.

针对现有的工作没有研究线性散列的全面设计和优化以便充分利用 NVM,而基于 NVM 的线性散列算法需要保证数据的一致性,香港城市大学的 Wan 等人提出了基于 NVM 的持久且崩溃一致的线性散列 NVLH[25].NVLH 基于操作系统的直接访问(DAX)功能,允许应用程序通过内存映射直接访问 NVM.NVLH 通过高速缓存行刷新指令(CLFLUSH)或弱有序高速缓存行刷新指令(如 Intel 处理器上的 CLFLUSHOPT 和 CLWB)和内存栅栏(MFENCE)的组合来完成 NVM 数据更新的持久化.为了支持崩溃一致性(crash consistency),NVLH 采用基于撤销日志(undo logging)的事务实现,在事务修改原始值(original values)之前将相应的原始值存储到 undo 日志中.

1.3 基于混合索引结构的一致性研究

针对现有基于 DRAM 和 NVM 的混合内存系统的键值存储的索引设计未能充分利用混合内存特定的性能特征,比如 DRAM 的快速写入、NVM 的慢写入以及 DRAM 和 NVM 中的相似的读性能.中国科学院计算技术研究所的 Xia 等人[26]提出了一个基于 NVM 和 DRAM 混合内存的键值存储 HiKV,其核心思想是在混合内存中构建混合索引.为了有效支持丰富的键值操作,HiKV 利用了散列索引和 B+Tree 索引的独特优点.HiKV 在 NVM 中构建并保持散列索引,以保留其快速索引搜索的固有能力.HiKV 在 DRAM 中构建 B+-Tree 索引以支持范围扫描,并避免长时间 NVM 写入以保持 2 个索引的一致性.此外,HiKV 将差异并发方案应用于混合索引,并采用有序写入一致性来确保崩溃一致性.

1.4 NVM 内存索引一致性研究小结

目前基于 NVM 的一致性持久索引主要有基于 Tree 索引结构的研究(如 CDDS B-Tree[8], wB+Trees[9], NV-Tree[10], FPTree[11], FAST&FAIR[12], WORT[13], CRB-Tree[14])和基于散列索引结构研究(如 Group Hashing[23], Level Hashing[24], NVLH[25])以及混合索引 HiKV[26].已有研究 HiKV[26]提出了 B+Tree 和散列的混合索引结构,并对混合索引结构实现了一致性.为了克服日志机制或写时拷贝等双倍写开销,许多工作均采用 8 B 故障原子写方式确保一致性[8-13,23-24],通过弱化 B+Tree 叶子节点中条目的有序性或减少散列冲突带来的条目移动[23-25,36]可以进一步降低 NVM 写;通过选择持久化方式[10-11]可以减少一致性开销;通过结合不同的数据结构的优势如混合索引[26]设计低开销的一致性研究.

2 基于 NVM 的文件系统的数据一致性研究

传统基于 DRAM 和磁盘的文件系统采用检查点机制、Journaling 技术、写时拷贝或日志技术保障系统崩溃时的数据一致性.然而,NVM 有着 DRAM 的性能和磁盘的持久性,学术界和工业界越来越多地开展基于 NVM 的文件系统研究[17-18,27-28,40-45].由于性能原因,当今内存控制器或 CPU 会重排序内存写操作,从而会造成 NVM 写的不一致性情况发生.针对一致性问题,我们从 NVM 作为直接内存文件系统(如 BPFS[17], PMFS[18], FCFS[27])、NVM 和 DRAM 混合内存文件系统(如 NOVA[28]和 HMVFS[43])以及 NVM 做元数据缓存方面文件系统(如 FSMAC[44]和 Fine-grained Metadata Journaling[45])的一致性工作展开论述.

2.1 基于 NVM 单层主存的文件系统数据一致性研究

针对传统的 shadow paging 文件系统如 ZFS 和

WAFL,对文件系统的更新会触发从更改位置到文件系统树根节点之间一连串的更新,即迭代更新问题.微软研究院 Condit 等人[17]提出了字节寻址持久内存文件系统 BPFS,采用 DRAM 模拟的字节寻址持久的 PCM 内存技术能够消除许多传统易失和非易失存储之间的不同,并且提出的短路影子分页(short-circuit shadow paging, SCSP)新技术能够提供细粒度的写时拷贝并且提供文件系统的任何级别的小更新,SCSP 技术克服了传统的 shadow paging 技术带来的从更改位置到文件系统树根节点之间一连串的更新问题,从而降低了更新的成本.SCSP 技术通过 2 种简单的硬件原语使之成为可能:原子的 8 B 写以及 epoch barrier.原子的 8 B 写(atomic 8-byte write)允许 BPFS 通过写单值到 NVM 以便提交更改,从而文件系统数据在断电或故障的时候不会造成不一致.而 epoch barriers 维护 BPFS 中 NVM 写之间的顺序性约束,同时 BPFS 依然可以使用 L1 和 L2 Cache 来提高性能.

针对传统的基于磁盘的文件系统存在双份拷贝开销问题,即:1)从块设备(block device)到页高速缓存(page cache);2)从页高速缓存到用户缓冲(user buffer).Intel 公司的 Dulloor 等人[18]提出了轻量级的持久内存文件系统 PMFS,它可以不通过块层直接访问 PM 设备,并且它遵守 POSIX 接口以便支持传统的应用程序.该轻量级的文件系统以及优化的内存映射 I/O 无需在 DRAM 和辅助存储中拷贝数据,从而避免了双倍拷贝开销.然而,PMFS 主要解决 3 个挑战:1)有序性和持久性等一致性问题;2)如何保护 PM 免受流浪写;3)如何验证以及判断一致性测试的正确性.PMFS 利用处理器的分页和内存排序功能进行优化,例如细粒度日志(fine-grained logging)用于一致性;透明大页面支持用于更快的内存映射 I/O.为了提供强大的一致性保证,PMFS 只需要一个简单的硬件原语 pm_wbarrier,它用于提供软件执行写 PM 的耐久性和有序性保证.最后,PMFS 使用处理器的现有功能来保护 PM 免受流浪写(stray writes),从而提高可靠性.

针对现有的应用程序的故障一致性更新(failure-consistent update)协议主要为磁盘存储系统做优化,然而这些协议的实现没有全面理解底层文件系统的持久性,它们过多的数据拷贝导致面向 NVM 的文件系统面临易错和较差性能的问题,清华大学的 Ou 等人[27]提出了一种针对 NVM 优化的新型文件系统 FCFS,它提供了一系列易于使用的基于文件的接口,以便为应用程序提供正确性和高性能,并且持久更新其数据到 NVM.具体而言,FCFS 基于一致性语义使应用程序能够有效且有选择性地更新多个存储在 NVM 的文件.为此,FCFS 采用 NVM 优化的写优先日志机制,通过充分利用 NVM 的字节可寻址性和高并发属性来降低一致性成本,但是不依赖于页面缓存层技术.首先,混合细粒度日志(hybrid fine-grained logging)将文件系统元数据日志和数据日志分离,以避免错误共享日志数据,并在复制成本和日志跟踪开销之间实现良好的权衡.文件系统元数据采用字节级回滚日志,从而消除高额的元数据索引开销;而文件系统数据采用缓存行粒度的重做日志,以便平衡数据拷贝和索引开销.其次,并行选择性检查点(concurrently selective checkpointing)允许并行执行异步检查点并使用最少的数据副本以提高其效率.

