计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2021.20200381
58(2):397-405,2021

一种多核友好的持久性内存键值系统

汪 庆 朱博弘 舒继武
(清华大学计算机科学与技术系 北京 100084)
(q-wang18@mails.tsinghua.edu.cn)

A Multicore-Friendly Persistent Memory Key-Value Store

Wang Qing, Zhu Bohong, and Shu Jiwu
(*Department of Computer Science and Technology, Tsinghua University, Beijing* 100084)

Abstract Compared with traditional memory, i.e., DRAM, persistent memory has the characteristics of large capacity and non-volatility, which brings new opportunities for building large-scale persistent memory key-value store systems. However, designing a persistent memory key-value store under a multicore server architecture faces many challenges, including CPU cache thrash due to concurrency control, the consumption and competition of limited write bandwidth for persistent memory, and aggravated conflicts between threads caused by high latency of persistent memory. In this paper, MPKV, a multicore-friendly persistent memory key-value store is proposed. By designing efficient concurrency control methods and reducing write operations to persistent memory, the system's multicore concurrency performance is fully improved. To avoid the extra persistent memory write bandwidth consumption caused by lock resources, MPKV introduces a volatile lock management mechanism to separate write lock resources from indexes and maintain them separately in DRAM. To ensure crash consistency and improve concurrent query performance, MPKV introduces a two-phase atomic write mechanism, which uses the atomic write operation instructions provided by the CPU to atomically switch the system from one consistent state to another consistent state, and further supports lock-free query. Based on the volatile lock management mechanism, MPKV also introduces a concurrent write elimination mechanism to improve the efficiency of concurrency between update operations. When two conflicting update operations occur, the concurrent write elimination mechanism allows one of the operations to return directly without any persistent memory allocation and write. Experiments show that MPKV has better performance and multicore scalability than pmemkv. Specifically, in the 18-thread environment, the throughput of MPKV reaches 1.7 to 6.2 times of pmemkv.

Key words persistent memory; multicore architecture; key-value store; concurrency control; crash consistency

摘 要 相比于传统内存,持久性内存具有容量大和非易失的特点,这为构建大规模键值存储系统提供了新的机遇.然而,在多核服务器架构下设计持久性内存键值系统面临着诸多挑战,包括并发控制带来的CPU缓存抖动、对持久性内存有限写带宽的消耗和竞争以及持久性内存高延迟带来的线程冲突加剧.提出一种多核友好的持久性内存键值系统(multicore-friendly persistent memory key-value store, MPKV),通过设计高效并发控制方法和减少对持久性内存的写操作,充分提高多核并发性能.为避免

收稿日期:2020-06-06;**修回日期**:2020-08-07
通信作者:舒继武(shujw@tsinghua.edu.cn)

锁资源带来的额外持久性内存写带宽消耗,MPKV 引入了易失性锁管理机制,将写锁资源从索引中分离,在 DRAM(dynamic RAM)中单独维护它们.为保证崩溃一致性和提高并发查询性能,MPKV 引入了 2 阶段原子写机制,利用 CPU 提供的原子写操作指令将系统从一个一致性状态原子地切换到另一个一致性状态,并支持了无锁查询.基于易失性锁管理机制,MPKV 还提出一种并发写消除机制,以提高更新操作之间的并发效率.当出现 2 个冲突的更新操作时,并发写消除机制让其中一个操作直接返回,不做任何持久性内存的分配与写操作.实验显示,MPKV 相比于 pmemkv 具有更良好的性能以及多核扩展性.其中,在 18 线程环境下,MPKV 的吞吐达到 pmemkv 的 1.7～6.2 倍.

关键词 持久性内存;多核架构;键值系统;并发控制;崩溃一致性

中图法分类号 TP302.1

新型的持久性内存(persistent memory, PM)通过内存总线与 CPU 相连,因此应用程序能够直接使用 CPU 的 load 和 store 指令对其访问.此外,它具有和 DRAM 相近的性能,以及和磁盘一样的持久化存储数据的功能.英特尔公司 2019 年正式发布了 Optane DC 持久性内存[1],将基于持久性内存的软件研究从模拟器时代带到了真实硬件平台时代,其具有百纳秒级的访问延迟以及最高 512 GB 的单条容量,为设计高速的键值存储系统提供了新的机遇.

然而,构建这样的持久性内存键值系统存在着诸多挑战,尤其是在现代基于多核架构的服务器中.具体而言,现有用于持久性内存键值系统的并发控制的方法会带来 CPU 缓存的抖动;同时,将锁资源嵌入到索引的传统设计会导致额外的持久性内存写带宽的消耗.除此之外,为保障崩溃一致性而使用的日志等机制也严重消耗着持久性内存有限的写带宽;当多个线程同时向持久性内存写数据时,会造成内存控制器和持久性内存芯片内部硬件资源的竞争,从而影响键值系统的整体性能.最后,由于存在持久化延迟,互斥临界区的时间被拉长,加剧了线程之间的冲突.

