计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2021.20200394
58(2):356-370,2021

基于持久化内存的索引设计重新思考与优化

韩书楷　熊子威　蒋德钧　熊　劲
(计算机体系结构国家重点实验室(中国科学院计算技术研究所)　北京　100190)
(中国科学院大学　北京　100049)
(hanshukai@ict.ac.cn)

Rethinking Index Design Based on Persistent Memory Device

Han Shukai, Xiong Ziwei, Jiang Dejun, and Xiong Jin
(*State Key Laboratory of Computer Architecture* (*Institude of Computing Technology*, *Chinese Academy of Sciences*), *Beijing* 100190)
(*University of Chinese Academy of Sciences*, *Beijing* 100049)

Abstract　NVM (non-volatile memory) is a new type of storage medium that has emerged in recent years. On the one hand, similar to DRAM (Dynamic RAM), NVM has low access latency and byte-addressable characteristics; on the other hand, it does not lose data after a power failure. Moreover, it has higher density and lower power consumption. The emergence of NVM provides new opportunities for improving indexing efficiency, and thus many works focus on building NVM-based indexing. However, these works are conducted based on simulated NVM devices. In April 2019, Intel released real NVM hardware AEP (apache pass) based on 3D-XPoint technology. The actual AEP devices are evaluated, and the results show that the write latency of AEP is close to that of DRAM, while the read latency is 3～4 times that of DRAM. Based on actual NVM hardware performance, we find that many past works have biased performance assumptions about NVM, which leaves some past works open to optimizing space. We then revisit previous persistent indexing works. We propose a read-optimized hybrid index (HybridIndex$^+$) and a hybrid-memory-based asynchronous caching approach for persistent index. Experimental results show that the read performance of HybridIndex$^+$ is 1.8 times that of existing hybrid index. The asynchronous cache-optimized indexes can reduce latency by up to 50%.

Key words　non-volatile memory; persistent memory; index; storage systems; key-value store

摘　要　非易失性内存(non-volatile memory, NVM)是近几年来出现的一种新型存储介质.一方面,同传统的易失性内存一样,它有着低访问延迟、可字节寻址的特性;另一方面,与易失性内存不同的是,掉电后它存储的数据不会丢失,此外它还有着更高的密度以及更低的能耗开销.这些特性使得非易失性内存有望被大规模应用在未来的计算机系统中.非易失性内存的出现为构建高效的持久化索引提供了

收稿日期:2020-06-08;**修回日期**:2020-10-20
基金项目:国家重点研发计划项目(2018YFB1003303);中国科学院战略性先导科技专项资助(XDB44030200);北京市自然科学基金-海淀原始创新联合基金项目(L192038);中国科学院青年创新促进会资助项目
This work was supported by the National Key Research and Development Program of China (2018YFB1003303), the Strategic Prionity Research Program of Chinese Academy of Sciences (XDB44030200), Beijing Natural Science Foundation-Haidian Joint Fund for Original Innovation (L192038), and Youth Innovation Promotion Association of the Chinese Academy of Sciences.
通信作者:蒋德钧(jiangdejun@ict.ac.cn)

新的思路.由于非易失性硬件还处于研究阶段,因此大多数面向非易失性内存的索引研究工作基于模拟环境开展.在2019年4月英特尔发布了基于3D-XPoint技术的非易失性内存硬件apache pass(AEP),这使得研究人员可以基于真实的硬件环境去进行相关研究工作.首先评测了真实的非易失性内存器件,结果显示AEP的写延迟接近DRAM,而读延迟是DRAM的3~4倍.基于对硬件的实际评测结果,研究发现过去很多工作对非易失性内存的性能假设存在偏差,这使得过去的一些工作大多只针对写性能进行优化,并没有针对读性能进行优化.因此,重新审视了之前研究工作,针对过去的混合索引工作进行了读优化.此外,还提出了一种基于混合内存的异步缓存方法.实验结果表明,经过异步缓存方法优化后的混合索引读性能是优化前的1.8倍,此外,经过异步缓存优化后的持久化索引最多可以降低50%的读延迟.

关键词 非易失性内存;持久化内存;索引;存储系统;键值存储系统

中图法分类号 TP391

非易失性内存(non-volatile memory, NVM)是近几年来出现的一类存储介质的统称,例如:PCM[1],ReRAM[2],STT-RAM[3]等.一方面,这些存储介质同DRAM(dynamic RAM)一样有着低访问延迟、可字节寻址的特性;另一方面,与DRAM不同的是,它们具有非易失性、较低的能耗和较高的存储密度.这使得基于NVM技术的非易失性内存有着更大的单片存储容量,同时能够作为存储设备保存数据.若利用基于非易失性内存构建存储系统,一方面,相对于传统基于内存的存储系统而言,可以受益于非易失性内存的非易失、大容量、低能耗特点[4];另一方面,相对于传统基于磁盘的存储系统而言,可以受益于更低的访问延迟以及更细粒度寻址方式.因此,NVM技术有望被大规模的应用在存储系统的研发与构建中,成为存储系统进一步发展的新机遇.

键值存储系统是数据中心中一类重要的基础性存储设施,例如内存键值存储系统Memcached[5],Redis[6]和磁盘键值存储系统LevelDB[7],RocksDB[8]等.无论是内存键值存储系统还是磁盘键值存储系统,索引结构都是这些存储系统中非常重要的基础技术,一直以来便是存储领域的热点研究问题.在之前的研究工作中,索引结构大多构建在DRAM中.随着低访问延迟、可字节寻址、可持久化数据的NVM的出现,这使得基于NVM构建持久化的高性能索引成为可能.近年来,很多面向NVM的索引研究工作[9-16]认为NVM有着同DRAM近似的读延迟以及高于DRAM数倍的写延迟的特性.因此,大部分研究工作提出通过降低NVM写开销从而构建不同的高效持久化索引.这些工作基本可以分成2类:一类是针对单一索引结构的设计优化.例如:Level Hashing[10]是面向NVM设计的Hash索引,它使用2层的Hash表结构以降低表拓展时搬运的数据总量.NV-Tree[12]是针对B^+树索引的优化,它不对同一叶子节点内的数据进行排序,从而减少了插入数据时的写开销,此外NV-Tree还将内部节点放在DRAM以降低树分裂时对NVM的写开销.另一类是面向DRAM-NVM混合内存结构的混合索引,例如,HiKV[15]是基于DRAM-NVM构建的混合索引键值存储系统,它对同一份数据同时维护Hash表和B^+树2种索引去保证高效的读性能,为了HiKV将写开销较低的Hash表放在的NVM而将写开销较高的B^+树放在DRAM中,从而降低对NVM写负担.

从2009年开始,NVM便被计算机系统领域广泛研究.然而,研究早期NVM技术还未成熟,尚未有成熟的NVM设备可供使用,上述大部分研究工作都是基于对非易失性内存器件的性能假设并使用模拟器进行实验评测[4-5].2019年4月,英特尔公司正式发布了基于3D-XPoint技术[17]的NVM硬件产品apache pass(AEP)[18],这为研究人员基于真实的NVM硬件进行研究提供了基础.目前,已有一些针对真实NVM硬件的评测工作显示,现有的NVM硬件有着接近DRAM的写延迟以及高于其数倍的读延迟.而之前研究工作所采用了如下性能假设:1)NVM有着同DRAM近似的读延迟,2)NVM有着高于DRAM数倍的写延迟.这些假设和现有真实NVM硬件性能评测结果并不完全相符.这使得我们需要重新审视之前基于NVM性能假设的索引研究工作,并基于实际的NVM硬件特性开展有效的优化工作.

本文的主要贡献有3个方面:

1) 对最新的 AEP 硬件进行了评测,我们评测了 AEP 硬件的不同访问线程、不同访问粒度下的读写延迟、带宽,以及读写混合下 AEP 的性能变化趋势.

2) 基于真实 AEP 硬件评测结果,我们分别重新审视了之前研究工作并针对混合索引结构和单一索引结构进行了优化.针对混合索引结构,本文基于 AEP 硬件真实性能,重点关注索引放置的优化.本文探索了不同索引放置方式对混合索引结构的影响,当面临读密集的负载时,通过将主索引放置在 DRAM,辅助索引放置在 AEP 上,从而可以有效提升索引的读性能.本文针对不同应用场景,提出索引放置相应的设计原则考虑.我们针对混合索引(HybridIndex)提出了读优化的改善方案 Hybrid-Index$^+$,该方案下 HybridIndex 最多可提升 80%的读性能.

