持久性内存和 RDMA 技术综述

秦庚辰[1)]

[1)](华中科技大学计算机学院 武汉市 中国 430074)

摘 要 随着数据规模与日俱增，典型的数据中心应用对吞吐率、扩展性、延迟等性能指标提出了更为严苛的要求。持久性内存、RDMA 等新型硬件的出现为构建高效的存储系统提供了新的机遇，同时也推动了存储软件设计的革新。一方面，持久性内存、RDMA 等新型硬件的低延迟特性导致现有存储系统软件开销占比极高，从而无法充分发挥硬件性能优势；另一方面，持久性内存和 RDMA 硬件具有与传统器件不同的内部特征，这些硬件特性如果未被充分考虑，还将进一步造成应用程序性能低效等问题。本文重点讨论了持久性内存和 RDMA 相结合的优势和面临的问题，并着重介绍了两个模型 Rowan 和 FlatStore.前者实现了高性能并很大程度上消除了设备级别写放大的现象，后者实现了高吞吐率、低延迟和多核扩展性等性能目标

关键词 RDMA 持久性内存 Rowan FlatStore

A review of persistent memory and RDMA technology

Gengchen Qin[1)]

[1)](Huazhong University of Science and Technology, Wuhan, China,430074)

Abstract As the scale of data increases day by day, typical data center applications place more stringent requirements on performance indicators such as throughput, scalability, and latency. The emergence of new hardware such as persistent memory and RDMA provides new opportunities for building efficient storage systems, and also promotes innovation in storage software design. On the one hand, the low-latency characteristics of new hardware such as persistent memory and RDMA result in a very high proportion of software overhead in existing storage systems, making it impossible to fully utilize the hardware performance advantages; on the other hand, persistent memory and RDMA hardware have different characteristics from traditional devices. If these hardware characteristics are not fully considered, they will further cause problems such as inefficient application performance. This article focuses on the advantages and problems of combining persistent memory and RDMA, and focuses on two models, Rowan and FlatStore. The former achieves high performance and largely eliminates device-level write amplification, while the latter Achieved performance goals of high throughput, low latency, and multi-core scalability.

Key words RDMA Persistent Memory Rowan FlatStore

1 引言

持久性内存（persistent memory，PM）能够像磁盘一样持久地存储数据，具有接近 DRAM 的性能，同时还能提供远高于 DRAM 的存储密度。PM 通过内存总线与 CPU 互连，因此，CPU 可以按字节粒度访问 PM。在网络传输方面，远程直接内存访问（remote direct memory access，RDMA）具有高带宽、低延迟的网络传输特性。同时，RDMA 可以在接收方 CPU 不参与的情况下直接访问远端服务器的内存，从而有效降低 CPU 资源的消耗。持久性内存和 RDMA 技术的出现为构建大容量、高性能和低延迟的内存存储系统带来了希望。

2 背景

2.1 持久性内存

经典的计算机存储器层次结构是由易失性内存（包括处理器缓存、DRAM 等）和持久性外存（包括机械磁盘、固态硬盘等）构成。在如图 2.1 所示的存储金字塔中，存储层级越高，其距离处理器越近，容量越小、访问速度越高，单位容量价格也就越贵。相变存储器（phase change memory，PCM）、电阻式存储器（resistive random-access memory，ReRAM）[1] 等新型持久性内存的出现则打破了上述存储层级。持久性内存像 DRAM 一样通过内存总线与处理器直接相连，处理器可以通过内存指令按字节粒度访问持久性内存，且访问延迟与 DRAM 十分接近。另外，持久性内存还能像外存设备一样持久地存储数据，即使在系统掉电时数据也不会丢失。因此，持久性内存属于全新的一个存储层级。

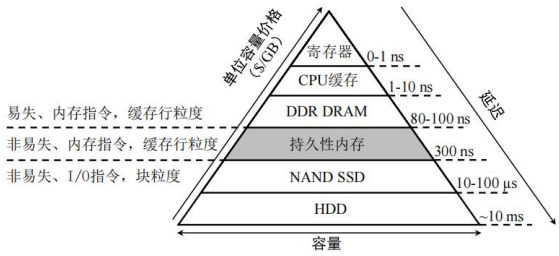

图 2.1 计算机系统存储金字塔结构示意图

为优化程序性能，现代处理器和编译器均会将程序指令进行乱序执行，并且处理器高速缓存也会乱序地将脏缓存行逐出到持久性内存。因此，计算机并不会按照程序员指定的顺序将数据写入到持久性内存中，这会导致数据持久性和一致性等方面的问题。为此，程序员需要显式地在需要立即持久化的数据之后增加缓存刷写指令（如 clflushopt、clwb 等），强制将数据写入到持久性内存中，并通过内存屏障指令（如 mfence）明确写操作之间的顺序，防止指令乱序执行。