本文提出一种多核友好的持久性内存键值系统(multicore-friendly persistent memory key-value store, MPKV),通过设计高效并发控制方法和减少持久性内存的写操作提高多核并发性能.MPKV 使用桶式散列索引管理键(key)到值(value)的映射,并在此基础上提出了 3 个针对多核优化的机制:易失性锁管理、2 阶段原子写以及并发写消除机制.具体地,在易失性锁管理机制中,MPKV 将写锁资源从索引中分离,在 DRAM 中单独维护它们,以避免锁操作消耗持久性内存的写带宽;MPKV 为 DRAM 中的锁表设计了紧凑的格式,以减少锁资源分离结构导致的 CPU 缓存以及 TLB(translation lookaside buffer) 的缺失.在 2 阶段原子写机制中,MPKV 将 key 的指纹、键值数据的地址以及持久化标志包装在 64 b 字段中,并利用 CPU 提供的原子写操作指令将系统从一个一致性状态原子地切换到另一个一致性状态;基于 2 阶段原子写机制,MPKV 将查询操作完全无锁化,消除了查询操作路径上对持久性内存的写,并避免了查询和更新操作之间的冲突竞争.在并发写消除机制中,MPKV 引入了基于序列号的低开销冲突检测方法;当出现 2 个冲突的更新操作时,并发写消除机制让其中一个操作直接返回,不做任何持久性内存的分配与修改,有效节省了持久性内存有限的写带宽.

本文的主要贡献有 3 个方面:

1) 分析了持久性内存键值系统在多核架构下的性能问题.

2) 提出了一种多核友好的持久性内存键值系统 MPKV,并引入了易失性锁管理、2 阶段原子写以及并发写消除 3 个机制,提升了并发控制效率,同时节省了持久性内存写带宽.

3) 通过实验分析,MPKV 相比于 pmemkv 具有更良好的性能以及多核扩展性.其中,在 18 线程环境下,MPKV 的吞吐达到 pmemkv 的 1.7～6.2 倍.

1 背景介绍与研究动机

在本节中,主要介绍 Optane DC 持久性内存的基本特性,并分析多核架构下持久性内存键值系统设计的挑战.

1.1 Optane DC 持久性内存

由于持久性内存的非易失、可字节寻址以及性能高等优点,研究者们从体系结构和系统软件等多方面对其进行了深入的探索.然而,因为没有真实

商用的持久性内存产品,之前的研究多基于模拟器或者 DRAM.幸运的是,英特尔公司在 2019 年 4 月发布了全球首款可大规模商用的持久性内存产品:Optane DC 持久性内存.Optane DC 持久性内存基于 3DXPoint 技术,它以内存条的形式插在标准的 DDR4 接口上,单条容量可以为 128 GB,256 GB,512 GB.Optane DC 持久性内存支持 2 种配置模式:内存模式(memory mode)和应用直访模式(app direct mode).在内存模式下,DRAM 作为 Optane DC 持久性内存的缓存;应用程序无需修改就能够利用到持久性内存大容量的特点,但此种模式下 Optane DC 持久性内存无法被用于数据持久存储.在应用直访模式下,Optane DC 持久性内存作为特殊的存储设备可直接通过 CPU 的 load 和 store 指令访问;通常情况下,应用程序利用文件系统或持久性堆来管理持久性内存的名字空间.在 MPKV 中,Optane DC 持久性内存被配置成应用直访模式.

Optane DC 持久性内存的独特硬件特性影响着上层软件的设计:1)读写不对称性[2],单条的 Optane DC 持久性内存的读带宽为 6.6 GBps,而写带宽只有 2.3 GBps.2)硬件内部的最小访问粒度是 256 B.除此之外,由于 CPU 缓存的存在,数据持久化操作需要显式地调用 CPU 硬件指令,将数据从 CPU 缓存刷写至持久性内存.为了提高持久化效率,英特尔公司提出 2 个新指令:clwb 和 clflushopt;相比于传统 clflush 指令,新指令支持乱序执行,且 clwb 不会将 CPU 缓存无效化,以更好地利用 CPU 缓存.CPU 持久化指令会带来高昂的性能开销,因此如何在减少使用持久化指令的同时保证系统的崩溃一致性是设计的难点.

由于持久性内存的性能与 DRAM 相比仍有差距,现有的系统多采用混合架构[3],即同时使用 DRAM 和持久性内存.在混合架构中,DRAM 用于暂存可丢失的数据,而持久性内存用于存储核心数据.

1.2　持久性内存键值系统的多核挑战

随着摩尔定律逐渐走向终结,CPU 单个核心的性能上升缓慢.因此,处理器生产商通过在单个 CPU 中添加更多的核心来提高处理器整体性能.在这种多核架构下,如何设计高效的软件一直是活跃的研究领域.同时,持久性内存的出现将存储带到了内存级,为构建高性能的键值存储系统带来了机遇.设计多核友好的持久性内存键值系统存在着 3 方面的挑战:

1) CPU 缓存的抖动.为了保证多线程并发的正确性,键值系统使用并发原语(如读写锁)协调线程之间的同步互斥.然而,这些并发原语的使用导致对应数据在不同 CPU 核心的缓存之间来回抖动,频繁触发高昂开销的 CPU 缓存一致性协议[4],造成了 CPU 性能的急剧下降.