2.2 基于 NVM-DRAM 混合内存文件系统数据一致性

针对于现有的 NVM 文件系统要么具有相似的软件开销,要么无法提供应用程序所需的强一致性保证问题,美国加州大学圣地亚哥分校的 Xu 等人[28]提出了一个基于混合易失和非易失内存的日志结构、POSIX 文件系统 NOVA,它扩展 LFS 并充分利用 NVM 的优势,旨在最大限度地提高混合内存系统的性能,同时提供强的一致性保证.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.特别是,它为每个 inode 维护单独的日志以提高并发性,并将文件数据存储在日志之外,以最小化日志大小并降低垃圾收集成本.NOVA 的日志提供元数据、数据和内存映射(mmap)的原子性,日志结构技术(log-structuring)用于单个日志结构更新,轻量级日志结构(journaling)用于多个日志结构更新,写时拷贝用于文件数据.

针对基于 NVM 的内存文件系统的现有方法侧重于对内存的低开销访问,并且仅保证数据和元数据之间的一致性.上海交通大学的 Zheng 等人[43]提出了一种基于 NVM 和 DRAM 混合内存的有效的版本控制文件系统 HMVFS,它解决了基于 NVM 的内存文件系统的连续快照之间一致性的问题.HMVFS 可以有效地实现容错,并且对 I/O 性能的影响很小.为了强一致性保证,HMVFS 采用分层文件系统树(stratified file system tree, SFST)表示整个文件系统的快照,并且快照的原子性操作得到保证.HMVFS 在故障恢复过程几乎没有 redo 或 undo

开销.HMVFS 充分利用 NVM 的字节寻址特性以字节粒度更新树元数据,采用日志结构的方式对文件更新避免了写放大,提高了 NVM 的写耐久性.

2.3 基于 NVM 的文件系统元数据缓存一致性工作

针对传统文件系统将元数据存放在 DRAM 中并周期性地写回磁盘,而元数据的更新频繁并且每次只更新很小一部分.华中科技大学 Chen 等人[44]提出了一种基于 NVM 的文件系统元数据加速器 FSMAC,通过有效地利用 NVM 的优点来优化元数据访问.FSMAC 解耦数据和元数据 I/O 路径,在运行时将数据放在磁盘上而元数据放在 NVM 上.因此,从 I/O 总线块中访问数据,并且以字节寻址方式从内存总线访问元数据.元数据访问得到显著加速,并且消除了元数据 I/O,因为 NVM 中的元数据不再定期刷回磁盘.FSMAC 引入了一种结合了细粒度版本控制和事务处理的轻量级一致性机制.

针对传统日志机制以 4KB 的块来存储文件系统的元数据,从而造成了 NVM 空间的浪费以及写入放大问题.新加坡数据存储研究所 A-STAR 的 Chen 等人[45]提出了一种基于 NVM 的细粒度的元数据日志机制(fine-grained metadata journaling),以充分利用低延迟字节可寻址的 NVM,从而显著降低传统 Journaling 的开销.新的日志格式采用 128 B 的inode 大小粒度来顺序记录到 NVM,通过对头尾指针以 8B 原子更新的方式标志事务的结束.因此,由于减少了对 NVM 的有序写入量,可以减少日志记录的开销,但由于该日志机制只把元数据记录到日志空间,所以并不能给文件系统带来最强的一致性级别保证.

2.4 基于 NVM 文件系统数据一致性小结

充分利用 NVM 的字节寻址特性,BPFS 提出细粒度的写时拷贝机制,克服迭代更新问题,因此减少了额外的 NVM 写[17].为了平衡索引和数据拷贝开销,FCFS 对文件系统元数据和数据分别采用不同粒度的日志机制,如对元数据采用字节级回滚日志、对数据采用缓存行粒度重做日志[27].充分利用 DRAM 和 NVM 各自的优势,保障数据一致性的同时最大限度地提高混合内存文件系统的性能[28,43].NOVA 对不同的更新采用不同的一致性技术,即对单个日志结构更新采用日志结构技术,对多个日志结构更新采用轻量级日志结构,对文件数据更新采用写时拷贝[28].HMVFS 解决了连续快照间一致性,对元数据更新采用字节粒度,而对文件更新采用日志结构方式[43].为了减少传统文件系统元数据写入放大问题并提高元数据访问效率,采用 NVM 存储元数据来优化元数据的访问效率[44-45].总的来说,面向 NVM 的文件系统可以有效地利用 NVM 的字节寻址特性,减少元数据管理开销,对文件系统元数据和数据采用不同适配粒度的一致性机制,确保数据一致性的同时尽量减少 NVM 写.

3 基于 NVM 的持久性事务的数据一致性研究

为保证 NVM 数据一致性的关键是要确保有序的持久化操作.不再考虑持久索引层面和文件系统层面可能相关工作,本节只从基于 NVM 的持久性事务层面的一致性相关文献出发,并将这些文献划分为实现一致性的 3 个角度,分别为基于事务[19-20,46-53]、持久性模型[54-62]、检查点[63-65]和日志机制[66-72].

3.1 基于事务的数据一致性研究

事务概念主要来自于数据库,有 ACID 四大属性,其中 A,C,I,D 分别代表原子性(atomicity)、一致性(consistency)、隔离性(isolation)以及持久性(durability).传统事务是针对于 CPU-DRAM-HDD 架构设计的,新型 NVM 具有 DRAM 的性能和磁盘的持久性,需要重新设计基于 NVM 的事务机制.保证 NVM 的事务中数据一致性依然要确保有序的持久化操作.基于新型编程接口的事务一致性研究现已开展,如 NV-Heaps[19]和 Mnemosyne[20].

针对易失性和非易失性区域之间指针安全错误问题,加州大学圣地亚哥分校的 Coburn 等人[19]提出了一个基于 NVM 的轻量级高性能的持久对象系统 NV-Heaps,它提供了事务语义,同时防止了这些错误并提供了易于使用和推理的持久性模型.为了指针安全,NV-Heaps 提供了引用计数的 NVM 空间分配机制,并且它通过 undo 日志和事务内存结合的方式提供数据一致性功能.

针对如何在 NVM 持久化数据并保证其数据一致性问题,威斯康星大学麦迪逊分校的 Volos 等人[20]提出了一个基于 NVM 的简单的编程模型 Mnemosyne.Mnemosyne 解决了 2 个挑战:如何创建和管理此类内存,以及如何确保出现故障时的数据一致性.Mnemosyne 通过内存映射方式将 NVM 空间直接映射到用户的地址空间中,用户可以使用关键字“pstatic”对 NVM 空间中的数据进行持久化定义或声明.Mnemosyne 同 NV-Heaps 一样采用事

务内存结合日志的方式确保一致的更新.然而与NV-Heaps不同,Mnemosyne采用的是redo日志.

惠普实验室的Volos等人[46]发现现有的基于内核的组件堆栈非常适合磁盘,不必要地限制了基于NVM的文件系统的设计和实现.他们提出了一种灵活的文件系统架构Aerie,利用拦截系统调用用户库libFS,直接让用户应用程序访问NVM.为了避免直接访问模式下的并发访问和控制问题,Aerie在用户态实现了受信任的文件系统服务(trusted file-system service, TFS)提供并发访问和锁机制.Aerie提供与Mnemosyne一样的持久性原语和redo日志来提供数据一致性保证.

由于严格的持久性模型带来了性能下降,基于NVM的降低顺序性开销的工作已经展开[47,49].

针对遵守严格的持久性内存写入顺序会显著降低系统性能的问题,清华大学的Lu等人[47]提出了一种基于NVM的松散顺序一致性(LOC)的新机制,它满足持久性内存写入的排序要求,其性能下降明显低于最先进的机制.LOC由2个关键技术组成:1)通过消除在事务结束时执行持久性提交记录写入的需要,Eager Commit减少了事务中写入的提交开销.他们通过确保将必要的元数据信息静态地以数据块方式写入内存中来确定恢复期间所有已提交事务的状态来实现此目的.2)Speculative Persistence通过允许推测性地写入NVM来放宽事务之间的写入顺序.只有在关联的事务提交后,才能使推测性写入对软件可见.要启用此功能,LOC机制需要跟踪已提交的事务ID并支持CPU缓存中的多版本控制.