3) 针对单一索引结构,本文基于 AEP 读延迟较高的特性,提出基于 DRAM 的异步缓存方法,将位于 NVM 中持久化索引通过高速 Hash 索引的方式缓存在 DRAM 中,从而获得高效的访问性能.此外,本文还针对 FP-Tree[13]、FAST-FAIR[11] 和持久化跳表[14]实现了异步缓存,评测结果显示经过我们的优化,最多可以降低持久化索引 20%~50%的读延迟.

1 非易失性内存简介

在实际的 NVM 硬件投放市场之前,学界对持久化内存的访存特性具有一些基本假设,比如:NVM 有着数倍于 DRAM 的写延迟及相近的读延迟.该假设是否成立尚需物理器件的验证.此外,非易失性内存的具体特性尚处于未知,如访问粒度对延迟带宽的影响、多线程对延迟带宽的影响、混合访存对延迟带宽的影响等.对 AEP 进行详细的测试可以为我们揭示一类非易失性内存的物理特性,为研究者和开发者在后续基于非易失性内存的工作中提供参考,也可为系统工作者构建包含非易失性内存的新型存储系统提供参考.本节主要介绍了非易失性内存的特性,并对当今最新的 NVM 硬件在各个维度进行了详细的评测.

1.1 非易失性内存技术

NVM 与传统 DRAM 一样,可以被挂载在 CPU 地址总线上按字节进行寻址.如表 1 所示,目前已有多种可用于生产持久化内存的存储介质.这些介质有着接近 DRAM 的纳秒级读写延迟以及更高的集成密度.此外,相比 DRAM 需要频繁的刷新去记录数据,非易失性内存介质不需要频繁的刷新去保证数据的有效性,这使其能耗相比 DRAM 更低.

Table 1 Comparison of the Properties of Different Storage Media[19]

表 1 不同存储介质的特性对比[19]

设备	存储单元尺寸/F^2	读延迟/ns	写延迟/ns	写寿命	写能耗	其他能耗
硬盘		5×10^6	5×10^6	$>10^{15}$	低	待机
DRAM	6~10	30	15	$>10^{15}$	低	刷新
SRAM	50~120	1~100	1~100	$>10^{15}$	低	漏电流
PCM	4~12	50	500	10^8	高	无
STT-RAM	6~50	10	50	10^{15}	低	无
ReRam	4~10	10	50	10^8	低	无

需要注意的是,表 1 仅为不同存储介质的性能对比而非实际的存储硬件性能.NVM 设备的复杂性要求我们进行细致的评测和分析以获得硬件的实际性能和详细特性.

1.2 Apache Pass

英特尔公司在 2019 年 4 月公布了第 1 代基于 3D-XPoint 技术[17]的商用持久化内存硬件 AEP[18],当前发布的 AEP 分别为 128GB/256GB/512GB 容量.这标志着持久化内存正式进入了实际可用的阶段.

2 AEP 硬件性能评测

目前存在着如 MLC(memory latency checker)[20]等内存检查的工具,但这些工具的高度定制化和闭源特性不符合我们对 AEP 进行细粒度测试的需求,因此我们编写了一系列测试代码①,这些测试代码主要完成对 AEP 硬件的延迟、带宽等基本性能的评测,探索 AEP 的物理特性,以揭示其带宽延迟特性,

① 已开源:https://github.com/KinderRiven/AEP_TEST

最佳访问模式等信息,为后续设计工作在 AEP 上的系统提供指导.

2.1 测试环境和方法

AEP 有着接近传统易失性内存的访问延迟,此外还有着传统持久化设备的持久化特性,因此它在传统存储层次架构中可以担任不同的角色.一方面作为内存而言,它可以作为 DRAM 的下层大容量存储介质;另一方面作为可持久化设备,它可以作为存储层中的高速上层存储介质,位于传统的高速 SSD 之上.

基于这 2 种使用方法,AEP 提供了 Memory Mode 和 App Direct Mode 这 2 种配置使用方法[21].若采用 Memory Mode 进行配置,AEP 将被当作大块内存而非持久化存储设备使用,因此对应用程序不可见;如果采用 App Direct Mode 进行配置,AEP 则被当作一块持久化设备使用,可以被应用程序直接看到并访问.需要注意的是.由于我们的研究工作主要将 AEP 作为高性能且持久化的存储设备而使用,因此本文所进行的评测都基于 Direct Mode 进行配置和使用,该模式可以有效反映 AEP 作为有效存储设备的性能.而对于 Memory Mode 模式下的性能评测,我们暂时没有进行.

表 2 展示了进行评测的物理环境.对于每组评测,我们均进行 5 次评测并在每组波动不超过 5% 的基础上取平均值.另外,为了避免 CPU 预取对测试结果造成影响,我们在测试中关闭了 CPU 预取.

Table 2 Experimental Environment

表 2 评测环境

项目	备注
CPU	Intel Xeon Gold 5215 CPU 2.5 GHz
Socket	2
物理核	20
一级指令高速缓存/KB	32
一级数据高速缓存/KB	32
二级指令高速缓存/KB	1 024
三级指令高速缓存/KB	14 080
DRAM 容量/GB	64
DCPMM 容量/GB	128
固态硬盘	Intel P3700 400 GB
文件系统	ext4
编译器	4.8.5 20150623(RedHat 4 4.8.5)(GCC)
编译选项	-O0
操作系统	CentOS Linux Release 7.6.8.10
内核版本	Linux 4.18.8

2.2 测试结果

2.2.1 延迟

表 3 展示了使用单线程对 AEP 进行小粒度随机和顺序读写时的延迟对比情况.可以看到,小于 256 B 的随机读延迟基本接近 256 B 的随机读延迟,这是由于 AEP 的访问粒度为 256 B.当访问粒度小于 256 B 时,顺序读相比随机读延迟要低很多,这是由于 AEP 的内部存在缓存机制,使得对同一访问单元(256 B 大小的物理块)进行连续访问时的性能得到提升.

Table 3 Comparison of Random and Sequential Read Latency of Small Access Granularity for AEP

表 3 单线程下 AEP 小粒度数据的随机和顺序读延迟对比

读粒度/B	延迟/ns		比值
	随机读	顺序读	
8	200	30	6.67
16	196	53	3.70
32	196	70	2.80
64	201	101	1.99
128	198	124	1.60
256	241	191	1.26
512	309	304	1.02
1 024	611	589	1.04

表 4 展示了使用单线程对 AEP 进行大粒度随机和顺序读时的延迟对比情况.可以看到,AEP 的随机和顺序读延迟基本一样,结合表 3 来看,在不考虑访问小粒度数据时内部缓存机制的影响,AEP 的随机和顺序读延迟基本一样.

Table 4 Comparison of Random and Sequential Read Latency of Large Access Granularity for AEP

表 4 单线程下 AEP 大粒度数据的随机和顺序读延迟对比

读粒度/KB	延迟		比值
	随机读	顺序读	
1	611 ns	589 ns	1.04
4	2 361 ns	2 326 ns	1.04
16	7 368 ns	7 273 ns	1.03
64	29 086 ns	28 850 ns	1.01
256	0.12 ms	0.12 ms	1.00
1 024	1.09 ms	1.09 ms	1.00
4 096	5.68 ms	5.68 ms	1.00
16 384	22.81 ms	22.82 ms	1.00

表 5 和表 6 展示使用单线程对不同粒度数据进行随机和顺序写的延迟对比,可以看到,不论在大小粒度下,AEP 的随机和顺序写延迟都是一样的.同

读访问一样,AEP 有着较好的随机访问特性.需要注意的是,在进行小于 64 B 数据的访问时,出现了延迟上升的情况,这主要是由于我们使用了 NTStore 指令进行写操作,该类指令可以不经 CPU cache 而将数据直接写入内中,而 NTStore 指令在处理小于 64 B 的数据时存在延迟升高的问题.

Table 5 Comparison of Random and Sequential Write Latency of Small Access Granularity for AEP

表 5 单线程下 AEP 小粒度数据的随机和顺序写延迟对比

写粒度/B	延迟/ns		比值
	随机写	顺序写	
8	658	694	0.95
16	669	717	0.93
32	663	774	0.86
64	190	187	1.02
128	203	199	1.02
256	218	217	1.00
512	279	259	1.08
1 024	538	503	1.07

Table 6 Comparison of Random and Sequential Write Latency of Large Access Granularity for AEP

表 6 单线程下 AEP 大粒度数据的随机和顺序写延迟对比

读粒度/KB	延迟		比值
	随机写	顺序写	
1	538 ns	503 ns	1.07
4	2 006 ns	2 004 ns	1.00
16	8 000 ns	8 089 ns	0.99
64	31 597 ns	32 115 ns	0.98
256	0.13 ms	0.13 ms	1.00
1 024	0.51 ms	0.51 ms	1.00
4 096	2.61 ms	2.60 ms	1.00
16 384	10.37 ms	10.50 ms	0.99

综上所述,AEP 有着良好的随机访问特性,最佳访问粒度为 256 B.在处理 1 KB 及以下的数据时,AEP 可以保证纳秒级别的访问延迟.