2) 有限的持久性内存写带宽.持久性内存的写带宽远远低于 DRAM 的写带宽.除了存储键值数据,系统为保证崩溃一致性而采用的日志等一致性机制也会带来对持久性内存写带宽的额外消耗.此外,当多个线程同时对持久性内存进行写操作时,内存控制器中会产生队列竞争,同时持久性内存芯片内的缓冲区也会发生冲突.

3) 持久化操作加剧线程冲突.在被并发原语保护的系统临界区中,线程进行持久性内存的读以及持久化操作.由于持久性内存较高的读延迟和持久化延迟,临界区的时间被拉长,导致不同线程间冲突的操作被严重阻塞.

2　MPKV 架构设计

本文提出一种多核友好的持久性内存键值系统.图 1 描述 MPKV 的总体架构.应用程序通过 GET(查询)、PUT(更新)和 DEL(删除)3 个接口操作键值系统.MPKV 同时使用 DRAM 和持久性内存.在持久性内存中,MPKV 维护着散列索引和键值数据.其中散列索引用于索引 key 对应的键值数据,而键值数据存储着应用程序写入的真正数据.

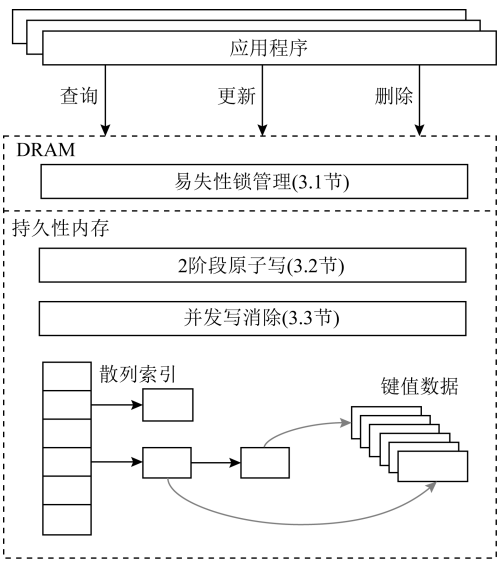


Fig. 1　Overall architecture of MPKV

图 1　MPKV 总体架构

MPKV 采用基于日志结构的策略分配内存[5].持久性内存作为存储设备以 Ext4-DAX 文件系统[6]的形式被挂载到特定目录.每个线程通过创建文件并使用 mmap(内存映射文件)系统调用获得持久性内存的字节可寻址空间.每个线程将各自的持久性内存空间组织成日志形式:每次服务持久性内存分配请求时,直接从日志尾部分配相应的空间;后台线程周期性回收日志空间的垃圾数据.该基于日志结构的分配器避免了持久性内存碎片的产生,同时由于不同线程独立管理持久性内存,分配器具有极好的多核扩展性.

为保证崩溃一致性,MPKV 需要显式地将数据从 CPU 缓存刷写至持久性内存.MPKV 选择使用 clwb 指令持久化数据,并在一组 clwb 指令之后通过加入 sfence 指令保证持久化顺序.相比其他持久化指令,clwb 具有 2 点优势:首先,多个 clwb 可以乱序执行,不阻塞 CPU 的流水线;其次,clwb 不会无效化 CPU 缓存中的数据,避免后续访问遇到 CPU 缓存缺失.

持久性内存中的散列索引为桶式散列表结构.一个桶中包含 8 个槽;每个槽存储着 key 的信息以及键值数据的地址.当输入 1 个 key 时,MPKV 通过散列函数将 key 计算成一个 64 b 的散列值:其中高 48 b 通过取余操作定位到某一个桶;低 16 b 用于定位桶中的槽,称为该 key 的指纹(fingerprint).当一个桶中的槽被用完时,系统分配新的桶,安装在原来的桶之后,形成链表结构.桶式散列表具有极好的空间局部性和紧凑的布局格式.

为解决研究动机中提到的 3 个挑战,MPKV 引入了易失性锁管理、2 阶段原子写以及并发写消除机制.易失性锁管理机制在 DRAM 中统一管理系统锁资源,以减少对持久性内存的写操作.2 阶段原子写机制通过原子指令保证系统崩溃一致性,并支持了无锁查询操作.进一步地,MPKV 设计了并发写消除机制,在存在更新冲突时减少对持久性内存的写.

3 关键技术

本节逐一介绍 MPKV 的 3 个关键技术,包括易失性锁管理机制、2 阶段原子写机制以及并发写消除机制.

3.1 易失性锁管理机制

易失性锁管理旨在通过利用 DRAM 管理锁资源,以减少对持久性内存的写开销.

散列索引使用锁保证对同一个桶并发操作的正确性.锁本质上通过 CPU 提供的原子指令如比较并交换(compare and swap, CAS)实现,包含对地址空间的写.传统的持久性索引将锁嵌入到数据结构内部,如散列表的桶中.由于锁和索引的部分数据在同一个缓存行中,这种设计能够带来比较好的空间局部性.但这种设计存在 3 方面问题:首先,锁的操作会带来对持久性内存的写,并消耗持久性内存的有限的带宽;其次,在 Optane DC 持久性内存中,由于内存内部最小更新粒度是 256 B,锁操作会带来严重的写放大;最后,当崩溃重启时,系统需要扫描整个索引,清空所有的锁字段,带来了较高的恢复开销.