2.2 RDMA 技术

RDMA 是一种区别于传统 TCP/IP 的网络传输协议，它允许本地 CPU 绕过操作系统直接读写远端节点内存，该过程无需远端 CPU 参与。目前有 3 类网络架构支持 RDMA 技术，分别为 IB（InfiniBand）、RoCE（RDMA over converged ethernet）和 iWARP（internet wide area RDMA protocol）。其中，RoCE 和 IB 具有相同的上层网络协议栈，RoCE 在数据链路层完全兼容以太网，而 iWARP 则保留了完整的 TCP/IP 协议栈。RDMA 通信链路可以被配置为 3 种模式，分别为可靠连接（reliable connection，RC）、不可靠连接（unreliable connection，UC）和不可靠数据报（unreliable datagram，UD）。其中，UD 支持“一对一”和“一对多”数据传输，而 RC 和 UC 仅支持“一对一”数据传输。

RDMA 可通过两类原语访问远端内存，分别为双边原语（two-sided verbs）和单边原语（one-sided verbs）。其中，双边原语包括 recv 和 send 两种，它们类似于套接字编程中的收发原语。在发送消息之前，接收方需要提前调用 recv 原语来指定接收消息的存放地址，因此，双边原语的完成需要收发双方共同参与。单边原语包含 read、write 以及相应的变种（例如 write-imm、原子操作等）。这类原语可以在远端 CPU 不介入的情况下直接读取或更新远端内存。不同原语在不同的通信链路模式下具有不同的支持程度，详细情况如表 2.1 所示。

表 2.1 不同连接模式对 RDMA 原语的支持及最大传输单元

原语类型	send/recv	write[-imm]	read	原子操作	最大传输单元
RC	✓	✓	✓	✓	2 GB
UC	✓	✓	✗	✗	2 GB
UD	✓	✗	✗	✗	4 KB

2.3 持久性内存存储系统发展趋势

随着各类硬件技术的迅猛发展，新一代存储和网络设备已经能够提供亚微妙级访问延迟和超过 10 GB/s 的访问带宽。然而，传统存储软件由于其老旧的设计已经逐渐难以充分发挥新型硬件的性能优势，系统研究人员开始密切关注新一代内存级存储系统的设计，本文将持久性内存存储系统的变化趋势总结为以下几点：

（1）用户态直接访问。

传统外存存储及网络设备均由操作系统统一接管，应用程序通过标准化系统调用接口陷入到内核访问硬件设备，操作系统则通过中断机制与外部

设备进行交互。由于持久性内存和 RDMA 的访问延迟极低，传统的软件栈则显得过于低效，其毫秒级的软件调度延迟相比于硬件设备的延迟高出一至多个数量级，高速硬件的低延迟特性难以被充分发挥。因此，用户态直接访问模式逐渐受到广泛关注。例如，在管理持久性内存时，应用程序通常将其存储空间通过内存映射机制导入到用户态地址空间，而存储软件可以在用户态直接管理持久性内存设备。英特尔为此开发了持久性内存编程库 PMDK（persistent memory development kit），该编程库可以在用户态直接访问持久性内存空间，并向应用程序提供完整的存储服务。因此，应用程序不再需要通过陷入到内核访问存储设备，访问延迟大幅降低。类似地，DPDK（data plane development kit）提供了用户态网络 I/O 栈，并使用用户态轮询机制 (polling) 处理数据包。在收到数据包时，经 DPDK 重载的网卡驱动不会通过中断通知 CPU，而是直接将数据包写入到内存，从而节省了大量的 CPU 中断时间和内存拷贝时间。

（2）细粒度存储管理。

机械硬盘、固态硬盘等传统存储器件均为块设备，其最小访问粒度为扇区（通常为 512 B）或页（通常为 4 KB）。因此，传统存储系统常采用粗粒度的空间管理方式。然而，应用程序往往生成较多的小写，例如，文件系统元数据更新和键值存储系统索引结构更新的粒度通常仅为几个或数十个字节，这将造成严重的写放大问题。持久性内存可以按字节粒度进行寻址，因此，针对持久性内存而设计的存储系统将精细化的存储管理作为其重要设计方式，这主要体现在以下两方面：首先，字节寻址特性可以帮助存储系统降低写放大问题。例如，加州大学圣地亚哥分校提出的 NOVA 文件系统[2] 将文件元数据组织为独立的日志结构，每次元数据修改仅需将更新内容追加到日志尾部即可，从而最大程度降低了元数据更新带来的写放大问题。其次，存储系统可以利用字节寻址特性对存储的数据进行紧密排布，从而更好地利用存储空间。例如，清华大学提出的 LSNVMM[3] 将持久性内存空间组织为一个大日志，所有更新的内容均追加至日志尾部，从而消除了传统空间管理方式带来的碎片问题。