图 1 为 AEP 的在不同线程、不同粒度随机读延迟变化趋势图.

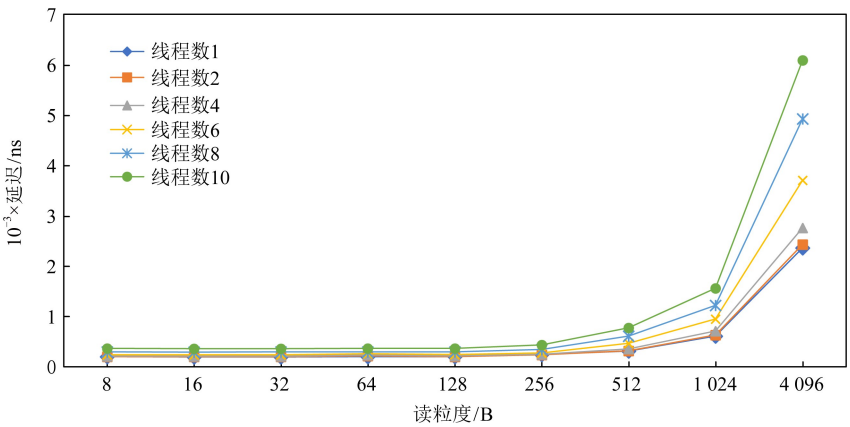


Fig. 1 Trend of random read latency of AEP over access granularity

图 1 AEP 的随机读延迟随访问粒度增加的变化趋势

我们注意到,在 256 B 以内的随机读延迟并不显著受到线程数的影响.但可以看出线程数的增加导致了延迟的升高,这个趋势在读粒度较大、延迟较高的情况下更为明显.在顺序读的测试中,AEP 也具有相似的趋势,读粒度小于 256 B 时延迟受线程数影响较小,且读延迟随线程数增加而升高.总结如下:

1) 多线程写操作会导致 AEP 读延迟的升高;

2) 为了维持较低的读延迟,小于等于 256 B 的读粒度是较优的读粒度.

图 2 是 AEP 的随机写延迟变化趋势图.与读延迟类似的是,写粒度小于 256 B 时 AEP 维持较低的延迟,且对线程数的增加更为敏感.且当写延迟在访问粒度接近 256 B 时取得最小值.总结如下:

1) 多线程读操作会导致 AEP 写延迟的急剧升高;

2) 接近 256 B 的写粒度是较优的写粒度;

3) 从延迟的角度来看,读写的最佳粒度是 256 B,且访问线程数不应太多.

2.2.2 带宽

图 3 和图 4 展示了多线程下不同粒度下随机读写带宽变化情况,可以看出,AEP 最大读带宽为

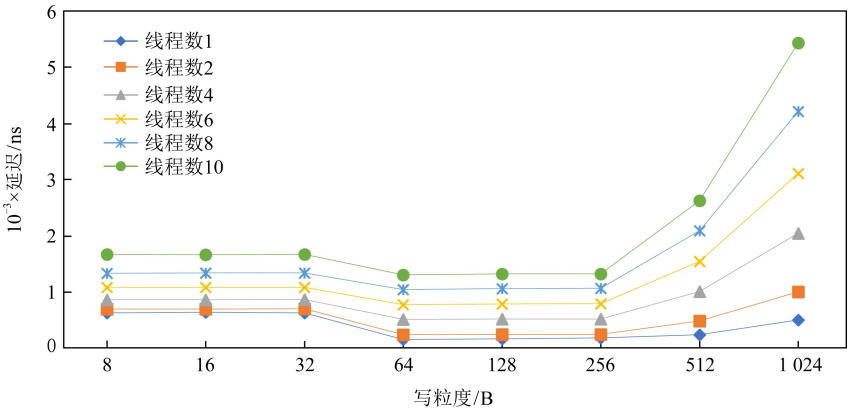


Fig. 2 Trend of random write latency of AEP over access granularity

图 2 AEP 的随机写延迟随访问粒度增加的变化趋势

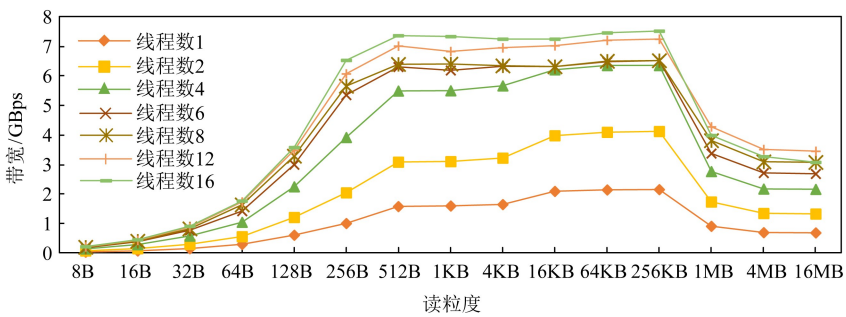


Fig. 3 Trend of random read bandwidth of AEP over access granularity

图 3 AEP 随机读带宽随访问粒度增加变化趋势

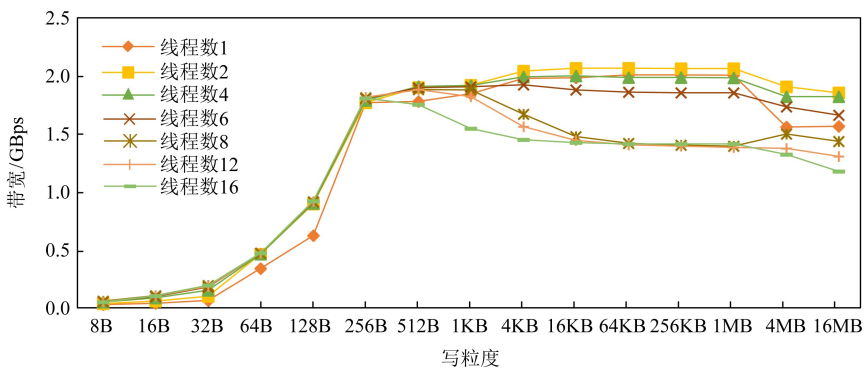


Fig. 4 Trend of random write bandwidth of AEP over access granularity

图 4 AEP 随机写带宽随访问粒度增加变化趋势

7 GBps 左右,最大写带宽为 2 GBps.且在多线程下,256 B 的访问粒度下读写带宽基本可以达到最大带宽上限.这主要是由于 AEP 的访问粒度为 256 B 所导致的,小于 256 B 的访问会造成读写放大从而降低有效带宽.

此外,我们从图 3 和图 4 中发现,若访问粒度超过一定阈值,AEP 的读写带宽反而出现了下降.这是由于对于读带宽,cache 的频繁替换造成了性能的损失.而对于写带宽,我们认为是 CPU 乱序导致写操作被拆分为 64 B 粒度,由于 AEP 写带宽较低并受 256 B 写粒度的影响,当数据被拆分成 64 B 的随机写后乱序写回,这造成了数据的写放大从而引起带宽下降.为了验证我们的判断,将一个数据块写回时,每隔 256 B 添加一次内存屏障(sfence),确保数据的有序写回.如图 5 所示,添加内存屏障后 AEP 的写带宽没有出现下降的情况.

图 6 和图 7 展示了多线程下不同粒度下顺序读写带宽变化情况.可以看出,相比随机访问需要在 256 B 达到最大读写带宽,顺序访问在 64 B 的访问粒度时即可达到最大读写带宽.对于读操作而言,这是由于 AEP 内部的缓存机制使得小于 256 B 的顺序访问可以有效命中缓存,从而降低读放大减少带宽浪费;对于写而言,这主要是由于 AEP 内部会对连续地址的请求进行合并,这使得 AEP 在顺序写入 64 B 的数据时能够将其合并成一个 256 B 单位的数据写入硬件中,从而避免了写放大问题.此外,我们依然可以看出,在大粒度的顺序访问时读写带宽依然出现了退化的情况.