为了解决如上问题,MPKV 引入了易失性锁管理机制.MPKV 将锁资源从散列索引中分离,在 DRAM 中单独维护它们.具体地,DRAM 中存有一个锁表结构,它的输入是散列桶的地址,输出是对应的锁字段.由于锁表会带来额外的一次查询,它的性能对 MPKV 至关重要.由此,MPKV 将锁表实现成 4 096 个 64 b 锁字段的数组;系统通过对散列桶的地址散列取余后得到对应的锁字段,用于并发控制.由于锁表只占 32 KB 的空间,它能够缓存在高速的 CPU 第 2 级缓存中(现有服务器 CPU 的第 2 级缓存大小在 256 KB 左右),且几乎不会引入 TLB 缺失.需要注意的是,这种空间占用极小的锁表结构会带来伪冲突,即不同的散列桶同时争抢一个锁字段.但在真实测试和负载中,2 个原因导致伪冲突的概率很低:首先,在一次键值操作中,一个线程最多持有 1 把锁,所以整个系统同一个时刻总持锁量的上限为线程数,它的值远远小于锁表的大小(4 096);其次,真实负载多为读密集的,而在 MPKV 中,易失性锁管理机制只用于管理散列桶的写锁,而 MPKV 的查询操作是无锁的(3.2 节).

3.2 2 阶段原子写

MPKV 设计了 2 阶段原子写机制,在保障系统崩溃一致性的同时,减少对持久性内存不必要的读开销,以及支持无锁的查询操作.

具体地,MPKV 利用了 CPU 提供的 64 b 原子写操作将系统原子地从一个一致性状态切换到另一个一致性状态.在 MPKV 的散列桶中,每个槽的内容为对齐的 64 b 字段,如图 2 所示.槽的最高 16 b 存着 key 的指纹,用于快速检测,以减少对键值数据的读取和比较;中间 47 b 存着键值数据地址;最低位(persist)标记着该槽的内容是否已经被成功持久化.高 16 b 和最低位可以用于存储额外信息的原因

在于2点:首先,现在的英特尔处理器的虚拟地址空间只有48 b;其次,在MPKV中,分配的持久性内存地址8 B对齐,即地址最低3 b为0.

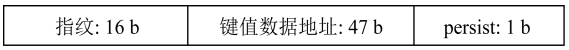


Fig. 2　Format of hash index slot

图2　散列索引槽的格式

MPKV的更新操作流程如下:1)定位到对应的散列桶以及其中的槽,并成功持有易失性锁表内的对应锁;2)从持久性内存中分配键值数据的空间,将键值数据写入其中,并通过调用CPU持久化指令将其持久化;3)根据散列槽的格式槽,生成〈指纹,键值数据地址,0〉三元组(64 b);4)进行2阶段原子写,在第1阶段中,将64 b三元组原子地写入对应的散列槽,并调用CPU持久化指令将其从CPU缓存刷写至持久性内存;在第2阶段中,将散列槽的最低位(即persist位)置成1,标志散列槽已成功被持久化.

2阶段原子写机制的引入使得MPKV的查询操作可以完全无锁化.具体地,查询操作的流程如下:1)定位到对应的散列桶;2)依次遍历其中的散列槽(通过64 b原子读),若某一个槽中的指纹与key的指纹匹配,且对应键值数据中的key也相同,则定位到该槽;如果该槽的persist位为1,则直接读取键值数据中的value;如果该槽的persist位为0,则协助完成2阶段原子写:调用CPU持久化指令使槽的内容持久性,最后调用CAS指令原子地更新persist值为1,若CAS成功,读取键值数据中的value,否则,从该槽开始继续向后遍历散列槽.值得注意的是,persist位用于保证查询操作不会读到未被成功持久化记录到散列索引的键值数据,以达到正确的语义.此外,查询操作协助完成2阶段原子写,这是为了系统崩溃重启后无需遍历所有的散列槽并将persist置位为1;同时,该协助过程未引入不一致的问题,这是因为在此之前对应写操作的数据已成功被持久化.

无锁查询操作带来了内存释放的问题,因为其他操作无法感知并发的查询操作.MPKV引入了一种基于epoch的内存回收方法[7];该方法将时间划分为连续的周期(epoch).具体地,MPKV维护一个全局epoch计数器,同时每个线程维护一个本地epoch计数器.每个键值操作开始时,将本地计数器的值更新成全计数器的值.当更新和删除操作需要释放持久性内存空间时,将该内存空间以〈本地计数器的值,内存空间地址〉格式记录到本地回收链表中.MPKV使用一个后台线程周期性地将全局计数器加1并安全回收内存:该线程收集所有其他线程的本地计数器,并计算出其中的最小值,称作最大安全epoch,此时,所有本地回收链表中小于最大安全epoch的内存可以被安全回收.

2阶段原子写机制带了3方面的性能优势:1)最小化崩溃一致性的开销.在更新操作的过程中,无需记录日志;2)最小化系统恢复的开销.在系统恢复的过程中,无需遍历散列索引或者日志内容;3)多核并发.查询操作完全无锁,避免了传统的读写锁造成的持久性内存写开销以及缓存抖动的问题,同时将更新操作的键值数据持久化过程排除至并发查询操作的关键路径之外.

