doi:10.3772/j.issn.1002-0470.2023.01.003

BOOM-KV:基于 RDMA 的高性能 NVM 键值数据库①

李文捷②*,**,*** 蒋德钧*,**,*** 熊 劲*,** 包云岗*,**,***

(*中国科学院计算技术研究所 北京 100190)

(**中国科学院大学计算机科学与技术学院 北京 100049)

(***鹏城实验室 深圳 518055)

摘 要 随着英特尔傲腾数据中心持久化内存模块(DCPMM)开始进入市场以及远程直接内存访问(RDMA)硬件成本的降低,设计融合非易失性内存(NVM)和 RDMA 的键值(KV)数据库面临新的机遇和挑战。构建基于 NVM 和 RDMA 的 KV 数据库的关键在于设计一个高效的通信协议。遗憾的是,现有工作或采用 NVM 不感知的 RDMA 协议,或采用低效的 NVM 感知的 RDMA 协议,这导致它们无法最大化 KV 数据库的性能。本文提出了 BOOM 协议——一种新型的 NVM 感知的 RDMA 协议。相较于 NVM 不感知的协议,BOOM 协议允许直接对远端 NVM 进行 RDMA 操作,消除了冗余的数据拷贝;相较于现有的 NVM 感知的协议,它可以显著减少元数据请求,降低 KV 请求的端对端延迟。在 BOOM 协议的基础上构建了 BOOM-KV,并针对服务端中央处理器(CPU)利用率和宕机持久化等问题进一步进行优化。将 BOOM-KV 与最新的研究成果进行对比,结果表明,BOOM-KV 能显著降低请求延迟,其中 PUT 延迟最大降低了 42%,GET 延迟最大降低了 41%,并且展现出良好的扩展性。

关键词 非易失性内存(NVM);远程直接内存访问(RDMA);键值(KV)数据库

0 引 言

键值(key-value,KV)数据库是现代数据中心基础设施的重要组成部分,电子商务、社交网络和图片存储等应用的发展对它的性能提出了更高的要求。近年来新型硬件也在迅速发展,非易失性内存(non-volatile memory,NVM)可以提供高性能的存储以及字节粒度的访问,大幅降低了访问存储介质的开销。远程直接内存访问(remote direct memory access,RDMA)技术可以提供高性能的网络通信,并允许直接对远端内存进行访问,大幅降低了网络通信的开销。因此,将 RDMA 和 NVM 这 2 种新兴技术融合到 KV 数据库中既是机遇,同时也面临着挑战。

构建 RDMA 和 NVM 融合的 KV 数据库的关键挑战在于设计一个定制的基于 RDMA 的通信协议,该协议利用 RDMA 和 NVM 的特性来支持 KV 数据库操作,如 GET、PUT 等。目前已有 2 类 RDMA 协议在相关文献中提出[1]。第 1 类协议是 NVM 不感知的(NVM-oblivious)RDMA 协议,相关工作有 Pilaf[2]、HERD[3]和 eRPC[4]等。这一类协议无法感知 NVM 的存在,因此客户端只能远程直接访问服务器上的动态随机存储器(dynamic random access memory,DRAM)。这一协议类似于传统的远程过程调用(remote procedure call,RPC)协议,当用于构建 KV 数据库时,它们只是简单地通过 RDMA 操作传递客

① 北京市自然科学基金 - 海淀原始创新联合基金(L192038),中国科学院战略性先导科技专项(XDB44030200)和中国科学院青年创新促进会资助项目。

② 男,1992 年生,博士生;研究方向:面向新型存储介质的键值数据库,分布式系统;联系人,E-mail:liwenjie@ict.ac.cn。

(收稿日期:2021-09-06)

户端发送的 GET、PUT 请求,并在服务器处理完请求后,返回给客户端相应的结果。由于无法对 NVM 进行感知,这类协议会导致 DRAM 和 NVM 之间额外的数据拷贝。第 2 类协议是 NVM 感知的(NVM-aware)RDMA 协议,代表的工作有 Octopus[5]、Forca[6]和 RDMP-KV[7]等。这一类协议能感知 NVM 的存在,因此客户端可以直接远程访问服务器上的 NVM。与 NVM 不感知的 RDMA 协议相比,它绕过了 DRAM 而直接对 NVM 中的数据进行访问,消除了 NVM 和 DRAM 之间的数据拷贝开销。尽管绕过了 DRAM,现有的 NVM 感知的 RDMA 协议均包含 2 个阶段,导致了额外的网络开销。

尽管相关工作[6-7]指出 NVM 感知的 RDMA 协议能为 KV 数据库提供更好的性能,但是这些工作都依赖于模拟的 NVM 方法,在英特尔傲腾数据中心持久化内存模块(Optane DC persistent memory module, Optane DCPMM)商用的当下,有必要对相关工作进行重新评估。除此之外,在构建 NVM 和 RDMA 融合的 KV 数据库时,还面临以下几点挑战。第一,如何降低 KV 数据库的请求延迟。NVM 感知的 RDMA 协议为了获取元数据会额外增加一轮网络往返,增加 KV 数据库请求的延迟。第二,如何降低 KV 数据库的垃圾回收开销。通过日志结构(log-structured memory)[8]管理 NVM 空间需要引入垃圾回收机制释放空间,简单地通过 RDMA 追加更新日志会产生大量的垃圾数据,会增加服务端中央处理器(central processing unit, CPU)的开销。第三,如何降低 KV 数据库的持久化开销。由于 CPU 缓存(cache)机制,RDMA 写入的数据会在服务端 CPU cache 中缓存,未写回的缓存数据掉电后会丢失,现有的工作要么不提供宕机持久化保证,要么简单地使用专用指令写回 KV 数据,持久化开销较大。第四,如何保证 KV 数据库在发生宕机后能正确恢复。客户端直接更新远端的 NVM,使得数据操作和元数据管理分离,需要引入额外的协调机制来保证宕机恢复。

为了解决上述几点挑战,本文提出 BOOM 协议,一种新型的 NVM 感知的 RDMA 协议,并基于该协议构建了 BOOM-KV。相较于 NVM 不感知的 RDMA 协议,BOOM 协议可以消除 DRAM 和 NVM 之间的数据拷贝。在保证数据持久化的基础上,BOOM 协议与现有 NVM 感知的 RDMA 协议的关键区别在于,它只需要一轮网络通信即可以完成 PUT、GET 操作。此外,为了减低垃圾回收的开销,BOOM-KV 提出混合的更新模式,以此减少垃圾数据的产生。为了降低持久化开销,BOOM-KV 提出了协作的写回策略,以此提高 KV 数据写回的并行性。最后,通过综合性测试,对于任意粒度的 KV 数据,BOOM-KV 的读、写延迟都优于 RDMP-KV,其中 PUT 延迟最大降低 42%,GET 延迟最大降低 41%。在多线程测试下,BOOM-KV 也展现出较好的扩展性能。

1 背 景

1.1 远程直接内存访问

RDMA 传统上主要应用在高性能计算领域。近年来,随着 RDMA 网络接口卡(RDMA-capable NIC, RNIC)价格下降,将 RDMA 应用在商业数据中心也越来越普遍。RDMA 允许客户端直接对服务器的内存进行访问,实现内核旁路(kernel-bypass)和零拷贝(zero-copy)等特性。通过旁路服务器的 CPU 以及硬件实现的网络协议栈(链路层到传输层),RNIC 可以提供低延迟和高带宽的性能:单个端口能达到 100 Gbps以及2 μs左右的延迟。