针对现有事务更新对块存储的优化而不适合字节寻址方式,英国爱丁堡大学的Chatzistergiou等人[48]提出了一种基于NVM的用户模式库方法REWIND,用于直接从使用命令式通用语言编写的用户代码管理事务更新.REWIND依赖于支持自身可恢复操作的日志的自定义持久内存数据结构.该方案还采用了非临时更新、持久性内存栅栏和轻量级日志记录(lightweight logging)的组合确保数据一致性.

针对现有基于NVM的事务系统中有些不必要的顺序性控制限制了事务系统的性能问题,密歇根大学的Kolli等人[49]提出一种基于NVM的高效事务编程模型Eager Sync,通过推迟提交事务(deferring commit transactions)机制避免顺序性约束带来的关键路径过长问题,其核心思想是一个事务在修改完数据之后释放相应的锁,这样可以减少顺序性限制,从而提高事务系统的性能.

Intel公司的Doshi等人[50]提出了一个非侵入式内存控制器,它使用后端操作来实现轻量级故障原子性WrAP.通过将同步持久内存操作移动到后台,可以最大限度地降低性能开销.该解决方案通过将隔离和并发驱动的原子性与故障原子性和持久性分离,避免了昂贵的软件干预,并且不需要更改前端缓存层次结构.

基于NVM的数据一致性需要有序的持久化操作.一致性开销包含了顺序化开销和持久化开销,如前面降低顺序化开销的工作[47,49],降低持久化的开销工作也已经展开了研究[51-53].

针对现有事务系统采用撤消日志记录(undo logging)或重做日志记录(redo logging)来保证一致性,但undo logging和redo logging有各自的缺点.清华大学的Liu等人[51]提出了基于NVM的崩溃一致的持久事务系统DudeTM,它结合撤消日志记录和重做日志记录方法以便充分利用undo和redo的优点并且避免它们的不足.具体实现上,DudeTM使用影子DRAM将持久事务的执行分解为3个完全异步的步骤(perform,persist,reproduce),同时规避了undo logging的多次MFENCE操作以及redo logging的额外日志索引操作.DudeTM的优点是只需要最少的内存栅栏(MFENCE)和无需读取内存的指令.此外,DudeTM还可以支持现有的硬件事务内存(HTM).

针对现有的原子性方法如undo logging和写时拷贝COW都需要在关键路径中产生额外的数据拷贝,从而显著增加事务的延迟.加州大学圣地亚哥分校的Memaripour等人[52]提出了一种基于NVM的事务更新的新方法Kamino-Tx,而无需在关键路径中复制任何数据.Kamino-Tx提供原子就地更新,避免在关键路径中创建数据项的拷贝,同时使用异步维护的数据项拷贝提供崩溃一致性.但Kamino-Tx必须克服2个重要的安全挑战,并尽量减少NVM存储开销.他们提出了一种更动态的方法来维护额外的数据副本,以减少存储开销.

美国威斯康星大学麦迪逊分校的Nalli等人[53]提出了一种放手持久化系统(hands-off persistence system, HOPS).基于NVM的应用程序通过在写入NVM之间插入排序点来确保持久性数据的一致性,从而允许构建更高级别的事务机制.该工作提出了一个名为WHISPER的PM基准套件,其中包括

收集的10个NVM应用程序,以涵盖所有当前的NVM接口如直接访问NVM、通过文件系统接口访问NVM以及通过使用NVM事务库方式访问NVM.通过对WHISPER定量分析揭示了4个结论:1)NVM感知应用程序只有4%的数据写入NVM,其余写入易失性内存DRAM;2)软件事务通常用5~50个有序点(ordering points)实现;3)75%的epoch更新恰好是一个64 B大小缓存行;4)80%的epoch取决于来自同一线程的先前的epoch,而少数epoch取决于来自其他线程的epoch.根据这些结论的分析,他们提出了HOPS来跟踪硬件中NVM的更新.当前的硬件设计要求应用程序在每个epoch结束时强制刷新数据到NVM.HOPS为应用程序提供高级ISA原语,并分离持久性和顺序性约束.

3.2 基于持久性模型的数据一致性研究

基于NVM硬件的降低持久化的工作已经展开了研究WSP[54]和Kiln[55].

针对基于NVM的用户级持久堆需要大量修改应用程序,并且仍然具有显著的运行开销.微软研究院剑桥部的Narayanan等人[54]提出了全系统持久性(WSP)作为替代方案.由于NVM技术允许内存状态近乎瞬时恢复而避免传统基于DRAM-HDD系统的后端磁盘存储恢复压力,WSP针对的是所有内存都是NVM的全内存系统,它透明地恢复应用程序的整个状态,使故障显示为挂起/恢复事件.通过使用"故障时刷新(flush on fail)"消除了运行时间开销:处理器寄存器中的瞬态状态和高速缓存仅在发生故障时使用系统电源的剩余能量刷新到NVM.

针对以前的基于NVM的设计使用日志记录或写时拷贝机制来更新持久性数据,然而系统性能大约是没有持久性机制支持的系统的一半性能.宾夕法尼亚州立大学Zhao等人[55]提出了一种采用NVM作为高速缓存的LLC(last level cache)的NVM内存系统Kiln,其性能非常接近没有一致性支持的NVM内存系统.Kiln采用非易失性高速缓存和非易失性内存来实现原子就地更新(atomic in-place updates),严格控制NVM的写入顺序而无需日志或写时拷贝机制.Kiln具有许多实际优势:简单直观的抽象接口、微架构级别优化、从故障中快速恢复,以及消除对NVM的冗余写入.虽然Kiln能够在没有logging或copy-on-write机制的情况下实现一致性,并且它接近无一致性支持的原始NVM内存系统,但是采用NVM作为LLC cache需要传统易失性缓存架构做出大量的修改.

基于NVM控制器的有序性方面已经展开了研究,如3.1节中WrAP[50]和接下来要讲的2个工作分别是FIRM[56]和NVM Duet[57].

针对持久性内存的应用程序与传统(非持久性)应用程序具有非常不同的内存访问特性,持久性应用程序会带来大量额外的写,在传统的内存控制方案下读写分配的公平行和并行性效率都不能得到保证.宾夕法尼亚州立大学Zhao等人[56]提出了一种基于DRAM和NVM混合内存的公平和高性能的内存控制方案FIRM,该系统运行持久性和非持久性应用程序.FIRM包含3个主要想法:1)FIRM将请求源分类为非密集型、流式、随机和持久性,并为每个请求源形成批量请求;2)FIRM跨多个bank的持久内存更新,因此提高了bank级并行性,从而提高了持久内存访问的存储器带宽利用率;3)FIRM以最小化总线周转和写入队列排放的方式批量调度来自不同源的读取和写入请求.

针对未来NVM将同时扮演易失性工作内存和持久性内存的双重角色,任何只考虑单一方面的统一架构将得到次优设计.国立台湾大学的Liu等人[57]提出了一种新颖的统一易失性工作内存(working memory)和持久存储(persistent store)内存架构NVM Duet,它为persistent store提供了所需的一致性和持久性保证,当NVM作为易失性工作内存时将放宽一致性和持久性约束,它采用跨层(cross-layer)设计方法来实现设计目标.NVM Duet包含一个硬件/软件接口,使内存控制器能够区分每次访问的用例,这是一种新颖的内存调度程序,旨在充分利用地址流中存在的bank级并行性,同时遵循程序员指定的写入顺序.保证NVM作为persistent store的一致性以及双保留PCM架构,满足持久内存的持久性要求,同时放宽NVM作为易失性working memory的持久性要求,以换取写延迟减少.