3.3　并发写消除机制

键值存储系统面临着严重的并发问题.易失性锁管理和2阶段原子写机制通过优化并发控制机制来加速并发访问.与这两者不同,并发写消除机制利用并发冲突带来的机遇,来减少对持久性内存的写,提高并发效率.

如图3所示,当出现2个冲突的更新操作时,并发写消除机制让其中一个操作直接返回,不做任何持久性内存的分配与修改,并标记为完成.该机制未违反线性化(linearizable)的并发语义[8]:2个在时间上重叠的操作执行完之后,等价于某一种串行顺序.

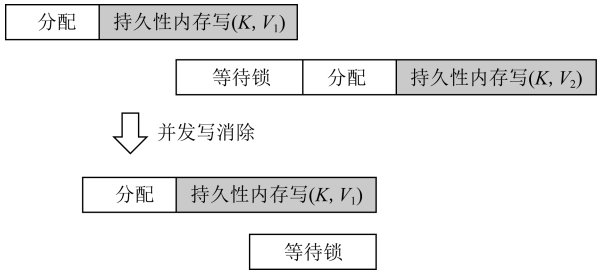


Fig. 3　Concurrent write elimination

图3　并发写消除

具体地,在MPKV引入了一种基于序列号的低开销方法,用以支持并发写消除.每个更新操作被赋予一个独一无二的序列号:该序列号所占空间为64 b,其中高8 b是执行该更新操作的线程号,低56 b在每次更新操作之时加1.利用序列号,并发冲突的更新操作可以被检测,进而被消除.在更新操作的执行过程中,线程将序列号写入易失性锁表的对应锁字段,并在键值数据中记录该序列号.当一个

线程发现存在冲突,即锁已经被其他线程获得时,它读取锁字段中对应的序列号,记作冲突序列号 cseq;该线程持锁成功之后,遍历散列桶中的槽;当需要更新的键值数据的序列号等于 cseq 时,该线程取消持久性内存分配和持久化操作,即写操作被消除.

当多个线程对同一个键值数据并发进行更新时,并发写消除机制能有效地减少对持久性内存的写带宽的消耗,提高了系统整体吞吐.同时,针对非冲突的更新操作,由于该机制使用基于序列号的轻量级方法,冲突检测的开销极低.

3.4 讨 论

MPKV 引入了易失性锁管理、2 阶段原子写以及并发写消除机制.这些技术在提高系统性能的同时也带来了额外的空间开销.易失性锁管理机制需要占用 32 KB 的 DRAM 空间.2 阶段原子写机制由于将指纹信息和持久化标记嵌入到 64 b 键值地址字段中,无需消耗额外空间.在并发写消除机制中,每份键值数据需要额外 64 b 空间存储序列号;对于平均键值大小为 150 B 的典型负载(如 Facebook 的 UDB[9]),并发写消除机制会造成 5%左右的空间开销.考虑到 3 个技术带来的性能优势,这些较少的空间开销是可以接受的.

4 实 验

本节实验将从 3 个方面对 MPKV 进行测试,并分析其与现有系统的性能差异:1)对比测试不同查询操作比例下键值系统的性能;2)对比测试不同键值尺寸下键值系统的性能;3)多核扩展性对比测试.

4.1 实验平台

本实验使用的实验平台配置信息如表 1 所示.本实验使用的持久性内存设备是英特尔 2019 年推出的 Optane DC 持久性内存.为支持持久性内存,实验平台配置了相应的 CPU 和主板.实验机器拥有 2 个 NUMA(non uniform memory access)节点.为避免跨 NUMA 访问带来的性能下降,本实验只使用单个 NUMA 节点,即只使用一块 CPU(包含 18 个核心)和对应 NUMA 节点上的持久性内存.

本实验将 MPKV 与 pmemkv[10]进行性能对比.pmemkv 是一款持久性内存键值系统,它基于英特尔的持久性内存开发套件(persistent memory development kit, PMDK)[11]实现,并提供了多种存储引擎.在本实验中,为了公平比较,pmemkv 选择了基于并发散列表的 cmap 存储引擎;该存储引擎采用 PMDK 进行持久性内存分配,同时使用读写锁进行并发控制.除此之外,实验还比较了 MPKV 的基准版本,称为 Baseline;Baseline 的散列索引和内存分配部分与 MPKV 相同,但使用嵌入在散列桶中的读写锁进行并发控制.在所有实验中,负载生成的 key 满足均匀分布.负载生成的 key 的空间为 1 亿,即存在 1 亿个不同 key.在默认情况下,实验使用 18 个线程,这与单个 NUMA 节点的 CPU 核心数目相同,以避免多个线程争夺相同 CPU 核心资源;同时 key 和 value 的尺寸分别为 8 B 和 64 B.