（3）存储与网络协同设计。

传统的分布式存储系统均采用了模块化的软件设计理念，存储模块和网络传输模块通过函数调用进行交互。然而，随着 RDMA 等新型硬件技术的出现，存储与网络协同设计成为优化系统性能的重要手段。例如，在键值存储系统中，系统设计人员利用 RDMA 的远程直接访问特性进行远程数据查询，从而实现了在服务端不参与的情况下直接读取远端数据项[4]。再例如，近年来兴起的新型可编程网卡及可编程交换机具有一定的可编程能力和少量的存储空间，系统开发人员可以利用这些特性在交换机或网卡缓存部分热点数据[5]，并在网卡或交换机中增加额外的执行逻辑以重构分布式协议[6]。

2.4 存在的问题

（1）存储软件开销高

由于持久性内存的访问延迟极低，因此高效的存储软件栈设计十分重要。例如，Linux 4.2 版本开始支持直接访问模式（direct access，DAX）。该模式下的文件系统可以将持久性内存空间直接映射到用户态，从而移除了传统 DRAM 页面缓存造成的冗余数据拷贝。一些专门为持久性内存而设计的文件系统还移除了设备驱动层、块设备层等，从而对存储栈进行进一步精简。然而，将持久性内存文件系统部署在内核态依旧无法避免系统调用陷入内核产生的现场切换开销以及虚拟文件系统（virtual file system，VFS）造成的软件开销。实验显示，单个文件操作在系统调用和 VFS 层的执行时间占比平均可达 25%。系统架构难扩展。新型持久性内存及 RDMA 网卡提供了极高的访问带宽，但是，单个 CPU 核心还不足以达到硬件设备的峰值带宽。近年来，基于 NUMA（non uniform memory access）架构的多核服务器已经被广泛使用，同时，多核并行技术在近年来也逐渐得到关注，相应地，存储系统软件的多核扩展性亦成为重点的设计方向。然而，操作系统作为一个已经迭代了数十年的产物，依旧存在大量的扩展性问题。例如，当多个线程在同一个目录下并发地创建文件时，VFS 会将该目录锁住，从而造成严重的扩展性问题。

（2）硬件特性难感知

在存储器件方面，英特尔发布的傲腾持久性内存增加了一定容量的 DRAM 缓冲区以提升性能，因此，新写入的数据将先被暂存到缓冲区，而多次写入的数据需要经过合并以后再以 256 B 的粒度写入到持久性内存介质中。因此，持久性内存的

实际写入粒度高于其字节寻址粒度。然而，应用程序无法感知这一特性，往往因不对齐写或小写造成额外的写放大以及读后写开销。在 NUMA 架构下，持久性内存还从存在严重的近远端访问不对称的问题。实验显示，多个线程并发写入远端持久性内存设备将导致总体性能急剧下降。在网络设备方面，RDMA 网卡采用了无内存架构，其片上缓存空间极为有限。在网络连接数量变多之后，连接信息将无法完全缓存在 RDMA 网卡中，从而出现严重的缓存争用现象，总体吞吐率将随着连接数量的增多而降低。因此，不合理的硬件使用方式不但不能充分发挥其性能优势，反而还会导致性能抖动、效率低下等问题。

（3）吞吐延迟难兼顾

相比于外存设备和传统以太网，高速持久性内存和 RDMA 网络显著提升了系统的总体吞吐率，并降低了延迟。然而，追求极致的吞吐率、延迟及扩展性则需要更加精巧的软件调度与权衡。例如，批量处理技术将多次网络或存储请求进行合并，从而降低对硬件设备的访问次数。该方法能够充分利用硬件设备的带宽，然而，批量操作势必会对先到达的请求进行延迟处理，从而响应延迟也会同步上升。再例如，在数据中心中，客户端请求通常展现出“高扇出”的特性，因此，数据中心还会对各个服务器节点的尾延迟和吞吐率同时提出严苛的要求。系统开发人员在协调上述性能指标时面临极大挑战。

3 Rowan：高效的 RDMA 抽象

[7]在此文提出了 Rowan，一种高效的 RDMA 抽象，用于处理 PM KVS 中的复制写入； 它聚合来自不同服务器的并发远程写入，并且将这些写入按顺序写入 PM（因此 DLWA 较低）以及单方面（因此低延迟）方式。 认识到 Rowan 使用现成的 RDMA NIC。 此外，此文构建了 Rowan-KV，使用 Rowan 进行复制的日志结构 PM KVS。评估表明，在写入密集型工作负载下，相比 PM KVS 使用 RPC 和 RDMA WRITE 进行复制，Rowan-KV 将吞吐量提高了 1.22 倍和 1.39 倍分别将中位 PUT 延迟降低 1.77 倍和 2.11 倍，同时很大程度上消除了 DLWA。