针对限制NVM写入顺序会限制NVM写入并发性并降低吞吐量,从而使得需要新的内存接口来最小化描述写入约束并允许高性能和高并发数据结构.密歇根大学的Pelley等人[58]提出了一种新的持久内存接口Strand Persistency,它建立在内存一致性上.与内存一致性类似,通过减少NVM写入约束,宽松的持久性模型可大大提高系统吞吐量.Strand Persistency还概述了可能的内存持久化模型的设计空间,并实现了2种持久性队列.

针对当前的持久性障碍将高速缓存行刷新添加到关键路径导致大的缓存行刷新开销,爱丁堡大学的 Joshi 等人[59]提出了有效的持久性障碍(efficient persist barrier),可以减少关键路径中发生的缓存行刷新次数.为了确保持久状态的一致性,需要从缓存中定期刷新脏缓存行,并按持久性模型指定的顺序进行持久化.Efficient persist barrier 可以实现 2 种持久性模型:BEP(buffered epoch persistency);以及在具有硬件插入障碍的批量模式下的 BSP (buffered strict persistency).

持久内存系统要求持久有序约束以保证故障时数据的正确恢复.基于软件的持久性和易失性执行相分离的方面已开展了研究[50,53,60-61].其中 WrAP[50] 和 HOPS[53] 已在 3.1 节中介绍过,接下来只介绍 DPO[60] 和 TC.

针对当前的内存系统没有提供有序内存写保证以及紧耦合强制执行顺序性和持久性写入 NVM 对于许多可恢复软件系统而言没有必要.密歇根大学的 Kolli 等人[60]提出了一种基于 NVM 的委托持久化排序(delegated persistent ordering, DPO),其中排序要求明确地传达给 NVM 控制器,完全从易失性执行和高速缓存管理中解耦 NVM 写入排序特性.在 ARMv7 这种宽松内存一致性模型系统中,委托排序(delegated ordering)实现了缓存严格持久性语义(relaxed consistency buffered strict persistency, RCBSP),RCBSP 将持久性排序从线程执行中分离.委托排序明确地公开排序约束,它成功地将持久性强制执行和缓存管理完全解耦.NVM 内存控制器接管持久性排序约束以便最大化提高调度的灵活性.

针对维持 NVM 的数据持久性方案要么产生很大的性能开销,要么需要对现有架构进行大量修改.国立台湾大学的 Lai 等人[61]提出了一种基于 NVM 的持久性内存加速器设计,它通过硬件保证 NVM 数据的持久性,同时保持缓存层次结构和内存控制器操作不变.非易失性事务高速缓存(transaction cache, TC)与高速缓存层次结构保持备用版本的数据更新,并且在不影响原始处理器执行路径的情况下铺设新的持久路径.在 TC 中采用先进先出(FIFO)的队列方式保证持久性内存写的有序性.TC 的设计实现了接近没有持久性保证的性能.

针对最近的工作提出了持久性模型作为内存一致性的扩展,以指定这种排序,那么考虑到持久存储组的原子性和顺序约束,是否有可能存在新的保证语言级持久性模型的分类.密歇根大学的 Kolli 等人[62]提出了一种基于 NVM 的C++11的语言级持久性模型 ARP(acquire-release persistency).ARP 扩展了语言级内存模型,保证了持久写入的顺序.他们描述如何将为 ARP 编写的代码编译为最先进的 ISA 级别持久性模型.然后,考虑对 ISA 级别持久性模型的增强,该模型可以区分正确同步所需的内存一致性约束,但不需要正确的恢复.

3.3 基于检查点和日志机制的数据一致性研究

检查点技术也是保障数据一致性的很好方式,每一次检查点都被认为系统一次有效的一致性状态.故障发生后,检查点之后的数据更新都会被抛弃以便恢复到最近一次检查点状态,因此检查点技术可以有效地减少检查点之间数据一致性更新的数量.基于 NVM 的检查点技术研究已经开展了[63-65].基于软件透明的崩溃一致性检查点技术如 ThyNVM[64],NICO[65].

针对如何在下一代高端机器中使用非易失性内存(NVM)来提供频繁、低开销的检查点.通过调整现有的多级检查点技术,乔治亚理工学院的 Kannan 等人[63]设计了名为 NVM 检查点(NVM-checkpoints)的新方法,可以在本地和远程节点 NVM 上有效地存储检查点,检查点频率由故障模型引导.为了降低开销,NVM 检查点减少了新的预复制机制所使用的 NVM 和互连带宽,该机制在本地检查点启动之前逐渐将检查点数据从 DRAM 移动到 NVM.NVM-checkpoints 通过限制在检查点时移动的瞬时数据量来减少本地检查点成本,从而释放应用程序使用的带宽.NVM-checkpoints 将 NVM 视为内存而不是 RAM 磁盘,因此可以推广预复制以直接将数据移动到远程 NVM.

针对基于 NVM 的系统需要在断电或系统崩溃的情况下保证一致的内存状态(即崩溃一致性),然而为了保证崩溃的一致性,大多数先前的工作需要依赖程序员区分持久和瞬态内存数据,并且在更新持久内存数据时使用专用软件接口,从而使得持久内存的采用可能会受到很大限制.清华大学 Ren 等人[64]提出了一种硬件辅助 DRAM+NVM 混合持久内存设计 ThyNVM(transparent hybrid NVM),它支持混合内存系统中内存数据的软件透明崩溃一致性.为了有效地实施崩溃一致性,他们设计了一种新的双方案检查点(dual-scheme checkpointing)机制,它有效地将检查点时间与应用程序执行时间重叠.关键的新颖性是以协调的方式实现多粒度,即高速缓存行或页粒度的数据检查点.由于检查点导致

的应用程序停顿时间与用于检查点的元数据的硬件存储开销之间存在折中,并且这两者都由检查点数据的粒度决定.为了获得最佳权衡,ThyNVM 使检查点粒度适应数据的写入局部性特征,并协调多粒度更新的管理.

针对现有研究提出的具有软件透明崩溃一致性保证的持久内存设计,以减少程序员在利用 NVM 时的手动努力,但由于创建检查点导致的性能开销,使得这些设计不是最理想的.华中科技大学的 Wei 等人[65]提出了一种基于 NVM 的非侵入式内存控制器设计 NICO,它使用后端操作来实现软件透明的崩溃一致性,同时最小化检查点开销.通过将数据持久化操作移至后台,NICO 完全将数据持久化操作与易失性执行和缓存管理分离.为了有效地实施崩溃一致性,NICO 采用了一种轻量级的检查点方案,在创建 NVM 数据的一致性快照(snapshot)时,只需要刷新和修改非常少量的数据.

汉阳大学的 Hwang 等人[66]开发了一个基于堆的持久对象存储(HEAPO)来管理 NVM 中的持久对象.HEAPO 定义了自己的持久堆布局、持久对象格式、命名空间组织、对象共享和保护机制,以及基于撤消日志的崩溃恢复,所有这些都是为 NVM 有效定制.HEAPO 采用轻量级和灵活的层,以利用类似 DRAM 的 NVM 访问延迟.为了实现这一目标,文献[66]的作者开发了:1)NVM 的本地管理层,以消除 in-core 和磁盘之间的元数据副本冗余;2)可扩展对象格式;3)基于 trie 的全局命名空间和本地命名空间的缓存;4)静态地址绑定;5)仅用于撤销的崩溃恢复的最小日志机制.由于 HEAPO 库使用锁同步对共享对象元数据的访问,因此可以保证共享持久对象的更新一致性.HEAPO 使用基于软件的方法如 CLFLUSH 和 MFENCE 进行 NVM 写的有序性保证.