Table 1 Platform Configuration

表 1 实验平台配置信息

名称	配置
处理器	Intel® Xeon® Gold 6240 M×2
DRAM	Samsung 32 GB×6
持久性内存	Intel Optane DC PM 256 GB×6
操作系统	Ubuntu18.04, Linux4.15.0

4.2 读写比例对比测试

图 4 展示了在不同的查询操作比例下,MPKV 及其对比系统的吞吐情况.从图 4 可知,当查询操作的比例上升时,所有键值系统的吞吐随之上升.这是因为持久性内存的写带宽很低,更新操作的效率比查询效率低;同时,冲突的更新操作无法并行.pmemkv 的吞吐在所有查询操作比例下均最低,这是因为:1)对于更新操作,pmemkv 使用了事务机制保证崩溃一致性,带来写日志的开销,同时 PMDK 的内存分配操作需要持久性化元数据,扩展性也很差;而 MPKV 和 Baseline 的更新操作无需写日志,且它们使用的基于日志结构的分配器开销低、扩展性良好.2)对查询操作,pmemkv 与其他系统不同,未使用指纹技术,造成了严重的持久性内存读放大现象.

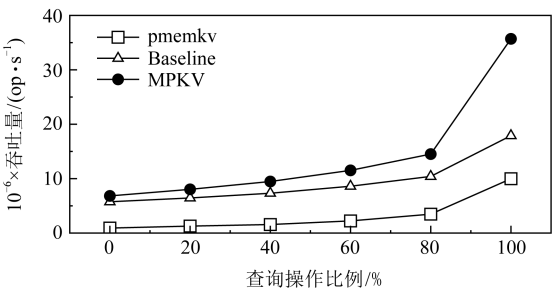


Fig. 4 Throughput with varying GET ratio

图 4 不同查询比例下的吞吐量

当查询操作比例为 0,即全为更新操作时,

MPKV 的吞吐比 Baseline 高出了 18%.这是因为 MPKV 的易失性锁管理机制避免了写锁带来的对持久性内存的写;进一步,并发写消除机制不仅在一定程度上节省了持久性内存的写入带宽,还通过缩短持锁时间,提高了冲突的更新操作之间的并发效率.

当查询操作比例为 100%时,MPKV 的吞吐比 Baseline 高出了 99%.这是因为 MPKV 基于 2 阶段原子写机制设计了无锁读.无锁读不仅避免了读锁带来的持久性内存的写,还消除了由于多个读者操作读锁带来的 CPU 缓存抖动的问题.为了进一步探究 MPKV 性能优势的来源,实验采集了单个 CPU 周期执行的指令数目(instruction per cycle, IPC).在查询操作比例为 100%时,Baseline 的 IPC 为 0.08,而 MPKV 的 IPC 为 0.14.这 2 个系统的 IPC 值均很低,这是由于负载的内存占用较大,缓存缺少发生频繁.MPKV 的 IPC 是 Baseline 对应值的 1.75 倍,这是因为如上文所述,无锁读减少了持久性内存的写以及缓存抖动,由此提高了 CPU 的执行效率.

4.3　键值尺寸对比测试

图 5 展示了在不同的键值尺寸下,MPKV 及其对比系统的吞吐情况.此时查询操作比例为 50%,同时 key 的尺寸固定为 8 B,而 value 的尺寸在变化.从图 5 可知,当 value 尺寸增大时,所有键值系统的吞吐随之下降.这是因为更大的 value 会消耗更多的持久性内存带宽和 CPU 执行周期.特别地,当 value 的尺寸为 64 B 和 128 B 时,性能下降幅度很小.这是因为 Optane DC 持久性内存内部的最小粒度是 256 B,在这 2 种情况下,持久性内存消耗的内部带宽相同.需要注意的是,当 value 尺寸为 256 B 时,由于 key 和其他元数据的存在,整个键值数据的总尺寸大于 256 B.

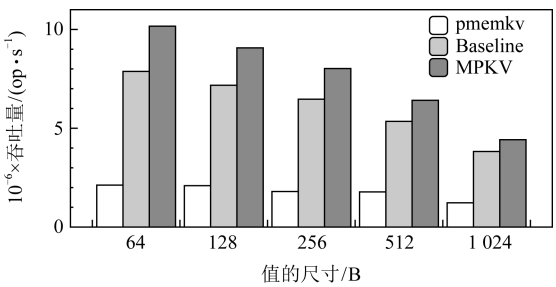


Fig. 5　Throughput with varying value size

图 5　不同值的尺寸下的吞吐量

当 value 尺寸逐渐增大时,MPKV 相对于 Baseline 的吞吐提高比例从 22%降到了 12%.这是因为持久化键值数据的开销逐渐增大,导致 MPKV 的优化技术带来的相对收益减小.pmemkv 由于冗余复杂的软件设计,性能远低于其他 2 个系统.

4.4　多核扩展性测试

图 6 展示了 MPKV 及其对比系统的在写密集负载下的多核扩展性.此时查询操作的比例为 50%.从图 6 中可以观察到,MPKV 具有最好的多核扩展性,而 Baseline 在使用了 12 个线程时,吞吐接近饱和.这是因为在写密集负载中,会产生大量的更新与更新、更新与查询操作之间的冲突,导致线程经常相互阻塞.针对更新与更新操作之间的冲突,MPKV 的并发写消除机制减少了冲突操作带来的持久性内存分配与持久化写,提高了并发效率;MPKV 基于 2 阶段原子写机制实现的无锁读,完全消除了更新与查询操作之间的冲突.