3.1 Rowan 整体框架

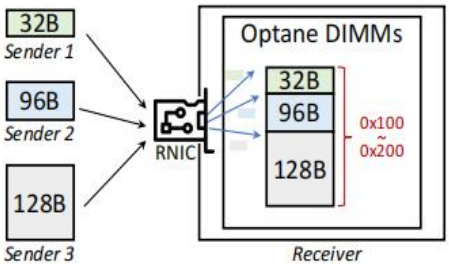

图 3.1 Rowan 实例

图 3.1 显示了一个 Rowan 实例。一个 Rowan 实例与一个接收者和一组发送者相关联。 发送方同时向注册了大 PM 区域的接收方发出写入。 接收方 RNIC 将这些写入顺序地登陆到 PM 区域，并最终将 ACK 返回给发送方。

Rowan 抽象具有以下优点。 首先，通过将并发远程小写入转换为单个写入流，Optane DIMM 中的 XPBuffer 可以轻松地将它们组合为 256B XPLine 写入，从而在很大程度上消除了 DLWA。

其次，由于所有数据操作均由接收端 RNIC 执行，不涉及接收端 CPU，因此 Rowan 享有像 RDMA WRITE 一样的低延迟和高 CPU 效率的好处。 此外，与 CPU 相比，RNIC ASIC 可以提供极高的吞吐量。

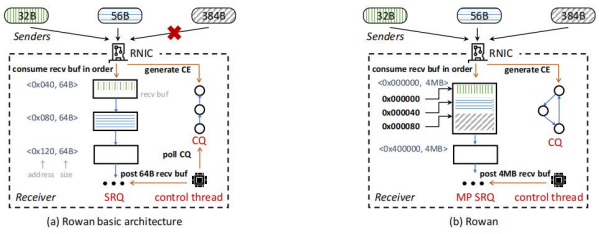

图 3.2 使用共享队列的 Rowan 架构

Rowan 实现的基本架构。 Rowan 使用可靠连接 (RC) QP 将传输可靠性委托给 RNIC。 创建一个与所有 QP 关联的共享接收队列（SRQ）； 因此，RNIC 可以将来自不同远程 QP 的 SEND 数据登陆到同一接收队列。 在接收器中，保留一个专用线程，即控制线程，来执行控制路径任务； RNIC 执行数据路径任务。具体来说，控制线程将 PM 区域分割为固定大小（例如，图 3.2（a）中的 64B）缓冲区，并将这些缓冲区（使用 RECV）以递增地址顺序发送到 SRQ 中。 发送方将写入封装成

SEND 请求，并将其发送给接收方； 每个 SEND 后面都会有一个 READ 以保持持久性。 当接收到 SEND（随后是 READ）时，接收方 RNIC 弹出 SRQ 中的第一个缓冲区，DMA SEND 的数据进入缓冲区，生成一个完成条目（CE）到 SRQ 的 CQ，最后返回一个 ACK 给发送方。 通过这种方式，来自不同发送方的写入可以合并到 PM 上的相同 XLine 中，从而减轻 DLWA。

处理可变大小的写入。 当 SEND 的数据大小大于 SRQ 中的第一个缓冲区时，RNIC 无法容纳它并会触发错误 CE。例如，在图 3.2(a) 中，使用 64B 接收缓冲区，无法处理 384B 写入。 如果我们为 SRQ 使用大于 256B 的缓冲区大小来支持相对较大的写入，则来自不同发送方的小写入将不会组合到相同的 XPLine 中，从而破坏了 Rowan 抽象的好处。幸运的是，当前的 RNIC（例如 ConnectX-4/5/6）支持一种新型 RQ，称为多数据包接收队列 (MP RQ)。 在 MP RQ 中，每个接收缓冲区可以容纳多个 SEND 请求。 我们需要为 MP RQ 定义一个步幅（例如 64B）。 当接收到 SEND 时，RNIC 将数据附加到正在使用的接收缓冲区，并且起始地址是步幅对齐的。 如果没有足够的空间，RNIC 从 MP RQ 中弹出一个新的接收缓冲区来使用,如图 3.2(b)中所示。

3.2 Rowan-键值设计

Rowan-KV 是一个使用 Rowan 进行主备份复制的 PM KVS。它有两个主要设计目标。

• 低延迟。Rowan-KV 利用单侧 Rowan 来消除复制过程中备份时的软件开销。

• 低 DLWA。 Rowan-KV 采用日志结构的方法来管理来自本地 CPU 和远程 CPU 的 PM 写入。 对于前者，每个线程都会在其本地日志中附加数据。 对于后者，Rowan 将复制写入合并到单个备份日志中。 因此，Optane DIMM 仅接收少量写入流，并且可以有效地将相邻的小写入组合到 XPLine 中，从而减轻 DLWA。