为了与服务器进行 RDMA 通信,客户端需要创建队列对(queuepair,QP),应用通过向 QP 发送请求来执行 RDMA 操作(又称 RDMA verbs)。此外,QP 还与完成队列(completion queue,CQ)相关联,它用于通知先前发送至 QP 中的请求是否已经完成。RDMA 支持多种操作类型,基于内存的操作包括 RDMA 写(RDMA write)、读(RDMA read)和原子操作。当执行这些操作时,应用需要在 RDMA 请求中指定要操作的远端内存地址,而服务器 CPU 完全不感知。基于消息的操作包括 RDMA 发送(RDMA send)和 RDMA 接收(RDMA receive),不同于基于内存的操作,应用不需要为 RDMA send 请求指定远端内存地址,但是服务器 CPU 需要预先发送 RDMA receive 来指定写入地址。

现有的 RDMA 实现提供了 3 种传输模式:可靠连接(reliable connection, RC)、不可靠连接(unreliable connection,UC)和不可靠数据报(unreliable datagram,UD)。不同的传输模式支持不同的 RDMA 操作子集,例如 RC 支持所有的 RDMA 操作,UC 不支持 RDMA read,而 UD 完全不支持基于内存的 RDMA 操作。

1.2 NVM

随着英特尔 Optane DCPMM 的发布,NVM 终于在商业上可用。NVM 不仅可以提供持久化的存储和近似 DRAM 的性能,同时也允许字节粒度的访问。另一方面,CPU 缓存层次结构(cache hierarchy)给 Optane DCPMM 的持久化带来挑战,所有写入 Optane DCPMM 的数据都先在 CPU cache 中缓存,如果发生系统宕机,未写回的数据将全部丢失。为了保证宕机持久化,英特尔的指令集提供了 clflush、clflushopt 和 clwb 等指令,它们可以将易失性的缓存行(cache-line)写回到持久化介质。此外,还需要执行 sfence 等指令以保证先前的 cache-line 写回已经完成。

Optane DCPMM 提供 2 种使用模式,一种是内存模式,在该模式下 Optane DCPMM 作为大容量的 DRAM 使用,可以满足有较大内存需求的应用,但不能提供持久化的存储;另一种是应用直连(App Direct)模式,在该模式下应用通过 mmap 将持久化内存映射到地址空间,并直接通过 load/store 指令对其进行访问,可以提供持久化的存储。本文默认在 App Direct 模式下使用 Optane DCPMM。

1.3 基于 RDMA 的通信协议

根据通信协议是否能直接访问远程 NVM,可以将它们分为 NVM 感知的 RDMA 协议和 NVM 不感知的 RDMA 协议,这 2 类协议在处理 KV 请求时的流程如下所述。

对于 NVM 不感知的 RDMA 协议,客户端通过 RDMA 将 GET 或 PUT 请求发送到服务端的 RDMA 缓冲区(在 DRAM 中分配),服务器线程不断轮询新的请求。如果收到 PUT 请求,首先向 NVM 分配器分配一个持久化对象,其次将 PUT 请求中的 KV 数据拷贝到持久化对象中,再次通过 clwb 等指令将持久化对象写回到 NVM,以确保宕机持久性。完成持久化操作后,需要进一步对索引进行更新,最后通过 RDMA 向客户端返回 PUT 操作的结果。如果收到 GET 请求,首先会通过查询索引判断 KV 数据是否存在,如果存在则进一步定位到 KV 数据所在的持久化对象的地址,然后将持久化对象拷贝到 RDMA 缓冲区,最后通过 RDMA 将该缓冲区的 KV 数据返回给客户端。

对于现有的 NVM 感知的 RDMA 协议,其流程基本上可以分为 2 个阶段,第 1 阶段进行元数据请求,第 2 阶段进行数据操作。在第 1 阶段中,客户端通过 RDMA 操作将 GET 或 PUT 的元数据请求发送到服务端 RDMA 缓冲区。服务端线程如果收到 PUT 请求,则向 NVM 分配器分配一个持久化对象,然后将该持久化对象注册为 RDMA 内存区域(memory region,MR),最后向客户端返回持久化对象的元数据——地址和权限信息(rkey),它们将用于第 2 阶段的数据操作。类似地,如果收到 GET 请求,则通过索引查找到对应持久化对象的元数据,并返回给客户端。在第 2 阶段中,客户端已接收到待操作的持久化对象的元数据。对于 PUT 请求,客户端通过 RDMA write 将本地的 KV 数据写入到远端的持久化对象中,服务器等待 RDMA write 完成后,将新写入的 KV 数据写回并更新对应的索引项,最后向客户端返回 PUT 操作的结果。对于 GET 请求,客户端通过 RDMA read 将远端持久化对象中的 KV 数据读到本地的缓冲区,该过程不需要服务器 CPU 的干预。

2 动机和挑战

一个高效的 RDMA 通信协议对于构建 RDMA 和 NVM 融合的 KV 数据库至关重要。采用 NVM 不感知的 RDMA 协议相对比较简单,它与 KV 数据库组件的耦合程度低,只需要将 KV 数据库的网络通信模块替换成现有的 RDMA 协议即可。但是 NVM 不感知的 RDMA 协议无法充分利用 RDMA 零拷贝的特性以及 NVM 可字节寻址的特性,导致需要在 DRAM 和 NVM 之间进行额外的数据拷贝。相反

地,采用 NVM 感知的 RDMA 协议可以避免 DRAM 和 NVM 之间数据拷贝,并且降低服务器 CPU 的开销。然而,现有的 NVM 感知的 RDMA 协议需要经过 2 个阶段来完成一次 KV 操作,这也增加了网络通信的开销,此外这类协议与 KV 数据库组件的耦合程度高,实现难度更大。

尽管有工作[6-7]指出 NVM 感知的协议比 NVM 不感知的协议有更大的性能优势,尤其是对于大粒度的 KV 操作,但现有的工作完全基于模拟的 NVM 开展。根据观察,在使用真实的 Optane DCPMM 的情况下,要发挥出前者的性能优势并不容易。通过初步的实验和分析,发现现有的基于 RDMA 的通信协议无法充分利用 NVM 和 RDMA 的特性,无法最大化、持久化 KV 数据库的性能。在构建 RDMA 和 NVM 融合的 KV 数据库的过程中,主要面临以下几点挑战。

挑战 1:如何减少 KV 数据库的端对端延迟。现有的 NVM 感知的 RDMA 协议允许客户端直接对远端服务器上的 NVM 进行访问,但客户端每次执行 KV 操作前,必须先从服务器获得相关持久化对象的地址和 rkey,随后才能进行 RDMA read/write。然而,获取地址和 rkey 等元数据会导致额外的网络开销,增加 KV 请求的端对端延迟。本文对 RDMP-KV 的两阶段协议进行了定量分析,结果如图 1 所示,对于 64 B 的 PUT 请求,阶段 1 的延迟占了总延迟的 41.4%,对于 64 B 的 GET 请求,阶段 1 的延迟占了总延迟的 57.5%。特别地,越是小粒度的 KV 请求,元数据请求的开销占比就越大。