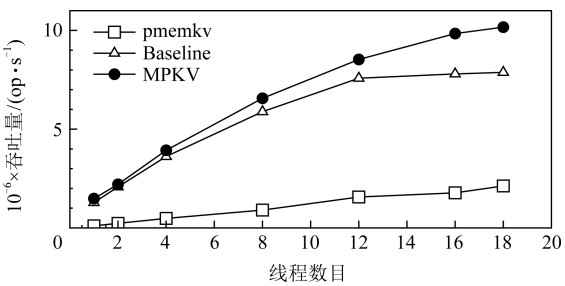


Fig. 6　Throughput with varying number of threads (write-intensive workload)

图 6　不同线程数目下的吞吐量(写密集负载)

特别地,当线程数目小于等于 4 时,MPKV 和 Baseline 之间的差别并不明显.这是由于此时并发冲突不足,MPKV 的设计优势难以充分体现.其中 MPKV 8%左右的性能提高来源于易失性锁管理机制和无锁读减少了对持久性内存的写,缩短了单个操作的完成时间.

进一步,图 7 展示了 MPKV 及其对比系统的在读密集负载下多核扩展性.此时查询操作的比例为

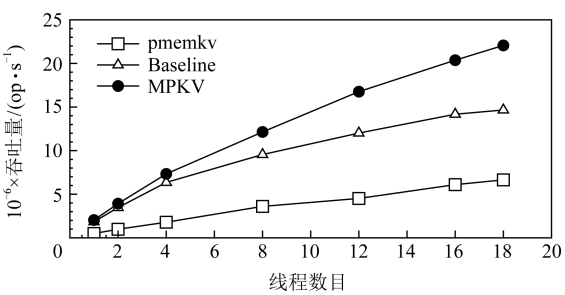


Fig. 7　Throughput with varying number of threads (read-intensive workload)

图 7　不同线程数目下的吞吐量(读密集负载)

95%.由于查询操作从语义上天生支持并发,在读密集负载下 3 个键值系统的扩展性均表现良好.此时 MPKV 的性能优势主要来自基于 2 阶段原子写机制的无锁读:它完全消除了查询操作由于读锁带来的线程之间 CPU 缓存的竞争.此外,pmemkv 的吞吐最低原因来源于 2 点:1)5%的更新操作需要使用基于日志的事务机制,软件开销高;2)95%的查询操作由于未利用指纹机制,造成了额外的持久性内存读,带来来了较高延迟.

5 相关工作

本节从持久性内存散列索引,持久性内存键值系统和多核友好的键值系统 3 方面介绍相关工作.

1) 持久性内存散列索引.2018 年提出的写优化持久性内存散列索引 level hashing[12],采用无日志的方式优化更新和删除操作,并通过 2 级的散列结构减少散列索引扩展时需要的数据挪动和持久性开销.2019 年提出的写优化动态散列索引 CCEH[13],一方面通过缓存优化,提高查询性能;另一方面通过采用可扩展散列结构,避免了散列索引扩展时需要耗时的散列表重建.2020 年提出的 Dash[14] 采用乐观读处理查询操作,并通过均衡策略提高整个散列索引的空间利用率.与上述的持久性内存散列索引相比,MPKV 采用了无锁读并把写锁放入了 DRAM,所以具有更好的多核扩展性和更少的持久性内存写带宽消耗.

2) 持久性内存键值系统.2017 年提出的持久性内存键值系统 HiKV[15],采用散列表和 B^+ 树混合的索引结构,以同时保证高效的单点查询和范围查询.HiKV 将 B^+ 树索引维护在 DRAM 中,避免对持久性内存额外的读写,并通过后台线程异步地更新 B^+ 树索引.2018 年提出的 NoveLSM[16] 利用持久性内存优化基于日志结构合并树(log structured merge tree, LSM Tree)的键值系统.具体地,NoveLSM 在持久性内存中构造了 memtable,这种设计不仅降低了序列化和反序列化的开销,也减少了垃圾回收带来的前台阻塞.与 NoveLSM 不同,2019 年提出的 SLM-DB[17] 完全改造了 LSM 树的结构.SLM-DB 采用单层的结构,通过持久性 B^+ 树索引键值数据,并设计了选择性的垃圾回收策略以减少写放大.2020 年提出的 FlatStore[18] 则利用 Optane DC 持久性内存内部最小粒度是 256 B 的特征,设计了紧凑的日志格式,通过高效的 CPU 核心之间协同批处理机制以避免持久性内存内部的写放大.此外,FlatStore 利用了高速的远程内存直接访问(remote direct memory access, RDMA)网络技术,以提供远程跨网络的键值访问接口.与上述持久性内存键值系统不同,MPKV 关注的是由于多核并发带来的性能问题,并通过一系列技术解决该问题.

3) 多核友好的键值系统.2017 年提出的 SLB[19] 将热点数据缓存住,以提高并发查询的效率,并在真实负载下获得了 73%的性能提高.2020 年提出的 HotRing[20],设计了环状的散列桶结构,通过在线检测数据的冷热程度,将热点数据挪动到环的头部,以减少内存访问次数.上述技术难以用于持久性键值系统,因为持久性键值系统需要保证崩溃一致性,数据缓存和挪动会带来高昂的一致性开销.