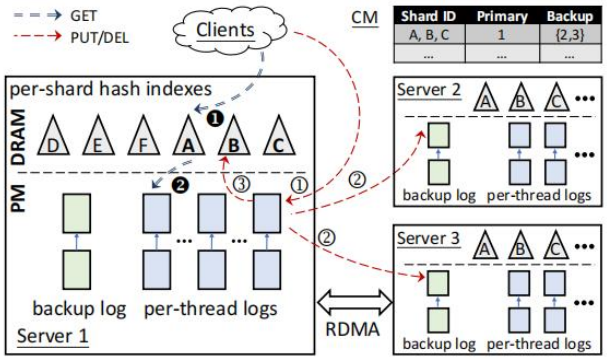

图 3.3 Rowan-KV 的结构

图 3.3 显示了 Rowan-KV 的架构。服务器在 PM 中持久存储对象（即键值对）并使用 RDMA 进行网络通信。 Rowan-KV 将整个数据集划分为许多分片并将它们分布在服务器上。 每个分片都经过复制以实现高可用性：复制因子为 k，它有一台服务器作为主服务器，k-1 个服务器作为备份服务器。 客户端通过 RPC 发出 KV 请求。

Rowan-KV 将每个对象的密钥哈希为 64 位数字，并让分片管理哈希密钥空间中的连续范围。分片分布由配置管理器 (CM) 维护，并缓存在服务器和客户端中。Rowan-KV 使用动态重新分片机制来缓解服务器过载造成的负载不平衡。

Rowan-KV 采用日志结构化方法，每个服务器具有三个组件：

• 线程日志。每个服务器启动许多工作线程来处理来自客户端的请求。 每个工作线程在 PM 中维护一个每线程日志（t-log），其中存储 PUT/DEL 请求的对象。 不为每个分片分配独立的日志，以减少随机 PM 写入。

• 备份日志。 每个服务器在 PM 中都有一个备份日志 (b-log)，它使用 Rowan 实例接收来自主服务器的复制写入。 通过这样做，Rowan-KV 可以在很大程度上消除来自高扇入小写入的 DLWA。

• 分片的哈希索引。每个服务器为其管理的每个分片构建一个 DRAM 驻留哈希表，以索引 t-log 或 b-log 中的对象。 将索引放入 DRAM 中可以避免随机 PM 写入和昂贵的 PM 读取。

当发出对象的 KV 请求时，客户端会向驻留在作为目标分片的主分片的服务器中的工作线程发送 RPC。

3.3 Rowan-kv表现

将 Rowan-KV 与专为 RDMA 网络设计的两种最先进的复制 KVS 进行比较：

• Clover[9]. Clover 运行在分散的 PM 上，其中 PM 服务器没有计算资源。客户端通过 RDMA READ 动词执行 GET 操作，并使用 RDMA WRITE 和 ATOMIC 的组合执行 PUT 操作（包括复制）。

• HermesKV [10]. 它是基于 Hermes [10] 的 DRAM 驻留 KVS，这是一种基于广播的复制协议。HermesKV 使用 RPC 进行所有服务器间通信（包括复制）。

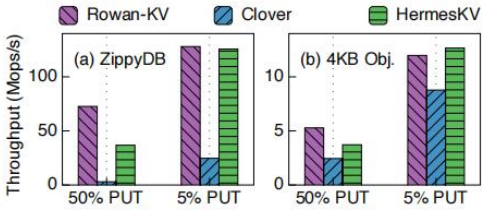

图 3.4 Rowan-KV 的结构

图 3.4 左显示了小型写入（ZippyDB 对象）的结果。 在写入密集型工作负载（即 50% PUT）下，Rowan-KV 的性能分别优于 Clover 和 HermesKV 24.5 倍和 1.98 倍。造成 Clover 吞吐量低的原因有两个。 首先，由于分散的架构，Clover 中的每个操作都需要多个网络通信。 其次，Clover 使用 RDMA ATOMIC 来解决客户端线程之间的冲突，这会在出现争用时导致性能显着下降。 由于其读取-修改-写入行为，在 PM 上使用 RDMA ATOMIC 也被认为很慢 [11]。 HermesKV 使用 RPC 进行复制，这会在备份时消耗 CPU 周期，因此它的性能优于使用单侧 Rowan 进行复制的 Rowan-KV。 测量这些 KVS 的 DLWA。Clover 有 1.86× DLWA，HermesKV 有 2.95× DLWA，因为它们都在 PM 上生成大量随机小写入：对于 PUT 操作，Clover 使用 WRITE 执行写时复制，HermesKV 执行就地更新。 相比之下，Rowan-KV 采用日志结构的方法来管理对象，并利用 Rowan 抽象来最小化写入流的数量； 因此，Rowan-KV 的 DLWA 小于 1.032×。在读取密集型工作负载下（即 5%PUT），Rowan-KV 和 HermesKV 的吞吐量相似，远远超过 Clover（约 5 倍）。