基于故障原子区(failure-atomic sections, FASEs)的锁机制工作有 Atlas[67]和 JUSTDO logging[68].基于日志实现一致性工作有 3.1 节介绍的 NV-Heaps,以及接下来将介绍的 Atlas[67],JUSTDO logging[68],ATOM[69], Proteus[70], undo+ redo logging[71], EAPR[72]等.其中,采用 undo 日志的有 Atlas,ATOM;基于 redo 日志的有 3.1 节介绍的 Mnemosyne,结合 undo 和 redo logging 的工作有 3.1 节的 DudeTM 和下面的 undo+ redo logging 工作,而 JUSTDO logging 是基于 resumption 的恢复方式.

针对如何确保 NVM 中数据在故障时的一致性.惠普实验室的 Chakrabarti 等人[67]提出了基于 NVM 日志的一致性锁机制 Atlas,它为基于锁的代码添加了持久性语义,通常允许即使在出现故障时也能自动维护一个全局一致的状态.Atlas 识别了现有的关键区代码的故障原子区(failure-atomic sections, FASEs)并描述了一个基于 undo 日志的实现,可用于在故障后恢复一致性状态.对于嵌套锁,只有先持久化内部关键区后才能持久化外层关键区.对于同步锁,只有持久化所有关键区才算整个持久化完成.

针对基于 NVM 的数据在故障时不一致,传统日志记录(logging)方法需要将 log 从易失性 CPU cache 刷新到 NVM 导致了额外的时间和内存开销,而持久缓存(persistent cache)可以消除刷新的需要,但是传统的 undo 或 redo logging 管理开销依然存在.惠普实验室的 Izraelevitz 等人[68]提出了一种基于 NVM 的新的故障原子日志机制 JUSTDO logging,可以大大减少日志的内存占用,简化日志管理,并实现快速并行故障后恢复. JUSTDO logging 与传统的 undo 和 redo logging 不同,不会舍弃在故障原子区(FASEs)中所做的更新,而是在最后一条 store 指令处重新执行每个被中断的 FASE,直到执行 FASE 完成.与 Atlas 一样,JUSTDO logging 将 FASE 定义为受一个或多个互斥锁保护的最外层临界区.

针对通过用于撤销日志(undo log)的流存储(streaming stores)来进行软件日志(software logging)或依靠 CLFLUSH 和 MFENCE 组合的重做日志(redo log)等现有技术,它们浪费了宝贵的执行周期来实现日志记录(logging).爱丁堡大学的 Joshi 等人[69]提出了一种基于 NVM 的撤消日志记录(undo logging)的硬件日志管理器 ATOM,它可以执行关键路径之外的 logging 操作.ATOM 通过 2 种方式在硬件中透明地执行日志记录:1)将日志写入与数据存储耦合;2)在同一存储控制器中共存数据及其相应的日志条目.因此,ATOM 不仅可以最大限度地减少数据移动,而且可以在内存控制器中执行在关键路径之外的日志到数据的排序约束.

针对基于 NVM 持久事务引入的高日志开销,北卡罗来纳州立大学的 Shin 等人[70]提出了一种基于 NVM 新的硬件日志记录方法 Proteus,它用于持久事务,实现了先前软件和硬件方法的有利特性.与软件一样,Proteus 没有限制可用的事务或日志数量等硬件限制,而且像硬件一样,它的开销非常低.

Proteus 引入了 2 条新指令:log-load 通过加载原始数据创建日志条目;log-flush 将日志条目写入 NVM.Proteus 主要在 core 内添加硬件支持,以管理这些指令的执行以及 logging 操作和数据更新之间的关键有序要求.Proteus 通过明智的接口设计避免了对事务大小或数量的任何限制:软件仍然可以控制分配日志区域,硬件可以将更新日志的成本保持在较低水平.

针对大多数先前的 NVM 设计通过仔细控制写 NVM 顺序带来次优性能的问题,一些研究如软件日志(software logging)或硬件日志(hardware logging)方案放松了这种写控制,但这些方案要么需要在处理器中添加昂贵的 NVM 高速缓存,要么因为无法提供足够的内存读带宽给关键读取操作而回退到低性能模式.加州大学圣克鲁兹分校的 Ogleari 等人[71]提出了基于 NVM 的硬件撤销+重做日志记录方案 undo+redo logging 来放宽这种写顺序控制.该方案通过利用商业缓存中使用的回写、写分配策略来维护数据持久性.此外,他们在硬件中开发缓存强制回写机制,以显著降低强制写入数据到 NVM 的性能和能量开销.与软件方法相比,undo+redo logging 还提供了强大的一致性保证.

针对大多数现有的页面置换方法都侧重于延长 NVM 内存系统的使用寿命,而 NVM 内存系统要么考虑能耗成本,要么处理由系统故障引起的一致性错误,电子科技大学的 Zhan 等人[72]提出了一种基于 NVM-DRAM 混合主存储系统的页面替换方法称为能量感知页面替换 EAPR,用于低功耗和一致性保证.EAPR 利用页面分组策略来提高 NVM 和 DRAM 之间的迁移效率.选择页面组根据其与能量相关的替代度进行迁移.为保证页面迁移过程的一致性,EAPR 在页面替换阶段和事务提交阶段设计了 2 种一致性保证方案.EAPR 不是直接删除日志,而是通过在提交应用程序的事务之后修改日志的数据结构来保证混合内存系统的一致性.

3.4 小 结

数据一致性的 2 个基本要素是顺序化和持久化,那么降低顺序化开销或持久化开销也就等于降低了数据一致性维护开销.降低软件持久化开销的工作有 NV-Heaps[19],Mnemosyne[20],REWIND[48],DudeTM[51],Kamino-TX[52],HOPS[53],Heapo[66]等,降低硬件持久化开销的工作有 WSP[54],Kiln[55],Strand Persistency[58],efficient persist barrier[59],ARP[62],ATOM[69]等,降低硬件顺序化开销的工作有 LOC[47],Eager Sync[49],Strand Persistency[58],DPO[60]等.基于检查点机制的数据一致工作有 NVM-checkpoints[63],ThyNVM[64],NICO[65]等,基于日志机制的数据一致性工作有 Heapo[66],JUSTDO logging[68],undo+redo logging[71]等.总的来说,面向 NVM 的持久性事务要减少数据一致性开销,要么降低顺序化开销[47,49,58,60],要么降低持久化开销[19-20,48,51-55,58-59,62],考虑到 NVM 有限写耐久性和读写不对称性,需要确保 NVM 中数据一致性并减少 NVM 写,从而提高面向 NVM 的持久性事务的性能.

4 基于 NVM 的数据一致性研究展望

非易失内存 NVM 同时具有 DRAM 的性能和磁盘的非易失特性,由于保存在非易失内存中的数据永久保存,因此传统基于易失性 DRAM 的数据结构需要重新设计以便充分利用 NVM 的非易失、低静态功耗、高密度以及字节寻址等特性,需要确保 NVM 中数据一致性前提下并且尽量避免或减少 NVM 有限写耐久性以及读写不对称的影响.

4.1 持久索引层面

1) 混合内存索引一致性

HiKV 实现了散列和 B+Tree 混合索引的一致性研究,未来有可能会考虑红黑树、前缀树、散列结构、B-Tree 或 B+Tree 及其变体等其他多种数据结构的混合索引.混合索引的一致性研究需要保证一致性的同时最大化地利用各种数据结构索引的优势并避免它们的不足.