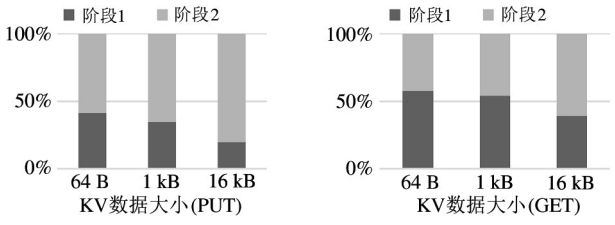

图 1 RDMP-KV 阶段延迟分析

挑战 2:如何降低 KV 数据库的垃圾回收开销。目前有不少工作将日志结构(log-structured)的思想应用在面向 NVM 的存储系统中[9-10]。日志结构能将随机写转换成顺序写,它的优势在于减少内存碎片化,并提供宕机原子性保障,最近还有工作[11]利用日志结构来批量处理 KV 请求,分摊 NVM 的持久化开销。然而,日志结构只允许追加更新(append-only)一种写模式,简单地通过 RDMA 进行追加更新会产生大量的垃圾数据,需要通过垃圾回收机制来释放新的空间,但频繁地进行垃圾回收会导致服务器的 CPU 开销增大,影响 KV 数据库的性能。

挑战 3:如何降低 KV 数据库的持久化开销。现有的 CPU 缓存层次结构会缓存 RDMA 写入的数据,如果不将这些数据主动写回到 NVM 中,那么系统宕机时可能会导致 KV 数据丢失。现有的 RDMA 和 NVM 融合的 KV 数据库要么不提供宕机持久化保证,要么采用低效的持久化方法。图 2 显示,对于 16 kB的 PUT 操作,数据持久化所用的时间占了总延迟的 41.2%。特别地,随着 KV 粒度的增大,数据持久化的开销就越大。

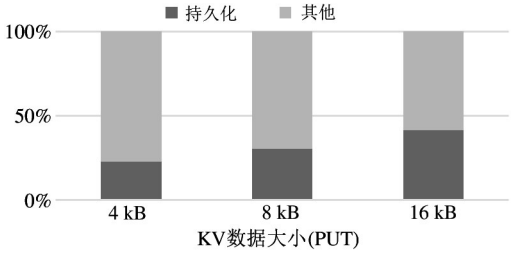

图 2 数据持久化开销

挑战 4:如何为 KV 数据库提供正确的宕机恢复机制。对于 NVM 不感知的 RDMA 协议,KV 数据库的宕机恢复完全由服务端保障,利用已有的机制即可。对于 NVM 感知的 RDMA 协议,KV 数据库的数据通路(数据操作)和控制通路(元数据和索引管理等)是分离的,数据通路主要由客户端负责,控制通路则由服务端负责,因此需要在客户端和服务端之间建立额外的协调机制,才能保障正确的宕机恢复。

简单地使用现有的 RDMA 通信协议并不能最大化 KV 数据库的性能。除了内存拷贝开销,它们还面临着上述的挑战,限制了 KV 数据库性能进一步提升。在接下来的一节中,将对 BOOM 协议和 BOOM-KV 进行详细介绍。

3 系统设计

本节的目标是构建一个高性能的 RDMA 和

NVM 融合的 KV 数据库,它可以充分利用 NVM 和 RDMA 提供的特性,以最大化提升 KV 数据库的性能。为了实现这一目标,首先确立了以下几点设计原则:(1)允许客户端直接对 NVM 进行 RDMA 操作,避免 DRAM 和 NVM 之间的数据拷贝。(2)充分降低 GET、PUT 请求的端对端延迟,保证只需要一轮网络通信即可完成 GET、PUT 请求。(3)尽可能降低服务端的 CPU 开销,例如减少 NVM 中的垃圾回收。(4)尽可能降低 KV 数据库的持久化开销。(5)提供正确的宕机恢复机制。

根据设计原则,本文提出一种新的 NVM 感知的 RDMA 通信协议,命名为 BOOM 协议,并基于它构建了一个新的 RDMA 和 NVM 融合的 KV 数据库——BOOM-KV。在 BOOM 协议中采用 RC 连接,主要考虑到它可以保证可靠的传输,并且能同时支持 RDMA write 和 RDMA send。每一个 BOOM-KV 的客户端,都需要和服务器建立一条 RC 连接,如图 3所示。在 BOOM-KV 中,采用日志结构对 NVM 空间进行管理,如图 4 所示,持久化日志(persistent log)由多个日志段(log segment)组成了双向链表。一个日志段包含多个日志项(log entry),日志项封装了 KV 数据及相关的元数据。与 Flatstore[11] 等类似,本文采用易失性的 hash 表对 KV 数据索引。

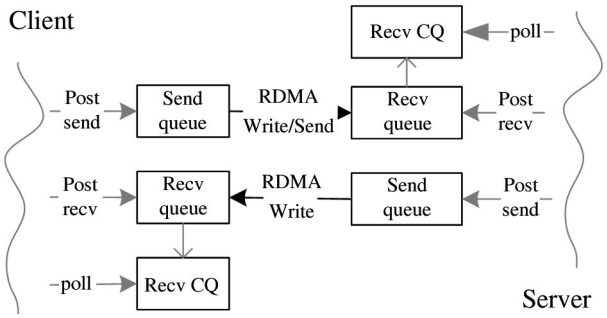

图 3　BOOM-KV 中客户端与服务端的 RC 连接

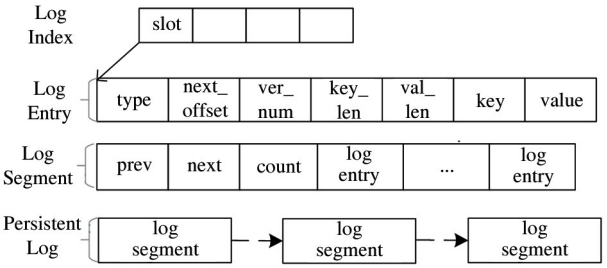

图 4　BOOM-KV 中的日志内存布局

BOOM-KV 与相关工作的对比如表 1 所示。与 RDMP-KV 等工作类似,协议的核心设计思想是让客户端直接对远端的 NVM 进行 RDMA 访问,避免不必要的数据拷贝。BOOM 协议的主要优势在于,无论对于 GET 或 PUT 操作,均只需要一轮网络通信即可完成,降低了KV请求的端对端延迟。接下来将详细介绍 BOOM 协议和 BOOM-KV 中应用的优化思想。

表 1　BOOM-KV 与相关工作的比较

基于 RDMA + NVM 的系统	宕机持久化保证	支持 RDMA-To-NVM	低开销的元数据访问	使用真实的 NVM 评估
Octopus	√	√		
Forca		√		
PMEM-redis + HERD	√		√	
RDMP-KV	√	√		
BOOM-KV	√	√	√	√

3.1　优化 KV 请求的端对端延迟

两阶段的 NVM 感知的 RDMA 协议会导致额外的网络开销,这可能使 KV 数据库无法从零拷贝的优化中获得性能改进,甚至导致性能下降。其原因在于,每一次 KV 请求(无论是 PUT 或 GET),客户端都需要发送一个元数据请求从服务端获取必要的元数据,才能进行随后的 RDMA read/write。然而,元数据请求并非必须,至少可以大幅减少元数据请求的数量,且同样支持客户端直接对远端服务器的 NVM 进行 RDMA 访问。为了实现这一目标,本文提出一个轻量级的元数据获取策略。该策略的核心思想是通过减少元数据请求的数量来降低 KV 请求