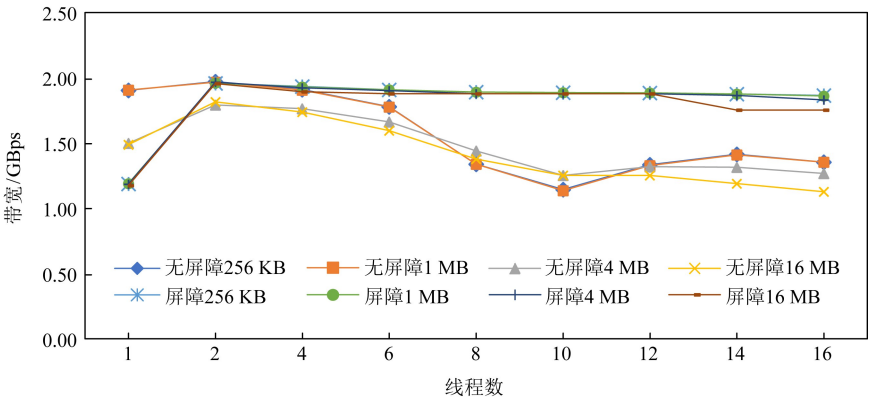


Fig. 5 Impact of using memory barriers on write bandwidth degradation

图 5 内存屏障对写带宽退化的影响

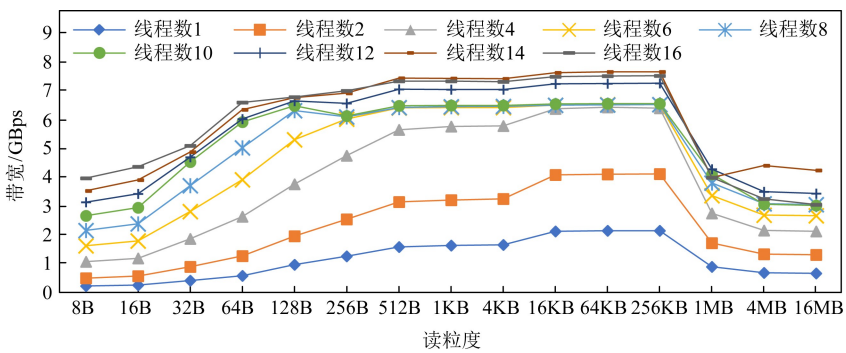


Fig. 6 Trend of sequential read bandwidth of AEP over access granularity

图 6 AEP 顺序读带宽随访问粒度增加变化趋势

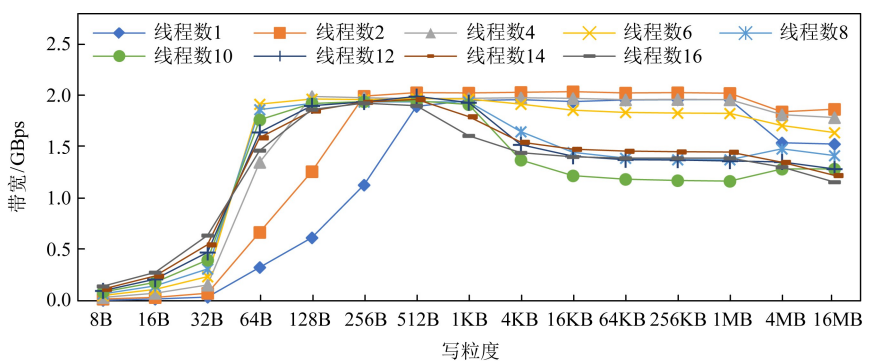


Fig. 7 Trend of sequential write bandwidth of AEP over access granularity

图 7 AEP 顺序写带宽随访问粒度增加变化趋势

经过评测有如下结论:

1) 当 AEP 进行大粒度数据访问时,会出现带宽退化的现象,对于读带宽的带宽,可以利用内存屏障进行写序控制从而防止退化.

2) AEP 更加适合处理小粒度数据,考虑到其访问粒度为 256 B,因此处理更小粒度的数据时可能会造成读写放大,从而浪费带宽.

3) 尽管受制于 256 B 的访问粒度,但 AEP 内部仍存在某些顺序访问的优化机制.

2.2.3 与 DRAM 和 SSD 的比较

在延迟方面,与 DRAM 的读延迟对比如表 7 所示,写延迟对比如表 8 所示.

Table 7 Latencies of AEP and DRAM Random Read with Single Thread for Small Access Granularity

表 7 单线程下 AEP 和 DRAM 的小粒度随机读延迟

读粒度/B	随机读延迟/ns		比值
	AEP	DRAM	
8	200	46	4.35
16	196	33	5.94
32	196	33	5.94
64	201	40	5.03
128	198	43	4.60
256	241	57	4.23
512	309	93	3.32
1 024	611	173	3.53

Table 8 Latencies of AEP and DRAM Random Write with Single Thread for Small Access Granularity

表 8 单线程下 AEP 和 DRAM 的小粒度随机写延迟

写粒度/KB	随机写延迟/ns		比值
	AEP	DRAM	
8	658	238	2.76
16	669	238	2.81
32	663	239	2.77
64	190	174	1.09
128	203	185	1.10
256	218	200	1.09
512	279	234	1.19
1 024	538	361	1.49

从表 7~8 中可以看出,读延迟上 AEP 为 DRAM 的 4~5 倍,之前关于“NVM 比 DRAM 有着更高的写延迟和一样的读延迟”的假设与实际器件是不符合的.而写延迟,DRAM 和 AEP 都表现出了非对称的写延迟,但 AEP 的写延迟在写粒度接近 256 B 时与 DRAM 相差不大.因此对非易失性内存的假设应当修正为:对于采用与 AEP 相似技术的 NVM,其整体具有比 DRAM 更高的读延迟以及与 DRAM 相近的写延迟.此外,在带宽方面,DRAM 具有高达 10 GBps 以上的读写带宽,而 AEP 读带宽为 7 GBps,写带宽仅 2 GBps.因此 AEP 目前还不具备完全代替 DRAM 的潜力.

尽管表现逊色于 DRAM,AEP 的性能远远优于目前已有的 SSD 产品.我们使用 fio 测试了传统 NVMe SSD 的性能,可以发现系统内 SSD 的峰值读带宽为 870 MBps,远远低于 AEP.尽管目前已有较高端的 SSD 产品达到了 2 GBps 的读带宽,但对比 AEP 高达 7 GBps 的读带宽,仍旧显得逊色.而写带宽上系统内 SSD 的峰值写带宽仅 680 MBps,也远低于 AEP 高达 2 GBps 的带宽.因此我们有如下观察:

1) AEP 在 256 B 访问下具有较优异的读写带宽和延迟表现;

2) AEP 并不是一个高并发友好的器件,访问 AEP 的线程数应控制在较低的数量;

3) AEP 目前暂不具备完全代替 DRAM 的潜力;

4) AEP 的性能表现远远好于 SSD,在现有的存储层次中,可作为 DRAM 于 SSD 甚至 HDD 之间的新层级.

如 2.2.2 节所述,访问粒度和线程数对 AEP 的性能表现存在较大影响,接下来将分别说明粒度和线程数对 AEP 性能表现的具体影响.

2.2.4 访问粒度的影响

2.2.2 节已粗略展示了随着访问粒度的升高,AEP 出现带宽下降、延迟升高的性能退化问题,本节将以单线程的结果,更详细分析访问粒度对 AEP 性能的影响.由于带宽的下降是有限的,因此本节不讨论 AEP 带宽的变化.

从图 8 和图 9 中可以看出,尽管较小粒度(小于 512 B)时 AEP 展现了百纳秒级别的优秀表现,但当访问粒度上升至 KB 级别,AEP 延迟开始上升到了微秒级别,而如果采用 MB 基本的访问,AEP 的延迟飙升至毫秒级别,逼近了 SSD 的延迟.随机写具有类似的现象.因此可见,AEP 是一个小粒度友好的设备,在设计工作在 AEP 上的系统时,应当尽量避免过大粒度的访问.

2.2.5 访问线程数的影响

本节同样聚焦于线程数对随机读延迟的影响.如图 10 所示,在小粒度的情况下(小于 256 B),AEP 的延迟上升不明显.线程数从 1 增加 15 倍到 16 之后,延迟并未上升 15 倍,而仅升高了 2 倍左右.但在大粒度的情况下,延迟基本随线程数线性增长.这进一步说明在并发的情景下,AEP 也仍然是一个小粒度访问友好的设备.较小的访问粒度可以避免高并发下延迟陡增的情况.随机写也具有类似的现象.