图 3.4 右报告了大型写入（4KB 对象）的结果。在写入密集型工作负载下，Rowan-KV 的性能优于 HermesKV 0.42 倍，并且受到 PM 写入带宽的瓶颈。 HermesKV 无法接近 PM 写入带宽的限制，因为它的备份浪费大量 CPU 周期来将大型对象从 RPC 缓冲区复制/持久化到 PM。在读取密集型工作负载下，Rowan-KV 和 HermesKV 受到网络带宽（每台服务器 11GB/s）的瓶颈，远低于 PM 读取带宽（18GB/s）。

4 FlatStore：日志结构键值存储引擎

键值存储系统具有简单的接口抽象（例如 put、get 等），并被广泛用于存储半结构化数据，其高可扩展、高性能等特性已使其成为数据中心架构的最基本组成部分。随着数据存储和处理的需求不断增长，一部分业务逐渐从读密集型变化为写密集，且写入数据尺寸普遍较小，而此类业务对存储系统的设计带来了极大的挑战。持久性内存、RDMA 等新型硬件的出现为解决上述难题带来了新的机遇。

然而，现有的键值存储系统均存在写入粒度与持久性内存访问粒度不匹配的问题。例如，键值存储系统的工作负载中大部分数据项仅包含几个或几十个字节，一种解决粒度不匹配问题的经典方法是使用日志结构，该方法将存储设备组织为一个日志，并将所有用户写入的数据追加到日志尾部。更为重要的是，存储系统可借助批处理技术将来自客户端的多个请求合并在一起，然后统一执行日志追加，从而降低对存储设备的访问次数。

基于上述内容，[8]作者提出了面向键值存储系统的持久性内存存储引擎 Flat-Store，进而充分发挥日志结构在持久性内存中的潜在性能优势。FlatStore 的核心思想是将键值存储系统划分为用于快速索引的易失性索引结构和用于高效存储的持久性日志结构，并引入压缩日志格式及流水线式水平批量持久化技术来分别解决上述挑战，从而实现高吞吐率、低延迟和多核扩展性等性能目标。

4.1 压缩日志格式

持久性内存键值存储系统生成的大量小写通常会造成严重的写放大问题。为了解决此问题，FlatStore 将键值存储系统拆分三部分，分别为 1）易失性索引结，用于快速数据查找，2）压缩日志

结构，用于批量处理小写，以及 3）持久性内存分配器，用于存储大 KV。

操作日志的设计关键在于如何构造日志项的数据布局。日志项的尺寸应足够小，从而可以更好地支持批量处理，使得 FlatStore 能够在一次持久化操作中处理更多的日志项。为了实现这一目的，FlatStore 仅将索引元数据和小 KV 存放在操作日志中，而其它大 KV 项则通过内存分配器进行单独存储。具体实现中，小于 256 B 的键值对被定义为小 KV，该 I/O 尺寸足以让持久性内存的带宽饱和。同时，每个日志项还需要包含足够多的元数据信息，从而能够在运行过程中进行正常的数据查询，以及在系统崩溃后安全地恢复丢失的易失性索引结构。

综合考虑上述因素，FlatStore 通过操作日志技术对日志项进行数据布局，具体地，每个日志项仅包含用于描述更新操作的必要信息，而不直接记录每次内存更新的内容。如图 4.1 所示

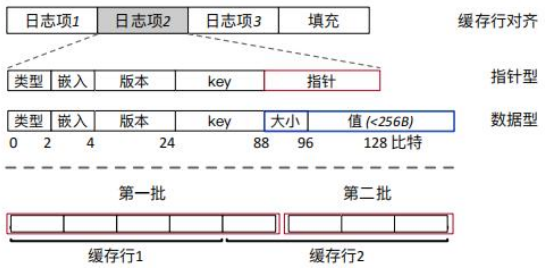

图 4.1 操作日志的两种布局格式

4.2 惰性持久性内存分配器

FlatStore 引入一种惰性持久性内存分配器。它首先将持久性内存空间切分成 4 MB 的连续内存区，然后这些 4 MB 的内存区被进一步切分为不同尺寸的内存块，且同一内存区中的数据块具有相同的尺寸。当前内存区的内存块大小会记录在内存区的头部；同时，用于记录分配信息的位图也放置在开头位置，用以跟踪未被分配的数据块。从上述设计中可以发现，每个内存区的起始地址都是 4 MB 对齐的，并且内存区的分配粒度都可以在其头部找到。

根据这些信息，可以使用日志项中的指针直接计算分配的内存块在内存区中所在位置（即偏移），从而在发生系统崩溃或断电故障后能够正确恢复位图元数据。为提升分配器的扩展性，这些 4 MB 内存区被划分到不同的工作线程。在收到分配请求时，分配器首先从其本地管理的持久性内存空间中选择适当尺寸的内存区，然后从该内存区中分配空闲的数据块，并修改其位图，该过程中修改的位图无需立即持久化。对于大于 4 MB 的空间分配操作，分配器直接为其分配连续的多个内存区。由于键值存储系统中数据普遍较小，因而此类情况发生的概率极低。