的端对端延迟。

对于 PUT 请求,让服务器一次性从 NVM 分配器中分配 N 个持久化对象,而不是每收到一个元数据请求就分配一个持久化对象。然后将这些持久化对象注册为 RDMA 可访问的内存区域,最后将它们的元数据(地址、rkey 等)聚集在一起,一次性地返回给客户端。在概念上,收到这些元数据的客户端就"拥有"了这些元数据所对应的持久化对象,也就意味着客户端可以独占地对它们进行 RDMA write。因此,客户端只需要在首次发送 PUT 请求时,发送一个元数据请求,随后的 $N-1$ 次 PUT 请求都不需要发送元数据请求,一直到第 $N+1$ 个 PUT 请求,客户端将再次发起元数据请求。

为了在一次回复(reply)中聚集多份元数据,服务器必须提前分配多个持久化对象。如果 PUT 请求的 KV 数据大小固定不变,那么只需要预先分配固定大小的持久化对象即可。然而 PUT 请求的 KV 数据大小通常是变化的,因此如何分配大小合适的持久化对象是一个亟需解决的问题。解决这一问题的方案是采用日志结构的思想,让服务器分配一大段地址连续的持久化内存(即日志段),然后将其对应的元数据返回给客户端,客户端随后通过 RDMA write 将 KV 数据以日志项的形式追加到日志段中(实际使用带立即数的 RDMA 操作[12],立即数用于传递日志项的偏移)。考虑到日志段的空间足够大(例如 64 MB),因此足以容纳多个 PUT 请求的 KV 数据。

对于 GET 请求,不采用 RDMA read 的方式让客户端读取远端 NVM 中的持久化对象,相反地,采用 RDMA write 的方式让服务端主动地将 NVM 中的 KV 数据写到 GET 请求指定的内存地址,这样可以彻底避免元数据请求。因此,客户端在发送 GET 请求前,需要提前准备一块足够大的 DRAM 内存,将其注册为 RDMA 可访问的内存区域,将它的地址和 rkey 伴随着 GET 请求发送给服务端,服务器接收到请求后,通过索引定位到 key 对应的日志项,然后将该日志项通过 RDMA write 写回给客户端。在 BOOM-KV 中,客户端发送的 GET 请求以及 PUT 的元数据请求本身使用 RDMA send 发送到了远端的 DRAM 中,而 PUT、GET 的数据操作使用 RDMA write 写到远端的 NVM 或 DRAM 中。

通过利用地址连续的日志段,将 PUT 元数据请求的开销分摊到了多个 PUT 请求中,参与分摊的 PUT 请求数量越多,其开销就越小甚至可忽略不计。而对于 GET 请求,让服务端主动进行 RDMA write,把 KV 数据写入客户端指定的内存地址,完全避免了元数据请求。

3.2 优化服务端垃圾回收的开销

采用日志结构来管理 NVM 空间,一方面可以提供宕机原子性和减少内存碎片化,另一方面可以为客户端提供地址连续的日志段,减少元数据请求的数量,降低 PUT 请求的端对端延迟。然而,日志结构必然会引入垃圾回收机制,过于频繁地进行垃圾回收会消耗大量的服务器 CPU 资源,影响 KV 数据库性能。本文发现,传统的日志结构会产生大量的垃圾数据,其根本原因在于它只允许追加更新,而不允许就地更新(update-in-place)。与追加更新不同,就地更新在将新数据覆写在旧数据之上,可以更充分地利用存储空间。尽管本文采用了日志结构思想,但客户端对其拥有的日志段具有独占的写权限,在满足一定条件的情况下,完全可以自由地对日志段进行更新。为此,本文提出一种混合的 KV 数据更新策略,如图 5 所示,它的核心思想是结合追加更新和就地更新 2 种模式,以减少垃圾数据的产生,达到降低垃圾回收开销的目的。

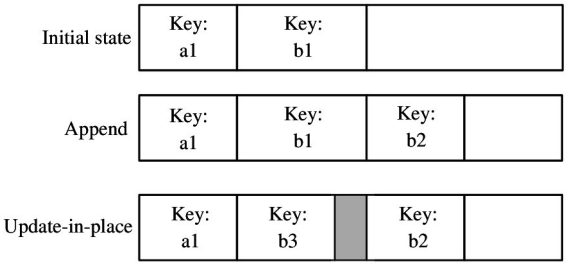

图 5 BOOM-KV 中的混合更新模式

不加限制地进行就地更新会影响 KV 数据库的正确性,因此,客户端在进行就地更新时必须满足以下几点条件。条件 1:客户端不允许覆写最新版本的日志项。如果更新一半时发生宕机,最新的 KV 数据将被腐蚀。条件 2:新 KV 数据所占用的日志项

大小不能超过被覆写的日志项的大小,如果超过被覆写的日志项的大小,会将紧随其后的日志项覆盖掉,造成数据腐蚀。条件 3:就地更新不能对还未完成的 GET 请求造成影响。在 GET 请求还未完成之前,就对 KV 数据所在的日志项进行就地更新,将导致 GET 请求读取到错误的数据。

对于每个 PUT 请求,客户端都需要确定它的更新模式:如果满足就地更新的条件,就进行就地更新,否则进行追加更新。为了判断更新模式,需要在客户端缓存一些辅助信息,这些辅助信息包括日志项初始长度、偏移(相对所在的日志段)和版本号。客户端可以确定日志项长度和偏移等信息,但日志项的版本号由服务端原子地递增,需要从服务端返回后能获得。除此之外,每一个 PUT 请求返回时,服务端还需要确定有没有未完成的 GET 请求(与 PUT 请求的 key 相同),如果有则返回其所访问的日志项的版本号,如果有多个则返回最小的版本号。

本文使用哈希(hash)表缓存辅助信息,hash 表的 key 即 PUT 请求的 key、value 为一个二元组,记作(first, second)。first 存储最近一次更新的辅助信息,second 存储比 first 更前一次的更新的辅助信息。通过查找辅助 hash 表,可以确定当前 PUT 请求的更新模式。判断流程如下。如果不存在与目标 key 对应的二元组,说明还没有对目标 key 进行过 PUT 操作(在该日志段内),必须采用追加更新。如果存在对应的二元组,此时 first 一定不为空。如果 second 为空,说明只对目标 key 进行了一次 PUT 操作,因此目标 key 对应的日志项版本是最新的,仍需采用追加更新。如果 second 不为空,说明对目标 key 至少进行了 2 次 PUT 操作,进一步比较日志项大小来判断是否会导致数据腐蚀,比较日志项版本号来判断是否会对未完成的 GET 请求造成影响,如果前文所述的条件 2、3 均满足,此时可以安全地进行就地更新。

3.3 优化 KV 数据持久化的开销

NVM 感知的 RDMA 协议允许客户端直接对服务端的 NVM 进行 RDMA write,但由于有 CPU cache 存在,使得服务端必须主动地将 cache 缓存的 KV 数据写回到 NVM,才能保证 KV 数据掉电不丢失。一旦客户端的 RDMA 操作完成,服务端的处理线程开始执行写回任务,通过使用 clwb 等指令将 KV 数据写回到 NVM 中,毫无疑问这些写回指令会增加数据持久化的开销。