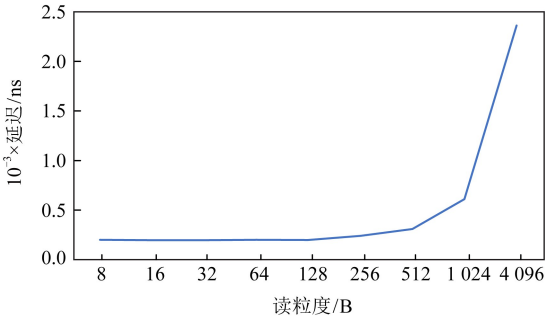


Fig. 8 The impact caused by small access granularity to the read latency of AEP with single thread

图 8 单线程下小粒度访问对 AEP 随机读延迟的影响

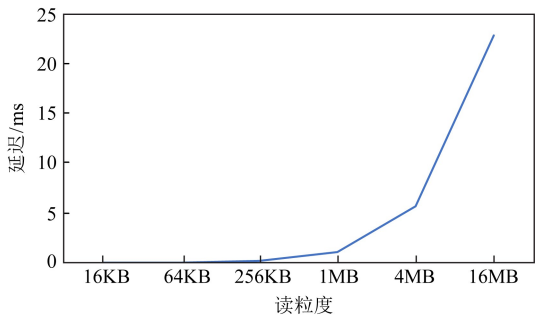


Fig.9 The impact caused by large access granularity to the read latency of AEP with single thread

图 9 单线程下大粒度访问对 AEP 随机读延迟的影响

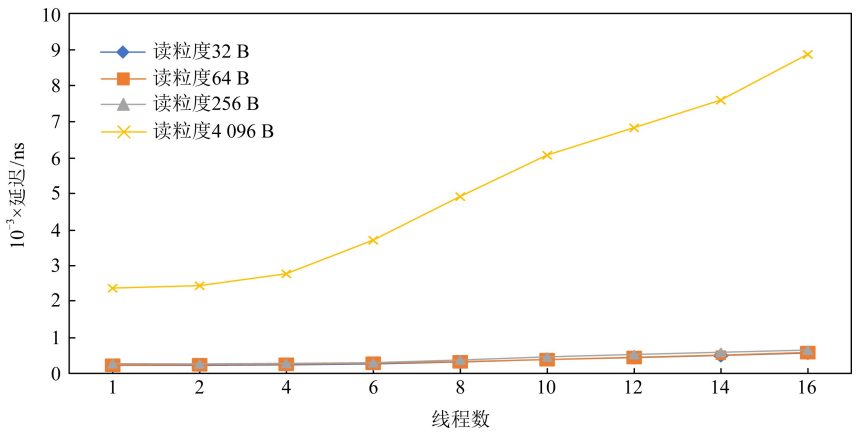


Fig. 10 The impact caused by number of thread to the read latency of AEP

图 10 不同线程数对 AEP 读延迟的影响

2.2.6 混合读写的影响

我们评测了读写混合下 AEP 硬件带宽和延迟的变化趋势,如图 11 所示.我们注意到若固定读线程数,一旦出现写线程,则读取带宽出现急剧下降的情况,但这种下降不是无限制的,随着写线程的不断增加,读带宽下降变得不再明显.这表明 AEP 内部存在一定的读保护机制,阻止读带宽受到写带宽的过度影响.如图 12 所示,反之若固定写线程,增加读线程数,写带宽也同样出现下降的情况,但最终也逐渐趋于平缓不再下降,.综上表现来看,AEP 内部对于

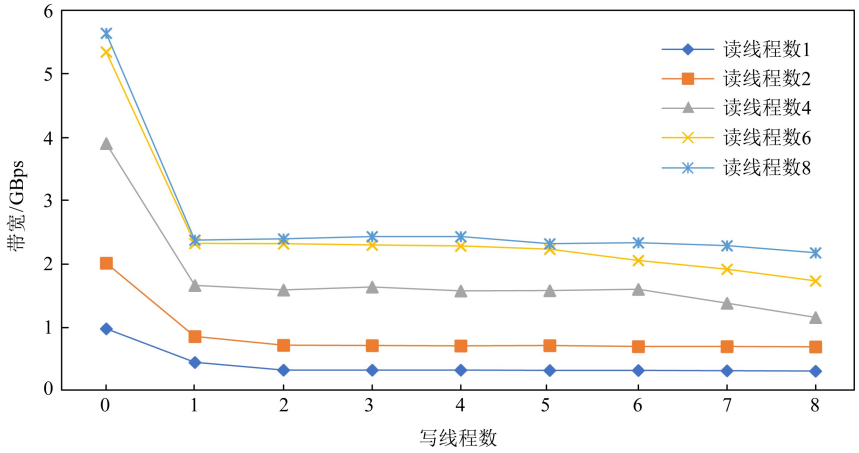


Fig. 11 Read bandwidth variation of AEP under mixed read-write at granularity 256 B

图 11 256 B 粒度混合读写下的 AEP 读带宽变化情况

读写带宽的控制存在一定的“保底”,从而避免了高并发情况下出现读写带宽不平衡的情况.

在延迟方面,AEP 也呈现出类似的趋势,从图 13 和图 14 中可以看出,AEP 内部的读保护机制也阻止了读写延迟受线程影响也发生过大的升高,保证了读写延迟稳定在 1 μs 左右.

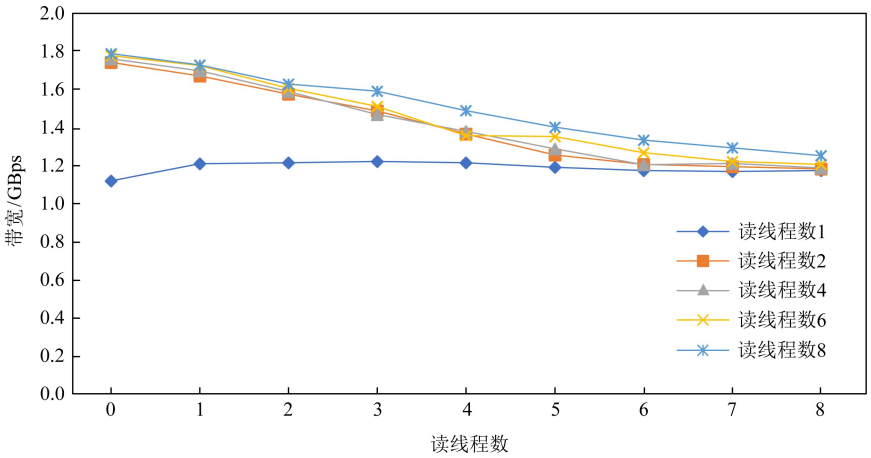


Fig. 12 Write bandwidth variation of AEP under mixed read-write at granularity 256 B

图 12 256 B 粒度混合读写下的 AEP 写带宽变化情况

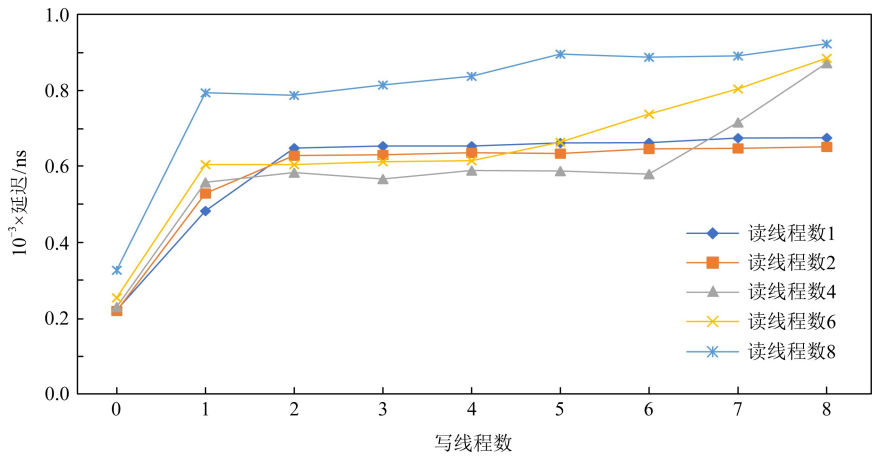


Fig. 13 Read latency variation of AEP under mixed read-write at granularity 256 B