2) 开发新的一致性的持久内存索引

虽然 Tree 和散列 2 种数据结构研究的比较多.未来也可能会出现更多的数据结构与 NVM 结合做数据一致性研究工作,比如日志结构合并树(log-structured merge-tree,LSM-Tree)、跳表(skiplist)等.

3) NVM 介质内部差异性的影响

由于 NVM 具有有限的写入耐久性和读写延迟及能耗的不对称性,需要我们在保证数据一致性前提下,尽可能减少 NVM 系统的写次数、延迟及能耗.因为当前基于 NVM 持久索引的一致性研究并未考虑底层存储介质单元的差异性,比如 PCM 存在 SLC 和 MLC 在可靠性和耐久性以及性能间的不同,所以基于 PCM 的内存索引一致性研究需要进一步设计针对特定介质差异化引发对一致性机制的重新设计.虽然当前 PCM 最成熟,单片容量也最大,

基于 NVM 的数据一致性工作主要针对 PCM 而设计,未来随着 ReRAM 技术的进一步发展成熟,那么基于 ReRAM 的一致性持久索引工作或许会得到众多关注.

4.2 文件系统层面和持久性事务层面

大部分 NVM 数据一致性工作主要只考虑 NVM 的非易失和写耐久性特性,很少考虑具体的底层存储介质如 PCM 分为 SLC 和 MLC,而 SLC PCM 又存在写 0、写 1 的延迟和能耗的不对称性,SLC 和 MLC 之间也有不同,那么如何结合具体底层存储单元的数据一致性研究工作依然具有深入的研究空间.

1) NVM 介质内部差异性对文件系统的影响

NVM 在文件系统主要以 3 种方式存在:①NVM 作为内存设备,通过传统 VFS 路径的文件系统如 BPFS[17], PMFS[18], FCFS[27], NOVA[28], SCMFS[40], HiNFS[42]等;②NVM 作为块设备的文件系统如 Shortcut-JFS[73], On-demand Snapshot[74]等;③使用用户库方式绕过 VFS 直接访问 NVM 的文件系统如 Aerie[46]等.NVM 介质内部差异性将会影响原有针对 NVM 共性而设计的文件系统研究工作,需要重新设计新的一致性研究方案.

2) NVM 介质内部差异性对持久性事务的影响

基于 NVM 的持久性事务一致性工作分别为降低持久化开销和降低顺序化开销.降低 NVM 持久化开销的工作有 DudeTM[51], Kamino-TX[52], HOPS[53], WSP[54], Kiln[55]等,降低顺序化的开销工作的有 Mnemosyne[20], LOC[47], Eager Sync[49], HOPS[53], Strand Persistency[58], DPO[60]等.这些工作主要针对 NVM 共性设计,由于以 PCM 为主的 NVM 介质存在内部差异性,基于 NVM 的持久性事务的数据一致性研究需要针对不同的存储介质单元采用不同的策略保障一致性,同时尽可能提升 NVM 系统的性能和寿命并降低能耗和读写延迟.

5 总 结

以 PCM 为代表的非易失内存(NVM)具有非易失、高密度以及低静态功耗,使得 NVM 成为 DRAM 的有力替代者或与 DRAM 共同作为系统主存.新型存储器给系统结构和软件设计带来了新的机遇和挑战.由于现代处理器或内存控制器可能会重排序内存写,因此,系统故障时的部分更新或重排序写会导致 NVM 中数据的不一致性问题.当前基于 NVM 的数据一致性被广泛研究,我们从持久索引层面、文件系统层面和持久性事务等方面分别介绍了相关工作,并且给出了未来基于 NVM 数据一致性方面可能重要的研究发展方向.NVM 固有的缺点如 PCM 读写延迟高、读写不对称和有限的擦写次数等,需要研究人员在性能和开销之间寻找最佳的平衡点.为了充分利用 NVM 实现更好的数据一致性,可能需要从应用层、软件层以及硬件层等多角度多层面协同设计.

参 考 文 献

[1] Zhang Hongbin, Fan Jie, Shu Jiwu, et al. Summary of storage system and technology based on phase change memory [J]. Journal of Computer Research and Development, 2014, 51(8): 1647-1662 (in Chinese)
(张鸿斌, 范捷, 舒继武, 等. 基于相变存储器的存储系统与技术综述[J]. 计算机研究与发展, 2014, 51(8): 1647-1662)

[2] Kultursay E, Kandemir M, Sivasubramaniam A, et al. Evaluating STT-RAM as an energy-efficient main memory alternative [C] //Proc of IEEE Int Symp on Performance Analysis of Systems & Software (ISPASS 2013). Piscataway, NJ: IEEE, 2013: 256-267

[3] Mao Manqing, Cao Yu, Yu Shimeng, et al. Optimizing latency, energy, and reliability of 1T1R ReRAM through appropriate voltage settings [C] //Proc of the 33rd IEEE Int Conf on Computer Design (ICCD 2015). Piscataway, NJ: IEEE, 2015: 359-366

[4] Strukov D B, Snider G S, Stewart D R, et al. The missing memristor found [J]. Nature, 2008, 453(7191): 80-83

[5] Zhou Ping, Zhao Bo, Yang Jun, et al. A durable and energy efficient main memory using phase change memory technology [J]. ACM SIGARCH Computer Architecture News, 2009, 37(3): 14-23

[6] Qureshi M K, Srinivasan V, Rivers J A. Scalable high performance main memory system using phase-change memory technology [J]. ACM SIGARCH Computer Architecture News, 2009, 37(3): 24-33

[7] Qureshi M K, Karidis J P, Franceschini M, et al. Enhancing lifetime and security of PCM-based main memory with start-gap wear leveling [C] //Proc of the 42nd Annual IEEE/ACM Int Symp on Microarchitecture (MICRO 2009). Piscataway, NJ: IEEE, 2009: 14-23

[8] Venkataraman S, Tolia N, Ranganathan P, et al. Consistent and durable data structures for non-volatile byte-addressable memory [C] //Proc of the 9th USENIX Conf on File and Stroage Technologies (FAST 2011). Berkeley, CA: USENIX Association, 2011: 5-19

[9] Chen Shimin, Jin Qin. Persistent B+-trees in non-volatile main memory [J]. Proceedings of the VLDB Endowment, 2015, 8(7): 786-797

[10] Yang Jun, Wei Qingsong, Chen Cheng, et al. NV-Tree: Reducing consistency cost for NVM-based single level systems [C] //Proc of the 13th USENIX Conf on File and Storage Technologies (FAST 2015). Berkeley, CA: USENIX Association, 2015: 167-181

[11] Oukid I, Lasperas J, Nica A, et al. FPTree: A hybrid SCM DRAM persistent and concurrent B-tree for storage class memory [C] //Proc of the 2016 Int Conf on Management of Data (SIGMOD 2016). New York: ACM, 2016: 371-386

[12] Hwang D, Kim W, Won Y, et al. Endurable transient inconsistency in byte-addressable persistent B^+-tree [C] //Proc of the 16th USENIX Conf on File and Storage Technologies (FAST 2018). Berkeley, CA: USENIX Association, 2018: 187-200

[13] Lee S, Lim K, Song H, et al. WORT: Write optimal radix tree for persistent memory storage systems [C] //Proc of the 15th USENIX Conf on File and Storage Technologies (FAST 2017). Berkeley, CA: USENIX Association, 2017: 257-270

[14] Wang Chundong, Wei Qingsong, Wu Lingkun, et al. Persisting RB-tree into NVM in a consistency perspective [J]. ACM Transactions on Storage, 2018, 14(1): 6-32

[15] Chi Ping, Lee Wang-Chien, Xie Yuan. Adapting B^+-tree for emerging non-volatile memory-based main memory [J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2016, 35(9): 1461-1474