现有的基于 NVM 和 RDMA 的 KV 数据库要么不提供宕机持久化,要么只能提供低效的宕机持久化机制。例如,RDMP-KV 采用单线程的方法将 KV 数据写回 NVM,这对于大粒度的 KV 数据而言是比较低效的,降低 KV 数据写回开销是进一步提升 KV 数据库性能的关键。为此,本文提出一种协作的 KV 数据写回策略,它的基本思想是:增加 KV 数据写回的并行度,以有效降低 KV 数据写回的开销。通过将写回任务划分多个子任务(按地址范围),并部署多个线程并行地执行这些子任务,可以实现这一目标。然而,简单地增加服务端线程意味着消耗更多 CPU 资源,但本文发现在不增加额外线程的情况下,也同样可以达到增加 KV 数据写回并行度的目的。

在 BOOM-KV 中,每一个客户端都在服务端“独占”一个持久化日志,日志头部的日志段为当前操作的日志段,每个新的 PUT 请求都会直接写到该日志段中。服务端为了尽可能快地处理客户端请求,会部署多个工作线程处理新到来的请求。研究发现,对于大粒度的 PUT 请求,服务端的工作线程会花费大量时间轮询(等待)新请求的到来,对于16 kB 的 PUT 请求,其等待时间占了总延迟的42.6%,浪费了大量的 CPU 资源。与此同时,不同客户端发送的请求到达服务端的时间各不相同,导致服务端部分工作线程已经开始处理请求,而另一部分仍然在等待。

因此,与其让轮询的工作线程白白消耗 CPU 资源,不如利用这段时间帮助其他的工作线程完成它们的数据写回任务。为了减少对锁的竞争,本文为每一个工作线程分配一个专属队列,称拥有该队列的工作线程为主线程,其他工作线程为从线程。当收到 PUT 请求时,主线程负责将写回任务划分若干个子任务,并逐个推入自己的专属队列。此后,主线程会不断地从专属队列中取出子任务并执行,一直到队列为空为止。如果服务端有处于等待状态的从

线程,则会随机地从非专属队列中取出子任务执行,如果期间收到新的 PUT 请求,则开始它们自己的数据写回任务。当写回任务已完成,主线程将继续进行下一步操作,即为新写入的 KV 数据更新索引项,向客户端返回响应等。通过利用空闲的工作线程来并行化已经到来的写回任务,既提升了 KV 数据写回的效率,同时也避免了消耗额外的 CPU 资源。

3.4 BOOM-KV 的宕机恢复机制

在大规模的数据中心,因故障而导致系统宕机的情况常常发生[13],因此如何在系统宕机后对 KV 数据库进行恢复尤为重要。在 BOOM-KV 中,首先要保证 PUT 操作的原子性,使得在恢复过程中能够识别出所有有效的 KV 数据。其次,在扫描日志过程中,要能够正确定位持久化日志中的日志项,一般而言,只需要根据当前日志项的偏移(offset)和大小即可定位下一条日志项,但是因为 BOOM-KV 采用了混合更新的日志结构,使得该方法失效,因而需要额外的机制来解决日志项定位问题。再次,在有大量并发访问的情况下,不同客户端可能对某一个 key 同时进行 PUT 请求,因此它们会分散在不同的持久化日志中,恢复的过程中,需要确定它们的先后顺序,以确定最新的 KV 数据。

为了保证宕机原子性,在每个日志段的头部中引入 8 字节的 count 域,如图 4 所示,它表示在日志段中已生成的日志项的个数。一旦服务端工作线程完成它的日志项的写回任务后,便对 count 进行加 1 操作,并保证其持久化。考虑到 count 域是 8 字节的,因此它的加 1 操作是原子的,进而保证了 PUT 的更新是原子的。需要注意的是,只有在追加更新的情况下,count 会被加 1,如果是就地更新,count 保持不变。考虑到就地更新永远只会覆盖旧数据,因此即便系统宕机导致了部分更新,也不会影响 KV 数据库的恢复。

为了正确定位日志项,在每个日志项中引入 next _ offset 字段,它表示下一个日志项在日志段中的偏移。每次进行追加更新时,都会生成一个新的日志项,它的 next _ offset 初始化为日志项的偏移加上日志项的大小,即使随后又对该日志项进行了就地更新,日志项大小发生改变,但 next _ offset 也不会再修改,因此始终可以正确地定位到下一条日志项,解决了日志项定位的问题。

为了确定不同的持久化日志中的 KV 数据的顺序,在日志项中引入 8 字节的 version _ num 字段,它表示本次 KV 更新的版本号。版本号越大的日志项,其对应的 KV 数据就越新。为了保证每次 PUT 更新都拥有一个不同的、递增的版本号,采用一个原子变量,一旦服务端接收到新的 PUT 请求,就对该原子变量进行加 1 操作,并赋值给相应的日志项的 version _ num。通过版本号确定日志项的新旧顺序。同时,客户端进行就地更新时也需要依赖日志项的版本号才能确保不影响未完成的 GET 请求。

4 实验评估

为了全面评估 BOOM-KV 的性能,本文用 RDMA 原语实现了一个 NVM 不感知的 RPC 协议,并基于该协议构建了一个持久化内存 KV 数据库,并称之为 RDMA-KV(类似 PMEM-redis + HERD)。此外,本文也对当前最先进的工作 RDMP-KV 进行了实现,Li 等人[7]的评测显示 RDMP-KV 的性能表现优于 Forca 和 Octopus,因此不再重复评估。实验评估主要分为 3 个部分。首先,介绍实验环境和配置。其次,分别评估 BOOM-KV 中各项优化技术的效果:(1)BOOM 协议对于减少 KV 请求延迟的效果;(2)混合的日志更新模式对于减少垃圾回收的效果;(3)多线程协作的写回策略对于减少持久化开销的效果。最后,通过 micro-bench、YCSB[14] 测试集对 BOOM-KV、RDMA-KV 和 RDMP-KV 进行综合评测。

4.1 实验环境和配置

本文共使用了 2 台机器进行测试,一台作为客户端,另一台作为服务端。它们均配备了 2 路 IntelXeon 8260 处理器,每个处理器的核心数为 24,主频为 2.4 GHz。每台机器还配备了 128 GB 的 DRAM,以及 40 Gbps 的 Mellanox ConnectX-5 RNIC。在服务端配有一块 128 GB 的 Optane DCPMM 用于存储 KV 数据。操作系统版本为 CentOS 7.8.2003,编译器版本为 gcc 4.8.5。为了减少 NUMA 架构对测试的影响,所有的实验都绑定到 NUMA 0 节点。