图 13 256 B 粒度混合读写下 AEP 读延迟的变化情况

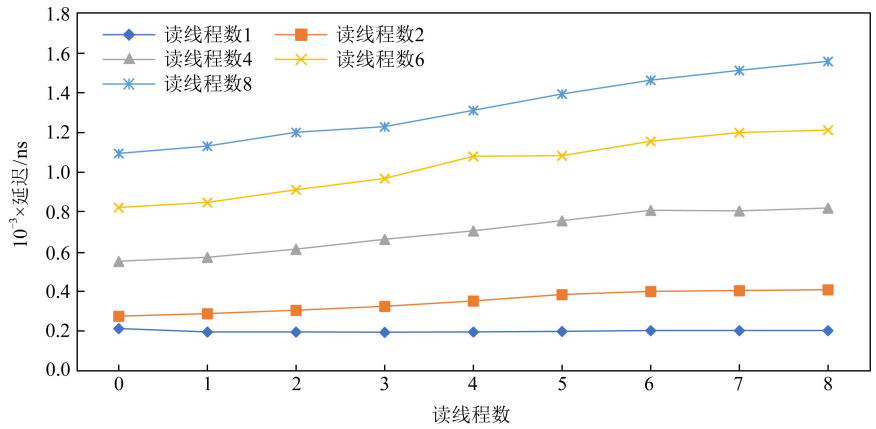


Fig. 14 Write latency variation of AEP under mixed read-write at granularity 256 B

图 14 256 B 粒度混合读写下的 AEP 写延迟变化情况

2.3 评测总结

针对 AEP 的评测可总结出以下的结论：

1) AEP 访问粒度为 256 B，其内部存在缓存机制，从而可对于小粒度的顺序访问进行优化.但对小粒度的随机访问该优化通常不奏效，因而对 AEP 的小粒度随机访问会造成读写放大问题.在进行大粒度数据访问时，AEP 存在带宽退化的问题且性能较差.综上所述，AEP 更加适合用于存储小粒度的数据，特别是 256 B 到 4 KB 之间大小的数据.

2) 在带宽方面，AEP 读写带宽较低且不平衡.AEP 的读带宽大约为 7 GBps，仅为 DRAM 的 1/3 左右；而写带宽更低约为 2 GBps，仅为 DRAM 的 1/6 左右.

3) 在延迟方面，AEP 顺序随机访问性能持平，作为一种随机存储设备是合格的.它的写延迟接近 DRAM，而读延迟为 DRAM 的 3～5 倍.

4) 相比其较低的带宽，AEP 的延迟显得更为优秀.因而 AEP 更适合作为低延迟的响应设备，用于存储对延迟需求较高且需要持久化的数据，比如索引、日志等重要的元数据.此外，同之前工作主要聚焦于 NVM 高写延迟上不同的是，经评测由于 AEP 的读延迟高于 DRAM，面向 AEP 的设计更应该关注高读延迟所带来的问题并进行优化.

3 基于非易失性内存的索引设计

近几年来，关于如何使用 NVM 一直是存储系统领域的热点研究问题.一方面，NVM 有着接近易失性内存的纳秒级访问延迟，以及可字节寻址的特性；另一方面，NVM 相比易失性内存有着更大的容量以及非易失性的特性，此外尽管 NVM 有着较低的访问延迟，但受制于介质特性，其拥有的带宽相比 DRAM 而言并不高.因此有很多工作考虑在 NVM 中构建高效的持久化索引[9-16].

然而，在过去几年里由于 NVM 硬件尚未商业化，很多工作都是基于模拟环境进行；此外，过去的针对 NVM 性能的假设也存在一些偏差，比如过去的很多模拟器大多模拟 NVM 比易失性内存写延迟高 5 倍，而读延迟同易失性内存一样，这同我们对 AEP 实际硬件的评测结果并不一致.因此，上述原因使得过去的很多工作的设计和评测存在一些局限性.

在本节，我们针对实际的 AEP 硬件重新优化并评测了过去针对 NVM 模拟器进行设计的索引，通过 2 个实例分析证明我们工作的有效性.

3.1 案例分析：面向混合索引的性能优化

混合索引键值存储系统(hybrid index key-value store, HiKV)[15]是面向混合内存架构的一种高性能键值存储系统.由于不支持范围查询的 Hash 索引的查询复杂度为 $O(1)$，而 B^+ 树等支持范围查询的数据结构查询复杂度为 $O(\log n)$，因此，该系统中提出了面向 DRAM-NVM 混合内存架构的混合索引(HybridIndex)机制.具体而言，HybridIndex 采用 Hash 表和 B^+ 树组成的混合索引机制来支持高效的查询.

如图 15 所示，HybridIndex 对同一份数据维护 2 个索引，HybridIndex 使用 Hash 索引进行单点查询，使用 B^+ 树进行范围查询.为了保证索引的持久化，HybridIndex 将 Hash 索引放在 NVM 而将 B^+ 树索引放在易失性内存，宕机之后根据 NVM 中的持久化 Hash 索引重建易失性内存中的 B^+ 树.此外，为了降低写延迟，HybridIndex 只对 NVM 中的 Hash 索引进行持久化更新，这种设计主要是针对当时 NVM 普遍被认为写延迟高于 DRAM 而读延迟和 DRAM 近似的情况，而随着实际的 NVM 硬件出现，这种设计不再合理.经过实际评测，NVM 的写延迟同易失性内存近似，而 NVM 的读延迟会比易失性内存高 3 倍.因此，受制于 NVM 的高读延迟，这使得置于 NVM 的 Hash 索引无法提供高效的单点查询.由于在很多实际负载中的查询操作大部分为单点查询，且查询操作在很多负载中为主要的负载占比，因此主要面向查询优化而设计的 HybridIndex 无法再提供高效的系统吞吐量.

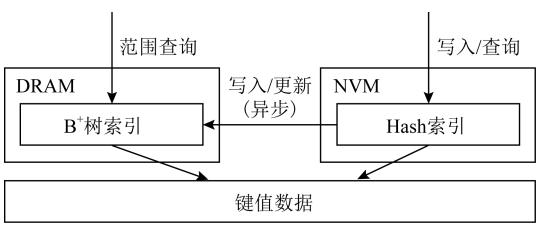


Fig. 15 HybridIndex system architecture

图 15 HybridIndex 系统架构

本文对 HybridIndex 面向实际的 NVM 硬件进行了优化并提出了 HybridIndex+.如图 16 所示，HybridIndex+ 对换了 Hash 索引和 B^+ 树索引的位置，将 Hash 索引放在了易失性内存，将 B^+ 树索引放在了 NVM，并采用同步更新的方法同时更新 2 个索引.HybridIndex+ 采用了 FAST-FAIR[11] 作为持久化 B^+ 树，在进行宕机恢复时，可以瞬间恢复

B^+树,之后异步地重建 DRAM 中的 Hash 索引. HybridIndex$^+$的设计优势为:1)将 Hash 索引放在读延迟更低的易失性内存有利于降低读开销,而B^+树由于只提供范围查询服务,因此在查询B^+树通常会查询整个叶子节点,可以有效地利用 NVM 较大读粒度的特点,不会造成读放大问题;2)在进行宕机恢复时,HybridIndex$^+$在恢复完B^+树后,HybridIndex$^+$便可以正常提供服务,而易失性内存中 Hash 索引的恢复会放在后台异步进行.在这个过程中,B^+树即可提供单点查询又可提供范围查询,当 Hash 索引恢复完成后,Hash 索引便可提供更高效的单点查询服务.相比 HybridIndex 需要等待B^+树全部恢复完成才可提供完整的查询服务,HybridIndex$^+$有着更低的宕机恢复时间.

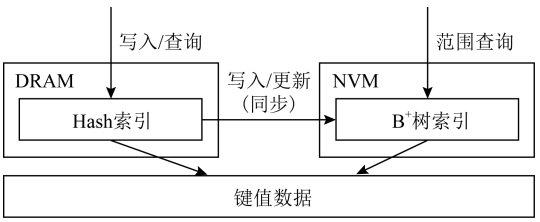


Fig. 16 HybridIndex$^+$ system architecture

图 16 HybridIndex$^+$系统架构

评测结果如图 17 所示,经过优化后,HybridIndex$^+$的读性能最多可提升至 HybridIndex 的 1.8 倍.但是由于采用了同步更新方法,HybridIndex$^+$的写性能相比 HybridIndex 降低了 30%.此外,由于 HybridIndex$^+$大部分读发生在 DRAM 中,而 HybridIndex 的读均发生在 NVM 中,由于 NVM 的读带宽相比 DRAM 较差,因此随着线程的不断增加,HybridIndex 的性能与 HybridIndex$^+$的性能差距逐渐增大.