6 结 论

构建适应于持久性内存和多核服务器架构的键值系统面临诸多挑战.本文提出一种多核友好的持久性内存键值系统 MPKV,通过引入易失性锁管理,2 阶段原子写以及并发写消除机制,有效减少持久性内存的写操作,并提升了并发效率.实验显示,相比于其他系统,MPKV 具有更良好的吞吐和多核扩展性.MPKV 目前未针对 NUMA 进行优化,这也是未来值得探索的方向.

参 考 文 献

[1] Intel. Optane DC persistent memory [OL]. [2020-06-01]. https://www.intel.com/content/www/us/en/architecture-and-technology/ optane-dc-persistent-memory.html

[2] Yang Jian, Kim J, Hoseinzadeh M, 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

[3] 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. Berkeley, CA: USENIX Association, 2016: 323-338

[4] David T, Guerraoui R, Trigonakis V. Everything you always wanted to know about synchronization but were afraid to ask [C] //Proc of the 24th ACM Symp on Operating Systems Principles. New York: ACM, 2013: 33-48

[5] Hu Qingda, Ren Jinglei, Badam A, et al. Log-structured non-volatile main memory [C] //Proc of USENIX Annual Technical Conf. Berkeley, CA: USENIX Association, 2017: 703-717

[6] Matthew W. Support ext4 on NV-DIMMs [OL]. [2020-06-01]. https://lwn.net/Articles/588218/

[7] Hart T E, McKenney P E, Brown A D, et al. Performance of memory reclamation for lockless synchronization [J]. Journal of Parallel and Distributed Computing, 2007, 67 (12): 1270-1285

[8] Herlihy M P, Wing J M. Linearizability: A correctness condition for concurrent objects [J]. ACM Transactions on Programming Languages and Systems (TOPLAS), 1990, 12 (3): 463-492

[9] Cao Zhichao, Dong Siying, Vemuri S, et al. Characterizing, modeling, and benchmarking RocksDB key-value workloads at Facebook [C] //Proc of the 18th USENIX Conf on File and Storage Technologies. Berkeley, CA: USENIX Association, 2020: 209-223

[10] Intel. Key/Value datastore for persistent memory [OL]. [2020-06-01]. https://github.com/pmem/pmemkv/

[11] Intel. Persistent memory development kit [OL]. [2020-06-01]. https://pmem.io/pmdk/

[12] 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

[13] Nam M, Cha H, Choi Y, 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

[14] Lu Baotong, Hao Xiangpeng, Wang Tianzheng, et al. Dash: Scalable hashing on persistent memory [J]. arXiv preprint, arXiv:2003.07302, 2020

[15] 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 Technical Conf. Berkeley, CA: USENIX Association, 2017: 349-362

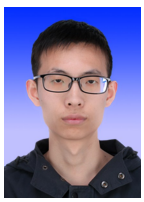
[16] Kannan S, Bhat N, Gavrilovska A, et al. Redesigning LSMs for nonvolatile memory with NoveLSM [C] //Proc of USENIX Annual Technical Conf. Berkeley, CA: USENIX Association, 2018: 993-1005

[17] Kaiyrakhmet O, Lee S, Nam B, et al. SLM-DB: Single-level key-value store with persistent memory [C] //Proc of the 17th USENIX Conf on File and Storage Technologies. Berkeley, CA: USENIX Association, 2019: 191-205

[18] Chen Youmin, Lu Youyou, Yang Fan, et al. FlatStore: An efficient log-structured key-value storage engine for persistent memory [C] //Proc of the 25th Int Conf on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2020: 1077-1091

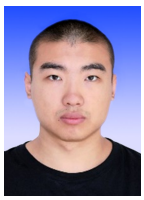
[19] Wu Xingbo, Ni Fan, Jiang Song. Search lookaside buffer: Efficient caching for index data structures [C] //Proc of the 2017 Symp on Cloud Computing. New York: ACM, 2017: 27-39

[20] Chen Jiqiang, Chen Liang, Wang Sheng, et al. HotRing: A hotspot-aware in-memory key-value store [C] //Proc of the 18th USENIX Conf on File and Storage Technologies. Berkeley, CA: USENIX Association, 2020: 239-252



Wang Qing, born in 1997. PhD candidate. His main research interests include storage systems and memory systems.

汪　庆,1997 年生.博士研究生.主要研究方向为存储系统和内存系统.



Zhu Bohong, born in 1993. Master candidate. His main research interests include storage systems and file systems.

朱博弘,1993 年生.硕士研究生.主要研究方向为存储系统和文件系统.

Shu Jiwu, born in 1968. Professor and PhD supervisor. Fellow of CCF. His main research interests include non-volatile memory systems and technologies, network (cloud/big data) storage system, storage security and reliability, and parallel and distributed computing.

舒继武,1968 年生.教授,博士生导师,CCF 会士.主要研究方向为非易失内存系统与技术、网络(云/大数据)存储、存储安全与可靠性以及并行与分布式计算.