4.2 BOOM-KV 请求延迟的优化效果

本小节将评估 BOOM 协议在降低 KV 请求延迟方面的效果。如图 6 和 7 所示,对于不同粒度的 KV 数据,BOOM-KV 的请求延迟均有明显下降,其中 PUT 请求延迟最大下降 38%,GET 请求最大下降 37%。特别地,越是粒度小的 KV 请求,BOOM 协议在降低 KV 请求延迟上的效果就越明显,例如对于 1 kB 的 PUT 请求,延迟降低了31.6%,而对于16 kB的 PUT 请求,延迟只降低了17.5%。出现这种现象的原因是,对于小粒度的 KV 请求,元数据请求的开销占比较大,数据拷贝的开销相对较小,因此 BOOM 协议通过减少元数据请求可以显著降低延迟。对于大粒度的 KV 请求,数据拷贝的开销占比更大,而元数据请求的开销占比变小,因此 BOOM 协议的优化效果相对地被弱化了一些。

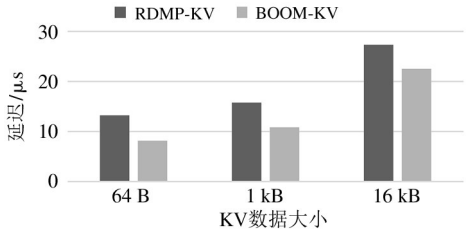

图 6 PUT 请求的端对端延迟

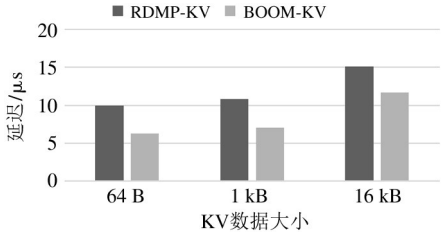

图 7 GET 请求的端对端延迟

进一步对 BOOM-KV PUT 的延迟进行敏感性分析。如图 8 所示,采用 1 kB 大小的 PUT 请求进行测试,同时配置不同的日志段大小。由图可知,随着日志段大小逐渐增大,BOOM-KV PUT 的吞吐量越高,延迟越低。

4.3 BOOM-KV 垃圾回收的优化效果

本小节将评估 BOOM-KV 在降低垃圾回收开销方面的效果。BOOM-KV 中采用混合更新模式,通过追加更新和就地更新相结合来减少垃圾数据的产生。为了评估 BOOM-KV 的混合更新模式的效果,本实验使用了均匀分布的负载和 Zipfian 分布的负载,负载为 100% 的 PUT 操作,总大小为 1 GB。分别测试了均匀负载和 Zipfian 负载(在不同倾斜度下)所占用的 NVM 空间大小。如图 9 所示,对于均匀分布的负载,其占用的 NVM 空间最大,与负载的大小一致;对于 Zipfian 分布的负载,随着倾斜度变大,负载所占用 NVM 空间逐渐减小。对于倾斜度 0.99 的负载,负载占用的 NVM 空间减少了 57.2%;对于倾斜度为 1.1 的负载,占用的 NVM 空间减少了 76.5%。负载倾斜度越大,使得就地更新所占比例越大,也就意味着有更多的日志项被重复利用,从而减少了对 NVM 空间的占用。

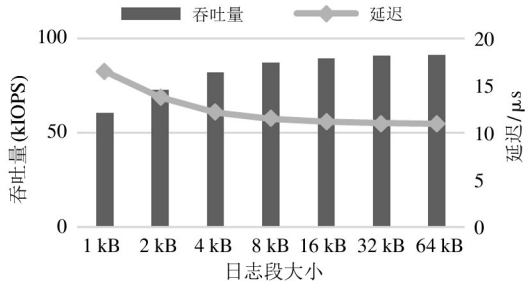

图 8 BOOM-KV PUT 请求的性能与日志段大小的关系

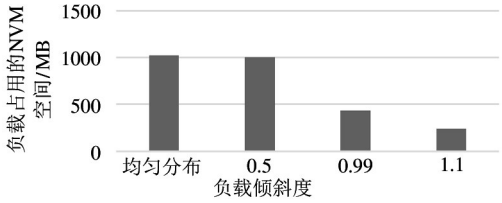

图 9 混合更新模式在不同负载下的优化效果

日志段的大小也会影响混合更新模式对于垃圾回收的优化效果。如图 10 所示,随着日志段增大,负载占用的 NVM 空间越少,优化效果就越好。

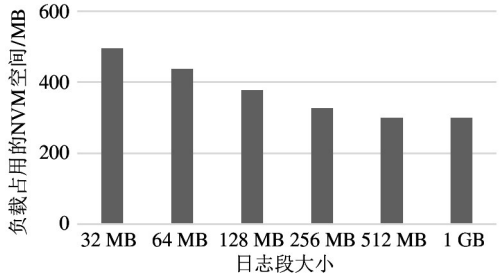

图 10 日志段大小对于混合更新模式的影响

4.4　BOOM-KV 持久化开销的优化效果

本小节将评估BOOM-KV在降低持久化开销方面的效果。为了评估该策略的效果,本文部署了 2 个客户端以及同等数量的服务端工作线程后,分别测试了 BOOM-KV 在启动或关闭多线程协作策略下的 PUT 性能,同时也考虑了该策略在搭配不同写回指令(clflush、clwb)时的优化效果。如图 11 所示,该测试使用 clflush 指令写回数据,可以观察到 PUT 吞吐量均有一定提升,对于4 kB、8 kB 和 16 kB 的 KV 数据,吞吐量分别提升了 25%、23.5% 和14.8%。如图 12 所示,该测试使用 clwb 指令写回数据,对于 4 kB 的 KV 数据,PUT 吞吐量基本没有变化,对于 8 kB和 16 kB,吞吐量分别提升了 6.4% 和 24.9%。整体而言,多线程协作写回策略在搭配 clfush 指令时,优化效果更加显著,这是因为 clflush 指令本身串行执行的,而 clwb 本身是并行执行的,因此优化效果相对差一些。

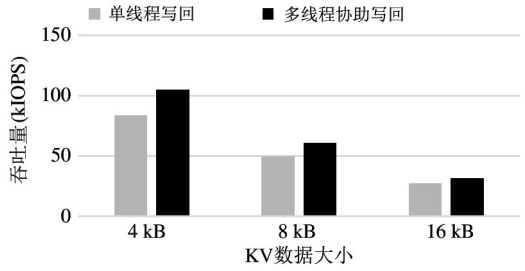

图 11　BOOM-KV PUT 请求的吞吐量(clflush)

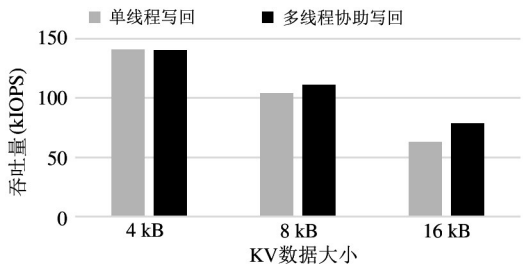

图 12　BOOM-KV PUT 请求的吞吐量(clwb)

4.5　BOOM-KV 的综合性能评估

本小节分别使用 micro-bench 和 YCSB 测试集对 BOOM-KV、RDMA-KV 和 RDMP-KV 进行综合的性能评估和对比。用 micro-bench 测试各 KV 数据库在不同 KV 粒度下,对于随机读、写负载的平均延迟,用 YCSB 测试各 KV 数据库在不同 KV 粒度下,对于读写混合、有倾斜的负载的平均延迟。除此之外,通过 micro-bench 评测各 KV 数据库在多线程下的性能表现。