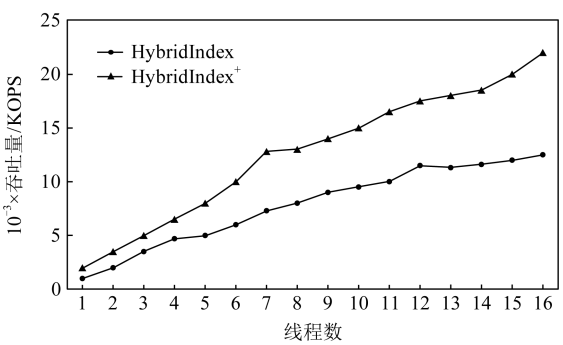


Fig. 17 Read performance comparison of HybridIndex and HybridIndex$^+$

图 17 HybridIndex 和 HybridIndex$^+$读性能对比

讨论:在上述评测中,HybridIndex$^+$考虑采用同步方式修改 Hash 表,这是因为若采用异步方式修改,可能存在查询 Hash 表时该更新还在异步队列中,这使得该查询需要等待队列中的更新完成后才能得到有效的数据,这将增加读请求的延迟.因此,对于写占比较大的负载场景,更加适合使用 HybridIndex 来降低写延迟,而对于读倾斜较高的负载场景,使用 HybridIndex$^+$能获得更低的读取延迟.此外,也可以根据实时的负载压力动态调节混合索引的同步异步更新方式.比如,可以在写负载较高而读负载较低时采用异步写的方法去降低写延迟,而当读负载较高写负载较低时可以采用同步写的方式去降低读延迟.

3.2 案例分析:基于持久化内存的异步缓存方法

在过去的几年里,很多研究普遍认为 NVM 的写延迟较高的问题[9-16],很多工作针对 NVM 设计了各种写优化索引,它们大多针对 NVM 设计了精细的数据结构从而减少了在索引持久化过程中的写开销.这些优化在现在看来依然是有效的,但过去的工作大多没有针对 NVM 硬件高读延迟的问题进行优化,因此,我们提出了一种基于持久化索引的异步缓存方法(asynchronous cache scheme, ACS)去优化现有持久化索引.

ACS 的设计思路是在不改变现有的持久化工作基础上,在易失性内存中缓存部分索引去提高 NVM 中索引的读性能.如图 18 所示,ACS 在易失性内存中维护了一个 Hash 索引去提供高效的单点查询.当进行一次查询时,首先需要搜索易失性内存中的 Hash 表,若该查询命中,则返回结果;若该查询未命中,则继续搜索 NVM 中的持久化索引,若命中持久化索引则返回结果,并将该索引异步更新到易失性内存中的读缓存中.需要注意的是,易失性内存中的 Hash 索引仅仅是 NVM 中索引的部分缓存,用户可以根据自身需求配置其大小,当它可缓存的索引项饱和从而无法再进行写入时,根据替换策略

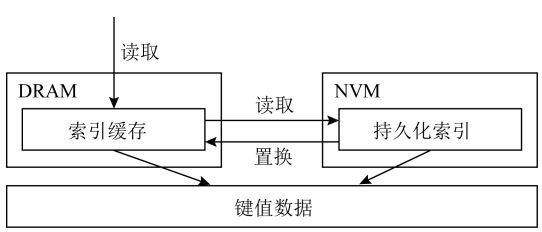


Fig. 18 Asynchronous cache scheme based on NVM

图 18 基于持久化内存的异步缓存方法

替换其中的索引项,在我们的实现中采用了 LRU 算法.此外,为了保证易失性内存和 NVM 中索引的同步,对于更新/删除操作,ACS 采用了同步更新策略,对易失性内存和 NVM 同时存在的索引项进行同步更新,从而避免了数据不一致性的问题.

为了验证有效性,我们使用该方法对一些过去的持久化索引的研究工作进行了优化.在这里我们选择了基于 B^+ 树实现的持久化索引 FAST-FAIR[11]、FP-Tree[13] 以及持久化跳表[14] 作为优化对象.FAST-FAIR 是在 FAST 会议上发表的基于 B^+ 树的持久化索引工作,它利用 FAST(failure-atomic shift) 和 FAIR(failure-atomic in-place rebalance)算法去实现索引的持久化并保证高性能.FP-Tree 也是基于 B^+ 树的持久化索引工作,为了降低 B^+ 树分裂时的写开销,FP-Tree 只将叶子节点保存在 NVM 并将内部节点保存在 DRAM,为了降低数据插入叶子节点的开销,FP-Tree 不对同一叶子节点内的数据进行排序.跳表作为一种简单且高效的索引结构被广泛应用在各种键值数据库中,如 LevelDB[7] 和 RocksDB[8] 都将跳表作为内部数据的索引结构,自 NVM 出现以来也有工作讨论基于 NVM 的持久化跳表实现.

在为索引加载了 5 000 万个索引项后,我们分别使用 YCSB[22] 中的均匀负载以及非均匀负载(负载分布为 zipfan)进行了读评测.在评测中,将缓存的容量设置为最多可容纳总数据量 10% 的容量,评测结果如图 19 所示.在均匀的负载情况下,由于缓存命中率较低从而导致一次搜索需要同时搜索缓存以及持久化索引,这使得 ACS 优化下的索引读性能并没有的提升;而在 zipfan 的负载情况下,缓存的命中率较高,因此一次搜索只会搜索缓存而不会搜索索引,因此经过 ACS 优化后的索引最多可以降低

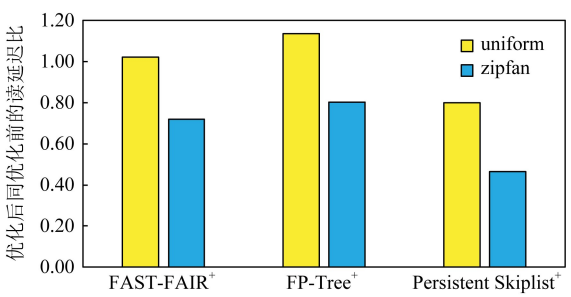


Fig. 19 Index read latency comparison after using ACS optimization

图 19 经过 ACS 优化后的索引读延迟对比

40% 的读延迟.可以看到,被优化索引的性能越差,通过 ACS 优化后所取得的收益越高.需要注意的是,对于插入操作,ACS 不会影响索引的性能,但对于更新操作,ACS 为了保证缓存和实际索引的一致性,需要在更新时对缓存进行修改,因此使用了 ACS 优化后的索引更新性能会下降 20%~30%.

讨论:ACS 可以在 zipfan 负载下取得较高的缓存命中率,因此可以获得较好的读性能,然而在面向 uniform 的负载倾斜时面临着读延迟升高的情况.这主要是由于 ACS 的搜索为串行执行,当搜索 DRAM 中缓存不成功时,还需要继续搜索位于 NVM 的持久化索引.为了解决该问题,我们可以使用并行搜索方法去降低请求延迟.如图 20 所示,当进行一次搜索时,可以在搜索 DRAM 中缓存的同时让后台线程搜索位于 NVM 的持久化索引,从而避免当缓存未命中时而导致的高读延迟.对于更新操作同样可以使用该方法进行优化.

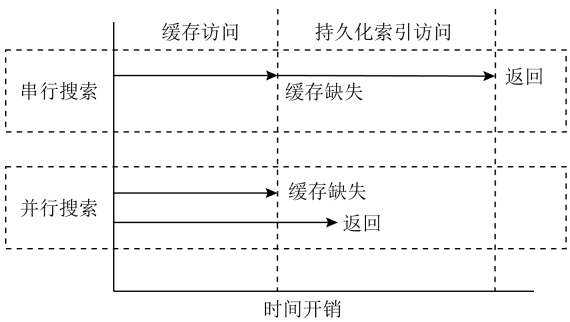


Fig. 20 Comparison of parallel/serial access time process based on ACS

图 20 基于 ACS 的并行/串行访问时间过程对比

实验表明,经过并行搜索优化后的 ACS,在缓存未命中情况下其读性能和更新性能不会下降.然而,进行并发搜索会消耗额外的 CPU 和带宽资源,因此在设计系统时可以根据缓存命中率决定是否启用并行执行,当缓存命中率较低时可以开启并行访问模式,反之当缓存命中率较高时可以只开启串行模式避免额外的系统资源消耗.