[16] Chen Shimin, Gibbons P B, Nath S. Rethinking database algorithms for phase change memory [C/OL] //Proc of Int Conf on Innovation Database Research (CIDR 2011). 2011: 21-31 [2013-03-10]. http://cidrdb.org/cidr2011/Papers/CIDR11_Paper3.pdf

[17] Condit J, Nightingale E B, Frost C, et al. Better I/O through byte-addressable, persistent memory [C] //Proc of the 22nd ACM Symp on Operating Systems Principles (SOSP 2009). New York: ACM, 2009: 133-146

[18] Dulloor S R, Kumar S, Keshavamurthy A, et al. System software for persistent memory [C] //Proc of the 9th European Conf on Computer Systems (EuroSys 2014). New York: ACM, 2014: 15

[19] Coburn J, Caulfield A M, Akel A, et al. NV-heaps: Making persistent objects fast and safe with next-generation, non-volatile memories [C] //Proc of the 16th Int Conf on Architectural Support for Programming Languages & Operating Systems (ASPLOS 2011). New York: ACM, 2011: 105-118

[20] Volos H, Tack A J, Swift M M. Mnemosyne: Lightweight persistent memory [J]. ACM SIGARCH Computer Architecture News, 2011, 39(1): 91-104

[21] Volos H, Magalhaes G, Cherkasova L, et al. Quartz: A lightweight performance emulator for persistent memory software [C] //Proc of the 16th Conf on middleware (Middleware 2015). New York: ACM, 2015: 37-49

[22] Mutlu O, Subramanian L. Research problems and opportunities in memory systems [J]. Supercomputing Frontiers and Innovations, 2015, 1(3): 19-55

[23] Zhang Xiaoyi, Feng Dan, Hua Yu, et al. A write-efficient and consistent hashing scheme for non-volatile memory [C] //Proc of the 47th Int Conf on Parallel Processing (ICPP 2018). Piscataway, NJ: IEEE, 2018: 87

[24] Zuo Pengfei, Hua Yu, Wu Jie. A write-optimized and high-performance hashing index scheme for persistent memory non-volatile memory [C] //Proc of the 13th USENIX Symp on Operating Systems Design and Implementations (OSDI 2018). Berkeley, CA: USENIX Association, 2018: 461-476

[25] Wan Hu, Li Fuyang, Zhou Zimeng, et al. NVLH: Crash-consistent linear hashing for non-volatile memory [C] //Proc of the 7th IEEE Non-Volatile Memory Systems and Applications Symp (NVMSA 2018). Piscataway, NJ: IEEE, 2018: 117-118

[26] Xia Fei, Jiang Dejun, Xiong Jin, et al. Hikv: A hybrid index key-value store for DRAM-NVM memory systems [C] //Proc of USENIX Annual Technology Conf (ATC 2017). Berkeley, CA: USENIX Association, 2017: 349-362

[27] Ou Jiaxin, Shu Jiwu. Fast and failure-consistent updates of application data in non-volatile main memory file system [C] //Proc of the 32nd Symp on Mass Storage Systems and Technologies (MSST 2016). Piscataway, NJ: IEEE, 2016: 1-15

[28] Xu Jian, Swanson S. NOVA: A log-structured file system for hybrid volatile/non-volatile main memories [C] //Proc of the 14th USENIX Conf on File and Storage Technologies (FAST 2016). Berkeley, CA: USENIX Association, 2016: 323-338

[29] Boukhobza J, Rubini S, Chen Renhai, et al. Emerging NVM: A survey on architectural integration and research challenges [J]. ACM Transactions on Design Automation of Electronic Systems, 2018, 23(2): 14-45

[30] Yue Jianhui, Zhu Yifeng. Accelerating write by exploiting PCM asymmetries [C] //Proc of the 19th IEEE Int Symp on High Performance Computer Architecture (HPCA 2013). Piscataway, NJ: IEEE, 2013: 282-293

[31] Li Zheng, Wang Fang, Feng Dan, et al. Tetris write: Exploring more write parallelism considering PCM asymmetries [C] //Proc of the 45th Int Conf on Parallel Processing (ICPP 2016). Piscataway, NJ: IEEE, 2016: 159-168

[32] Garciamolina H, Salem K. Main memory database systems: An overview [J]. IEEE Transactions on Knowledge & Data Engineering, 1992, 4(6): 509-516

[33] Pagh R, Rodler F F. Cuckoo hashing [J]. Journal of Algorithms, 2004, 51(2): 122-144

[34] Fan Bin, Andersen D G, Kaminsky M. MemC3: Compact and concurrent memcache with dumber caching and smarter hashing [C] //Proc of the 10th USENIX Conf on Networked Systems Design & Implementation (NSDI 2013). Berkeley, CA: USENIX Association, 2013: 371-384

[35] Rumble S M, Kejriwal A, Ousterhout J. Log-structured memory for DRAM-based storage [C] //Proc of the 12th USENIX Conf on File and Storage Technologies (FAST 2014). Berkeley, CA: USENIX Association, 2014: 1-16

[36] Debnath B, Haghdoost A, Kadav A, et al. Revisiting Hash table design for phase change memory [J]. ACM SIGOPS Operating Systems Review, 2016, 49(2): 18-26

[37] Zuo Pengfei, Hua Yu. A write-friendly hashing scheme for nonvolatile memory systems [C] //Proc of the 33rd IEEE Symp on Mass Storage Systems and Technologies (MSST 2017). Piscataway, NJ: IEEE, 2017: 1-10

[38] Leis V, Kemper A, Neumann T. The adaptive radix tree: ARTful indexing for main-memory databases [C] //Proc of the 29th IEEE Int Conf on Data Engineering (ICDE 2013). Piscataway, NJ: IEEE, 2013: 38-49

[39] Kim W H, Kim J, Baek W, et al. NVWAL: Exploiting NVRAM in write-ahead logging [J]. ACM SIGOPS Operating Systems Review, 2016, 50(2): 385-398

[40] Wu Xiaojian, Reddy A L. SCMFS: A file system for storage class memory [C] //Proc of the 24th Int Conf for High Performance Computing, Networking, Storage and Analysis (SC 2011). New York: ACM, 2011: 39

[41] Sha Edwin H-M, Chen Xianzhang, Zhuge Qingfeng, et al. Designing an efficient persistent in-memory file system [C] //Proc of Non-Volatile Memory System and Applications Symp (NVMSA 2015). Piscataway, NJ: IEEE, 2015: 1-6

[42] Ou Jiaxin, Shu Jiwu, Lu Youyou. A high performance file system for non-volatile main memory [C] //Proc of the 11th European Conf on Computer Systems (EuroSys 2016). New York: ACM, 2016: 12

[43] Zheng Shengan, Liu Hao, Huang Linpeng, et al. HMVFS: A versioning file system on DRAM/NVM hybrid memory [J]. Journal of Parallel and Distributed Computing, 2018, 120: 355-368

[44] Chen Jianxi, Wei Qingsong, Chen Cheng, et al. FSMAC: A file system metadata accelerator with non-volatile memory [C] //Proc of the 29th Symp on IEEE Mass Storage Systems and Technologies (MSST 2013). Piscataway, NJ: IEEE, 2013: 1-11

[45] Chen Cheng, Yang Jun, Wei Qingsong, et al. Fine-grained metadata journaling on NVM [C] //Proc of the 32nd Symp on IEEE Mass Storage Systems and Technologies (MSST 2016). Piscataway, NJ: IEEE, 2016: 1-13

[46] Volos H, Nalli S, Panneerselvam S, et al. Aerie: Flexible file-system interfaces to storage-class memory [C] //Proc of the 9th European Conf on Computer Systems (EuroSys 2014). New York: ACM, 2014: 14