图 13 展示的是各 KV 数据库 PUT 请求的平均延迟。由图可知,与 RDMP-KV 相比,BOOM-KV 在所有 KV 粒度上,PUT 请求延迟均有下降,特别是对于小粒度的 KV 数据,如 KV 数据大小为 32 B 时,PUT 延迟下降幅度最大,达到了 42%。总体而言,BOOM-KV 的 PUT 延迟相比 RDMP-KV 降低了 10% ~42%。与 RDMA-KV 相比,对于小粒度 KV 数据(< =1 kB),BOOM-KV 的 PUT 延迟有所上升,对于大粒度 KV 数据(>1 kB),PUT 延迟下降了 5% ~13%。图 14 展示的是各 KV 数据库 GET 请求的平均延迟。由图可知,与 RDMP-KV 相比,BOOM-KV 在任意 KV 粒度上,GET 延迟均有较大幅度的下降,特别是对于小于等于 4 kB 粒度的 KV 数据,GET 延迟均下降了 30% 以上,其中 32 B 下降了 41%。与 RDMA-KV 相比,对于小粒度的 KV 数据(< =1 kB),BOOM-KV 的 GET 延迟下降 8% 左右,对于大粒度 KV 数据(>1 kB),GET 延迟下降了 38% ~44%。

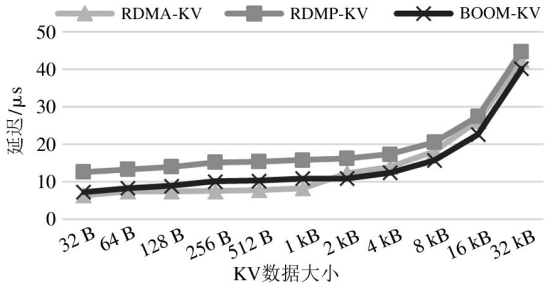

图 13　PUT 请求的平均延迟

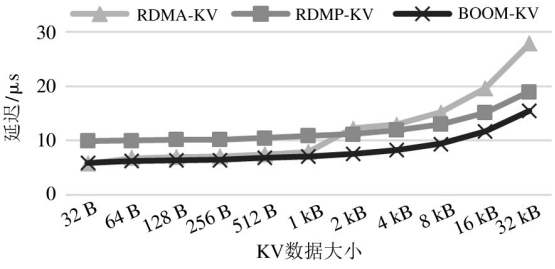

图 14　GET 请求的平均延迟

图 15、16 和 17 分别是通过 YCSB 的测试各 KV 数据库的平均延迟。对于 YCSB-A,BOOM-KV 的延迟相比 RDMP-KV 下降了 13.4% ~41.5%,相比

RDMA-KV 下降 21.2% ~25.4%(>1 kB)。对于 YCSB-B,BOOM-KV 的延迟相比 RDMP-KV 下降了 18.2% ~41.7%,相比 RDMA-KV 下降了2.6% ~39.2%。对于 YCSB-C,BOOM-KV 的延迟相比 RDMP-KV 下降了 18.4% ~41.5%,相比 RDMA-KV 下降了 4.7% ~39.5%。

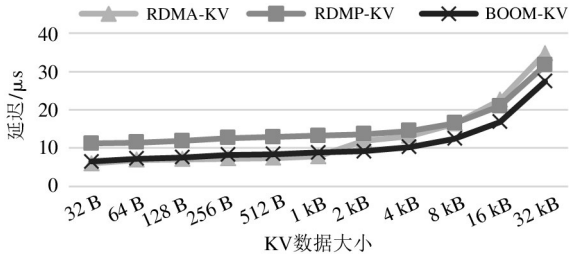

图 15 YCSB-A(50%:GET,50%:PUT)

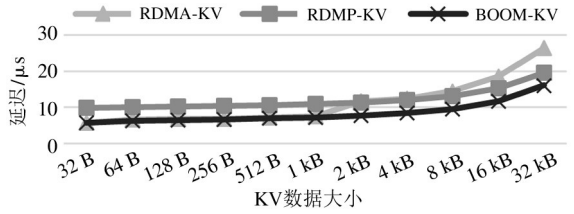

图 16 YCSB-B(95%:GET,5%:PUT)

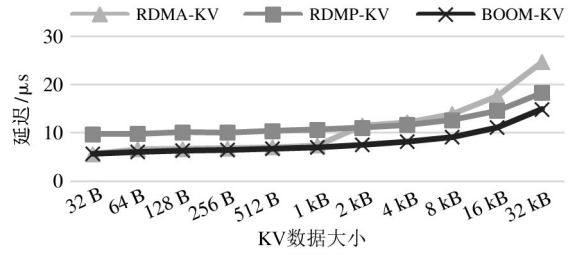

图 17 YCSB-C(100%:GET)

图 18、19 是用多个客户端线程测试的 PUT、GET 吞吐量,服务端线程数量与客户端数量一致,KV 数据配置为 2 kB。对于 PUT 请求,BOOM-KV 的吞吐量相比 RDMP-KV 提升了 40.5% ~46.7%,相比 RDMA-KV 提升了 7.2% ~22.1%。对于 GET 请求,BOOM-KV 的吞吐量相比 RDMP-KV 提升了 56.7% ~61.7%,相比 RDMA-KV 提升了 47.5% ~48.9%。

通过上述实验,可以观察到无论是 PUT、GET 请求还是大、小粒度的 KV 数据,BOOM-KV 的性能均胜过 RDMP-KV。对于小粒度的 KV 数据,BOOM-KV 的性能与 RDMA-KV 相差无几,对于大粒度的 KV 数据,BOOM-KV 要优于 RDMA-KV。同时 BOOM-KV 具有较好的多线程扩展性。

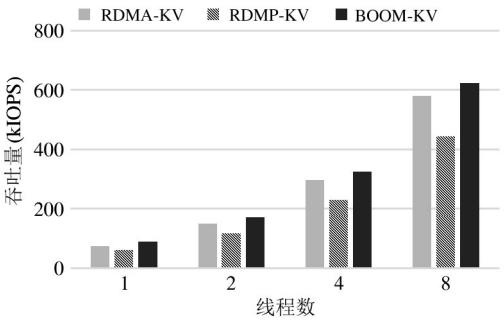

图 18 PUT 请求的多线程扩展性

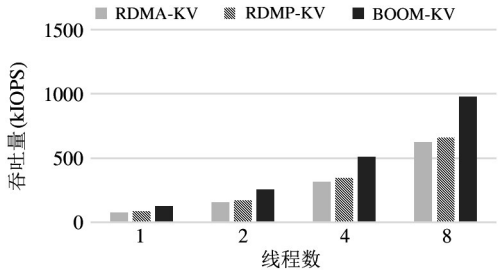

图 19 GET 请求的多线程扩展性

5 相关工作

本工作在现有的基于 RDMA-NVM 的系统基础上,继续探索如何改进系统性能[5-7,15]。Octopus[5] 基于 RDMA 和 NVM 构建了一个分布式文件系统,它将内部的文件系统机制与 RDMA 特性紧密结合。对于元数据操作,它通过基于 RDMA 优化的 RPC 进行。对于数据操作,它提供一个共享的持久化内存池抽象,并直接通过 RDMA 进行数据 I/O,以减少不必要的内存拷贝。Forca[6] 作为 RDMA-to-PM 的框架,它采用了服务器旁路(server-bypass)来减轻服务端的负载,同时通过日志结构消除就地更新,消除单边操作的数据竞争。尽管 Forca 声称提供宕机一致性保证,但是它不会主动将 cache 缓存数据写回 NVM,因此无法保证宕机持久化。RDMP-KV[7] 是一个基于 RDMA 和 NVM 的 KV 数据库,与本工作直接相关。它采用了混合的 server-reply/server-bypass 方法,在 server-reply 阶段采用 RPC 协议访问元数据,在 server-bypass 阶段采用 RDMA 进行数据操作,避免不必要的数据拷贝。为了保证数据持久性,RDMP-KV