4 相关工作

目前已有其他 AEP 的测试工作,在文献[23]中,作者测试了 AEP 的带宽和延迟数据和 AEP 在若干系统中的表现(如 Memcached, RocksDB).但该工作主要聚焦于 AEP 在系统内的表现,忽视了

AEP本身的物理特性,未能挖掘出AEP的详细访问特性.而在文献[24]中,除AEP的带宽延迟信息外,作者进一步探索了AEP的尾延迟特性和多线程下的表现,分析了AEP的多线程访问中性能下降的问题,但并发提及混合读写下AEP的读写保护机制.

在过去的几年里,有许多工作基于NVM去设计高性能索引.Level Hashing[10]采用了2层Hash结构设计,在Hash表拓展时只进行一层结构的重构,从而减少了Hash拓展时的写开销;CCEH[9]采用段结构来降低Hash表拓展时的写开销;NV-Tree[12]针对B^+树进行优化,它将内部节点保存在易失性内存中,将叶子节点保存在NVM中,从而减少了树分裂时对NVM设备被造成高昂写开销;FP-Tree[13]基于NV-Tree基础上设计了简单高效的并发算法并利用signature槽优化了访问叶子节点时开销;FAST-FAIR[11]设计了一种高效的持久化B^+树算法;WORT[16]针对字典树在NVM中进行了写优化.

这些工作大多面向NVM高昂的持久化开销来进行优化,本文的工作主要面向非易失性的高读延迟进行优化,在保留过去研究设计的基础上设计了面向混合内存架构的异步缓存方法,在高倾斜的读负载情况,我们的优化能降低不同索引20%~40%的读延迟.

5 总 结

本文对实际的非易失性内存器件AEP进行了评测.基于评测结果发现,AEP的硬件性能同之前的假设有所不同:AEP同DRAM相比有着更高的写延迟以及更低的读延迟.为此,本文重新审视了之前的工作.一方面,针对过去的混合索引研究工作,本文提出对换索引位置提高索引的搜索性能,实验表明,HybridIndex$^+$相比HybridIndex可提升80%的读性能;另一方面,不改变原索引结构的基础上,本文提出了在DRAM中建立高速缓存的方法去加速AEP中的持久化索引搜索.实验表明,本文提出的优化方法最多可以降低50%的读延迟.在本文中主要讨论了在不改变原索引设计的基础上如何基于混合内存架构去提升原索引性能.然而,根据硬件评测结果显示,AEP作为可字节寻址设备依然与DRAM有着不小的区别,比如AEP有着更低的读写带宽以及更大的访问粒度,这也为日后的持久化索引研究工作提供了更多挑战.

参 考 文 献

[1] Noé P, Vallée C, Hibbert F, et al. Phase-change materials for non-volatile memory devices: From technological challenges to materials science issues [J]. Semiconductor Science and Technology, 2018, 33(1): article id:013002

[2] Chang Mengfan, Lee A, Lin C, et al. Read circuits for resistive memory (ReRAM) and memristor-based nonvolatile Logics [C] //Proc of the 20th Asia and South Pacific Design Automation Conf. Piscataway, NJ: IEEE, 2015: 569-574

[3] Li Hai, Chen Yiran. Emerging non-volatile memory technologies: From materials, to device, circuit, and architecture [C] //Proc of the 53rd IEEE Int Midwest Symp on Circuits and Systems. Piscataway, NJ: IEEE, 2010: 1-4

[4] Malladi K T, Nothaft F A, Periyathambi K, et al. Towards energy-proportional datacenter memory with mobile DRAM [C] //Proc of the 39th Int Symp on Computer Architecture. New York: ACM, 2012: 37-48

[5] Brad Fitzpatrick. Memcached [CP/OL]. [2020-03-18]. https://memcached.org/

[6] Salvatore Sanfilippo. Redis [CP/OL]. [2020-04-15]. https://redis.io/

[7] Google. LevelDB [CP/OL]. [2020-02-04]. https://github.com/google/leveldb

[8] Facebook. RocksDB [CP/OL]. [2020-02-04]. https://rocksdb.org/

[9] Moohyeon N, Hokeun C, Young-ri C, et al. Write-optimized dynamic hashing for persistent memory [C] //Proc of the 17th USENIX Conf on File and Storage Technologies Berkeley, CA: USENIX Association, 2019: 31-44

[10] Zuo Pengfei, Hua Yu, Wu Jie. Write-optimized and high-performance hashing index scheme for persistent memory [C] //Proc of the 13th USENIX Symp on Operating Systems Design and Implementation. Berkeley, CA: USENIX Association, 2018: 461-476

[11] Deukyeon H, Wook-Hee K, Youjip W, et al. Endurable transient inconsistency in byte-addressable persistent B^+-tree [C] //Proc of the 16th USENIX Conf on File and Storage Technologies. Berkeley, CA: USENIX Association, 2018: 187-200

[12] 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. Berkeley, CA: USENIX Association, 2015: 167-181

[13] Ismail O, Johan L, Anisoara N, 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. New York: ACM, 2016: 371-386

[14] Chen Qichen, Hyojeong L, Yoonhee K, et al. Design and implementation of skiplist-based key-value store on non-volatile memory [C] //Proc of Cluster Computing. Piscataway, NJ: IEEE, 2019, 22(2): 361-371

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

[16] Se Kwon L K, Hyun Lm, Hyunsub S, et al. WORT: Write optimal radix tree for persistent memory storage systems [C] //Proc of the 15th USENIX Conf on File and Storage Technologies. Berkeley, CA: USENIX Association, 2017: 257-270

[17] Wikipedia. 3DXPoint-Wikipedia [EB/OL]. [2020-01-15]. https://en.wikipedia.org/wiki/3D_XPoint

[18] Intel. Big memory breakthrough for your biggest data challenges [EB/OL]. [2020-01-15]. https://www.intel.com/content/www/us/en/architecture-and-technology/optane-dc-persistent-memory.html

[19] Shu Jiwu, Lu Youyou, Zhang Jiacheng, et al. Research progress on non-volatile memory based storage system [J]. Science & Technology Review, 2016, 34(14): 86-94 (in Chinese)
(舒继武, 陆游游, 张佳程, 等. 基于非易失性存储器的存储系统技术研究进展[J]. 科技导报, 2016, 34(14): 86-94)

[20] Intel. Intel memory latency checker [EB/OL]. [2019-01-15]. https://en.wikipedia.org/wiki/3D_XPoint

[21] Alper I. Intel Optane DC persistent memory operating modes explained [EB/OL]. [2019-01-05]. https://itpeernetwork.intel.com/intel-optane-dc-persistent-memory-operating-modes/#gs.83ahf1

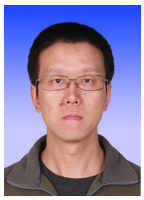
[22] Brian F, Adam S, Erwin T, et al. Benchmarking cloud serving systems with YCSB [C] //Proc of the 1st ACM Symp on Cloud Computing. New York: ACM, 2010: 143-154

[23] Joseph I, Yang Jian, Zhang Lu, et al. Basic performance measurements of the Intel Optane DC persistent memory module [R/OL]. 2019 [2020-01-15]. http://arxiv.org/abs/1903.05714

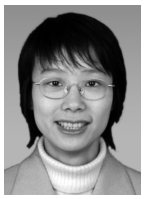
[24] Yang Jian, Juno K, Morteza H, et al. An empirical guide to the behavior and use of scalable persistent memory [C] // Proc of the 18th USENIX Conf on File and Storage Technologies. Berkeley, CA: USENIX Association, 2020: 169-182

Han Shukai, born in 1995. PhD candidate. His main research interests include key-value storage system, non-volatile memory, etc.
韩书楷,1995 年生.博士研究生.主要研究方向为键值存储系统和非易失性内存等.

Xiong Ziwei, born in 1996. Master candidate. His main interests include storage systems, non-volatile memory, distributed systems, data structures, etc.
熊子威,1996 年生.硕士研究生.主要研究方向为存储系统、非易失性内存、数据结构等.



Jiang Dejun, born in 1982. PhD, associate professor, master supervisor. Member of CCF, ACM, IEEE. His main research interests include storage architecture, storage system, distributed system, etc.
蒋德钧,1982 年生.博士,副教授,硕士生导师,CCF,ACM,IEEE 会员.主要研究方向为存储体系结构、存储系统、分布式系统等.



Xiong Jin, born in 1968. PhD, professor, PhD supervisor. Senior member of CCF, ACM, IEEE. Her main research interests include storage systems, file systems, big data storage and management, etc.
熊　劲,1968 年生.博士,教授,博士生导师,CCF,ACM,IEEE 高级会员.主要研究方向为存储系统、文件系统、大数据存储管理等.