[47] Lu Youyou, Shu Jiwu, Sun Long, et al. Loose-ordering consistency for persistent memory [C] //Proc of the 32nd IEEE Int Conf on Computer Design (ICCD 2014). Piscataway, NJ: IEEE, 2014: 216-223

[48] Chatzistergiou A, Cintra M, Viglas S D. Rewind: Recovery write-ahead system for in-memory non-volatile data-structures [J]. Proceedings of the VLDB Endowment, 2015, 8(5): 497-508

[49] Kolli A, Pelley S, Saidi A, et al. High-performance transactions for persistent memories [J]. ACM SIGOPS Operating Systems Review, 2016, 50(2): 399-411

[50] Doshi K, Giles E, Varman P. Atomic persistence for SCM with a non-intrusive backend controller [C] //Proc of IEEE Int Symp on High Performance Computer Architecture (HPCA 2016). Piscataway, NJ: IEEE, 2016: 77-89

[51] Liu Mengxing, Zhang Mingxing, Chen Kang, et al. DudeTM: Building durable transactions with decoupling for persistent memory [J]. ACM SIGOPS Operating Systems Review, 2017, 51(2): 329-343

[52] Memaripour A, Badam A, Phanishayee A, et al. Atomic in-place updates for non-volatile main memories with kamino-tx [C] //Proc of the 12th European Conf on Computer Systems (EuroSys 2017). New York: ACM, 2017: 499-512

[53] Nalli S, Haria S, Hill M D, et al. An analysis of persistent memory use with WHISPER [J]. ACM SIGARCH Computer Architecture News, 2017, 45(1): 135-148

[54] Narayanan D, Hodson O. Whole-system persistence [J]. ACM SIGARCH Computer Architecture News, 2012, 40(1): 401-410

[55] Zhao Jishen, Li Sheng, Yoon D H, et al. Kiln: Closing the performance gap between systems with and without persistence support [C] //Proc of the 46th Annual IEEE/ACM Int Symp on Microarchitecture (MICRO 2013). Piscataway, NJ: IEEE, 2013: 421-432

[56] Zhao Jishen, Mutlu O, Xie Yuan. FIRM: Fair and high-performance memory control for persistent memory systems [C] //Proc of the 47th Annual IEEE/ACM Int Symp on Microarchitecture (MICRO 2014). Piscataway, NJ: IEEE, 2014: 153-165

[57] Liu Renshuo, Shen Deyu, Yang Chialin, et al. NVM duet: Unified working memory and persistent store architecture [J]. ACM SIGPLAN Notices, 2014, 49(4): 455-470

[58] Pelley S, Chen Peter M, Wenisch T F. Memory persistency [C] //Proc of Int Symp on Computer Architecuture (ISCA 2014). New York: ACM, 2014: 265-276

[59] Joshi A, Nagarajan V, Cintra M, et al. Efficient persist barriers for multicores [C] //Proc of the 48th Annual IEEE/ACM Int Symp on Microarchitecture (MICRO 2015). New York: ACM, 2015: 660-671

[60] Kolli A, Rosen J, Diestelhorst S, et al. Delegated persist ordering [C] //Proc of the 49th Annual IEEE/ACM Int Symp on Microarchitecture (MICRO 2016). Piscataway, NJ: IEEE, 2016: No.58

[61] Lai Chunhao, Zhao Jishen, Yang Chialin. Leave the cache hierarchy operation as it is: A new persistent memory accelerating approach [C] //Proc of the 54th Annual Design Automation Conf (DAC 2017). New York: ACM, 2017: 1-6

[62] Kolli A, Gogte V, Saidi A, et al. Language-level persistency [C] //Proc of the 44th ACM/IEEE Annual Int Symp on Computer Architecture (ISCA 2017). New York: ACM, 2017: 481-493

[63] Kannan S, Gavrilovska A, Schwan K, et al. Optimizing checkpoints using NVM as virtual memory [C] //Proc of the 27th Int Symp on Parallel and Distributed Processing (IPDPS 2013). Piscataway, NJ: IEEE, 2013: 29-40

[64] Ren Jinglei, Zhao Jishen, Khan S, et al. ThyNVM: Enabling software-transparent crash consistency in persistent memory systems [C] //Proc of the 48th Annual IEEE/ACM Int Symp on Microarchitecture (MICRO 2015). New York: ACM, 2015: 672-685

[65] Wei Xueliang, Feng Dan, Tong Wei, et al. NICO: Reducing software-transparent crash consistency cost for persistent memory [J/OL]. IEEE Transactions on Computers, 2018 [2019-01-20]. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8496805

[66] Hwang T, Jung J, Won Y. Heapo: Heap-based persistent object store [J]. ACM Transactions on Storage, 2015, 11 (1): 3-23

[67] Chakrabarti D R, Boehm H J, Bhandari K. Atlas: Leveraging locks for non-volatile memory consistency [J]. ACM SIGPLAN Notices, 2014, 49(10): 433-452

[68] Izraelevitz J, Kelly T, Kolli A. Failure-atomic persistent memory updates via JUSTDO logging [J]. ACM SIGARCH Computer Architecture News, 2016, 44(2): 427-442

[69] Joshi A, Nagarajan V, Viglas S, et al. ATOM: Atomic durability in non-volatile memory through hardware logging [C] //Proc of IEEE Int Symp on High Performance Computer Architecture (HPCA 2017). Piscataway, NJ: IEEE, 2017: 361-372

[70] Shin S, Tirukkovalluri S K, Tuck J, et al. Proteus: A flexible and fast software supported hardware logging approach for NVM [C] //Proc of the 50th Annual IEEE/ACM Int Symp on Microarchitecture (MICRO 2017). New York: ACM, 2017: 178-190

[71] Ogleari M A, Miller E L, Zhao Jishen. Steal but no force: Efficient hardware undo+redo logging for persistent memory systems [C] //Proc of IEEE Int Symp on High Performance Computer Architecture (HPCA 2018). Piscataway, NJ: IEEE, 2018: 336-349

[72] Zhan Jinyu, Zhang Yiming, Wei Jiang, et al. Energy-aware page replacement and consistency guarantee for hybrid NVM-DRAM memory systems [J]. Journal of Systems Architecture, 2018, 89: 60-72

[73] Lee E, Yoo S, Jang J E, et al. Shortcut-JFS: A write efficient journaling file system for phase change memory [C] //Proc of the 28th Symp on Mass Storage Systems and Technologies (MSST 2012). Piscataway, NJ: IEEE, 2012: 1-6

[74] Lee E, Jang J E, Kim T, et al. On-demand snapshot: An efficient versioning file system for phase-change memory [J]. IEEE Transactions on Knowledge and Data Engineering, 2013, 25(12): 2841-2853

Xiao Renzhi, born in 1989. PhD candidate. Student member of CCF. His main research interests include in-memory key-value store, non-volatile memory and NVM-based data structures.

Feng Dan, born in 1970. PhD, professor and PhD supervisor. Senior member of CCF. Her main research interests include computer architecture, massive storage systems, file system, and non-volatile memory.

Hu Yuchong, born in 1983. PhD, associate professor and PhD supervisor. His main research interests include cloud storage, heterogeneous storage, network coding, and erasure coding. (yuchonghu@hust.edu.cn)

Zhang Xiaoyi, born in 1989. PhD. His main research interests include NVM-based storage systems and NVM-based data structures. (zhangxiaoyi@hust.edu.cn)

Cheng Liangfeng, born in 1994. PhD candidate. His main research interests include in-memory key-value store and erasure code fault-tolerance. (lenfungcheng@hust.edu.cn)