借助服务端的辅助对 cache 缓存数据进行写回。为了保证数据一致性,它还提供了一个轻量级的一致性策略。BOOM-KV 与 Octopus、Forca 以及 RDMP-KV 均能通过 RDMA 直接访问 NVM 资源,它们主要的区别在于,BOOM-KV 极大地减少了元数据请求的开销,而其他系统无论读、写都需要先通过 RPC 获取元数据,然后才能进行数据操作,增加了 KV 请求的延迟。此外,Forca 没有考虑宕机持久化的问题。

6 结 论

本文首先分析了现有的基于 RDMA 的通信协议所存在的问题,并指出构建高效的基于 NVM 和 RDMA 的 KV 数据库所面临的关键挑战。为了降低 KV 请求的端对端延迟,本文提出了 BOOM 协议——一个全新的 NVM 感知的 RDMA 协议,通过减少元数据请求,BOOM 协议可以显著降低 KV 请求的端对端延迟。为了降低服务端 CPU 的开销,提出一个混合的数据更新模式,该模式允许客户端通过 RDMA 进行就地更新或追加更新,大幅减少垃圾数据的产生。为了降低数据持久化的开销,本文提出一个协作的数据持久化策略,该策略通过多线程互相协助提升数据写回的并行性,有效地降低了大粒度 KV 数据的持久化开销。最后,通过 micro-bench 测试集逐个验证了 BOOM-KV 中各优化策略的效果,并通过 YCSB 测试集将 BOOM-KV 与最新的研究工作对比,测试结果表明 BOOM-KV 在请求延迟和吞吐上均比 RDMP-KV 表现更好。

参考文献

[1] CHEN Y, LU Y, LUO S, et al. Survey on RDMA-based distributed storage systems[J]. Journal of Computer Research and Development, 2019, 56(2): 227.

[2] MITCHELL C, GENG Y, LI J. Using one-sided RDMA reads to build a fast, CPU-effcient key-value store[C]// 2013 USENIX Annual Technical Conference. San Jose: USENIX Association, 2013: 103-114.

[3] KALIA A, KAMINSKY M, ANDERSEN D G. Using RDMA efficiently for key-value services[C]// Proceedings of the 2014 ACM Conference on SIGCOMM. Chicago: Association for Computing Machinery, 2014: 295-306.

[4] KALIA A, KAMINSKY M, ANDERSEN D. Datacenter RPCs can be general and fast[C]// The 16th USENIX Symposium on Networked Systems Design and Implementation. Boston: USENIX Association, 2019: 1-16.

[5] LU Y, SHU J, CHEN Y, et al. Octopus: an RDMA-enabled distributed persistent memory file system[C]//2017 USENIX Annual Technical Conference. Santa Clara: USENIX Association, 2017: 773-785.

[6] HUANG H, HUANG K, YOU L, et al. Forca: fast and atomic remote direct access to persistent memory[C]// 2018 IEEE 36th International Conference on Computer Design (ICCD). Orlando: IEEE, 2018: 246-249.

[7] LI T, SHANKAR D, GUGNANI S, et al. RDMP-KV: designing remote direct memory persistence based key-value stores with PMEM[C]// SC20: International Conference for High Performance Computing, Networking, Storage and Analysis. Atlanta: IEEE, 2020: 1-14.

[8] RUMBLE S M, KEJRIWAL A, OUSTERHOUT J. Log-structured memory for DRAM-based storage[C]// The 12th USENIX Conference on File and Storage Technologies. Santa Clara: USENIX Association, 2014: 1-16.

[9] XU J, SWANSON S. NOVA: a log-structured file system for hybrid volatile/non-volatile main memories[C]// The 14th USENIX Conference on File and Storage Technologies. Santa Clara: USENIX Association, 2016: 323-338.

[10] HU Q, REN J, BADAM A, et al. Log-structured non-volatile main memory[C]//2017 USENIX Annual Technical Conference. Santa Clara: USENIX Association, 2017: 703-717.

[11] CHEN Y, LU Y, YANG F, et al. FlatStore: an efficient log-structured key-value storage engine for persistent memory[C]// Proceedings of the 25th International Conference on Architectural Support for Programming Languages and Operating Systems. Lausanne: Association for Computing Machinery, 2020: 1077-1091.

[12] MACARTHUR P, RUSSELL R D. A performance study to guide RDMA programming decisions[C]//2012 IEEE 14th International Conference on High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems. Liverpool: IEEE, 2012: 778-785.

[13] BARROSO L A, HÖLZLE U. The datacenter as a computer: an introduction to the design of warehouse-scale machines[J]. Synthesis Lectures on Computer Architecture, 2009, 4(1): 1-108.

[14] COOPER B F, SILBERSTEIN A, TAM E, et al. Benchmarking cloud serving systems with YCSB[C]// Proceedings of the 1st ACM symposium on Cloud computing, Indianapolis: Association for Computing Machinery, 2010: 143-154.

[15] 黄海鑫. 基于 RDMA 的 NVM 高并发单边访问机制设计与实现[D]. 上海:上海交通大学计算机科学与工程系, 2019.

BOOM-KV: an RDMA-enabled high-performance NVM key-value store

LI Wenjie[*,**,***], JIANG Dejun[*,**,***], XIONG Jin[*,***], BAO Yungang[*,**,***]

([*] Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190)

([**] School of Computer Science and Technology, University of Chinese Academy of Sciences, Beijing 100049)

([***] Peng Cheng Laboratory, China, Shenzhen 518055)

Abstract

With the commercialization of Intel Optane DC persistent memory module (DCPMM) and the decrease in remote direct memory access (RDMA) hardware costs, the design of key-value (KV) stores that integrate non-volatile memory (NVM) and RDMA faces new opportunities and challenges. The key to building a KV store based on NVM and RDMA lies in designing an efficient communication protocol. Unfortunately, existing work either uses an NVM-oblivious RDMA protocol or an inefficient NVM-aware RDMA protocol, which prevents them from maximizing the performance of KV stores. This paper proposes the BOOM protocol, a new type of NVM-aware RDMA protocol. Compared to NVM-unaware protocols, the BOOM protocol allows direct RDMA operations on remote NVM, eliminating redundant data copies. Compared to existing NVM-aware protocols, it can significantly reduce metadata requests and reduce end-to-end latency of KV requests. Based on the BOOM protocol, BOOM-KV is built and further optimized for server central processing unit (CPU) utilization and crash persistence. Finally, BOOM-KV is compared with the state-of-the-art protocols , and results show that BOOM-KV can significantly reduce request latency with a maximum reduction of 42% in PUT latency and 41% in GET latency, exhibiting good scalability.

Key words: non-volatile memory(NVM), remote direct memory access(RDMA), key-value (KV) store