将以上所有设计合并到一起，FlatStore 将通过以下步骤在服务端处理 put 请求：

1. 从持久性内存分配器中分配一个大小合适的数据块，将键值对的数据部分以(V_{len} , V)的格式复制到分配的内存空间中，并立即持久化，其中 V_{len} 表示数据大小（若插入操作的数据为小 KV 则跳过此步骤）。
2. 初始化一个日志项并填充各字段。如果将键值对放置在操作日志之外（即大 KV），则将日志项格式化为指针型，且内部指针指向步骤 1 中分配的数据块。如果该键值对已经存在（即重复插入），则将日志项中的版本字段加 1。日志项初始化完毕后将其追加到工作线程本地操作日志的尾部，并持久化。最后更新操作日志的尾指针以重新指向日志的尾部，并持久化尾指针。
3. 更新易失性索引中对应的索引项以指向新追加的日志项。

4.3 水平批量持久化技术

通过引入操作日志，FlatStore 的服务端工作线程可以从网络中同时接收多个客户端请求，并将它们进行合并处理。例如，假设有 N 个 put 请求到达服务端，FlatStore 首先为每个请求分配适合大小的数据块，并存储对应的键值对；然后为每个请求分别生成日志项，并将其合并持久化到操作日志中；最后，更新各个请求对应的易失性索引使更新内容生效。

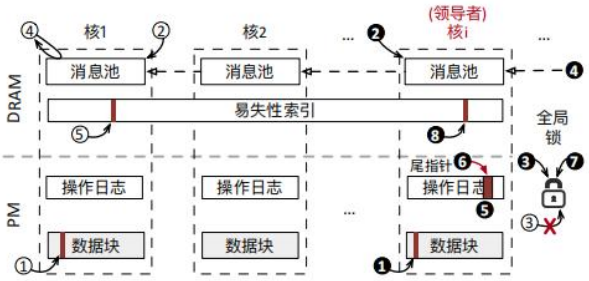

图 4.2 流水线式水平批量持久化技术工作流程

图 4.2 首先描述了一种简单版的水平批量持久化工作流程，在完成键值对持久化阶段（❶/①）之后，各工作线程都将要持久化的日志项对应地址存放在消息池中（❷/②），并尝试获取全局锁（❸/③）。成功获取全局锁的工作线程将成为领导者（即图中的第 i 个核心），而其它线程则成为跟随者，并同步等待领导者帮助其完成日志项的持久化工作（④）。领导者在抢到全局锁之后便开始从其它工作线程中窃取日志项（❹），然后将收集的日志项合并到一起并以批量方式追加到操作日志中（❺和❻）。此后，领导者释放全局锁（❼）并通知其它核心持久化操作已经完成。最后，所有工作线程更新相应的 DRAM 索引，并将响应信息发送给客户端（❽/⑤）。

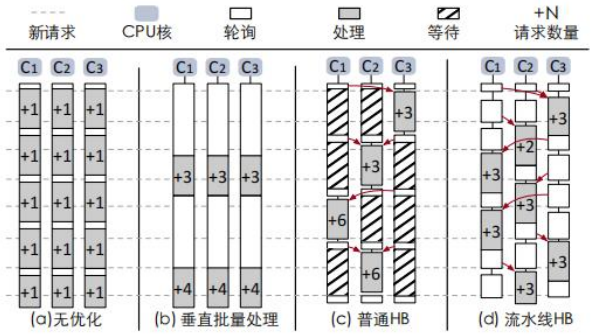

图 4.3 不同请求处理模式的时间线对比

可以观察到，简单版水平批量持久化流程通过从其它核心窃取请求来实现批量处理。然而，上述流程将 put 操作的三个阶段严格分割开，这会导致跟随者将大多数 CPU 资源都花费在等待领导者完成日志项持久化上，从而其性能并不能达到最优状态（图 4.3c 从微观角度解释了这一点）。因此，FlatStore 进一步引入了流水线式水平批量持久化技术让多个 put 操作的各阶段可以交叠执行。具体方法为：一旦某工作线程没有成功获取全局锁，那么它将从网络连接中查询下一个请求，并开始执行下一轮水平批量持久化逻辑，同时异步等待前一轮批量处理的领导者完成日志项持久化工作。此外，领导者从其它工作线程收集完日志项后，便立即释放全局锁，从而将日志持久性过程从全局锁中移出，使得前后两轮批量处理可以并行执行（如图 4.3d）。

4.4 日志清理

为了避免操作日志的长度无限制增长，FlatStore 需要及时清理失效的日志项，从而提升持久性内存的空间利用率。在 FlatStore 中，操作日志由一系列 4 MB 的内存区串接构成。服务端为每个内存区维护了一个内存记录表，并在处理 put 和 delete 请求时实时更新操作日志中每个 4 MB 内存区的空间使用情况。如果一个内存区的空间利用率低于某阈值，或总可用空间非常受限，则将触发日志清理流程，并将空间利用率低的内存区放入到回收列表中。每个批量处理组都会启动一个后台线程来清理日志，该线程被称作清理程序。通过这种设计，日志回收程序可以在多个组之间并行执行，从而提升回收效率。清理程序会定期检查回收列表，然后扫描对应内存区的各日志项是否有效，其判断依据是通过将日志项内部的版本与内存索引中的最新版本进行比较得出。扫描完成之后，清理程序新分配一个内存区，然后将回收内存区中所有的有效日志项复制到新分配的内存区中。墓碑日志的有效性判断相对更为复杂[12]，这是因为只有在与该键值对相关的所有日志条目均被回收之后才能安全地回收该墓碑日志。紧接着，清理程序使用 CAS 原子指令将内存索引结构中的相应索引项进行更新，以指向其最新位置。最后，此内存区便可安全地被持久性内存分配器回收。当清理过程中新分配的内存区装满后，清理程序将其链接到操作日志尾部。为防止新分配的内存区在系统故障之后丢失，清理程序还会将其起始地址记录在预先约定的位置。系统重启之后，相应的地址信息会被重新读取，用以恢复相应的数据。

5 总结

本文首先介绍持节内存和 RDMA 的特点介绍了持久内存和 RDMA 相结合的优势，为构建大容量、高性能和低延迟的内存存储系统提供了希望。同时这种结合又面临着存储软件开销高，硬件特性难感知，吞吐延迟难兼顾的问题。

本文着重介绍了两个模型 Rowan 和 FlatStore, 前者将并发远程小写入转换为单个写入流，并基于 Rowan 的 KVS 实现了高性能，同时很大程度上消除了 DLWA，后者将键值存储系统划分为用于快速

索引的易失性索引结构和用于高效存储的持久性日志结构，并引入压缩日志格式及流水线式水平批量持久化技术来分别解决上述挑战，从而实现高吞吐率、低延迟和多核扩展性等性能目标。目前的这种结合没有一个统一的抽象层次，存在功能冗余、兼容性差、编程困难、架构复杂等缺陷。

近年来，研究人员提出了一种解聚合的数据中心架构并受到广泛关注。该架构下，处理器、内存、存储等硬件资源将以硬件资源池的方式独立维护，而资源池之间通过高速网络进行互连。因此，不同数据中心可以根据需求独立的扩展各类硬件资源，从而有效提升硬件资源利用率。然而，处理器访问内存的方式将从传统的内存总线访问变为跨网访问，这对操作系统的设计、存储资源的管理、以及性能调优等方面将带来巨大变化。虽然有很多需要调整的地方，但这或许是未来数据中心新的架构的方向。

致 谢 *感谢施展老师、胡燏翀老师、童薇老师给我们讲授这门数据中心技术课程，组织我们完成作业和详细解答我们的问题，从中我学到了严谨治学的态度，以及对自己研究的内容深入的决心。读论文做实验就是要沉下心来急不得也敷衍不得，其中施展老师提出的带着问题和怀疑的心态读论文令我印象深刻。我会将其运用到以后的科研学习之中。

参 考 文 献

[1] Baek I G, Lee M S, Seo S, et al. Highly scalable nonvolatile resistive memory using simple binary oxide driven by asymmetric unipolar voltage pulses[C]//Kim K. Proceedings of the 2014 International Electron Devices Meeting. Piscataway, NJ, USA: IEEE, 2004:587-590.

[2] Xu J, Swanson S. NOVA: A log-structured file system for hybrid volatile/non-volatile main memories[C]//Brown A, Popovici F. FAST '16: Proceedings of the 14th USENIX Conference on File and Storage Technologies. Berkeley, CA, USA: USENIX Association, 2016:323-338.

[3] Hu Q, Ren J, Badam A, et al. Log-structured non-volatile main memory[C]//Silva D, Ford B. USENIX ATC '17: Proceedings of the 23rd Conference on USENIX Annual Technical Conference. Berkeley, CA, USA: USENIX Association, 2017:703–717.

[4] Lu Y, Shu J, Chen Y, et al. Octopus: An RDMA-enabled distributed persistent memory file sys tem[C]//Silva D, Ford B. USENIX ATC '17: Proceedings of the 23rd Conference on USENIX Annual Technical Conference. Berkeley, CA, USA: USENIX Association, 2017:773–785.

[5] Liu Z, Bai Z, Liu Z, et al. Distcache: Provable load balancing for large-scale storage systems with distributed caching[C]//Merchant A, Weatherspoon H. FAST '19: Proceedings of the 17th USENIX Conference on File and Storage Technologies. Berkeley, CA, USA: USENIX Association, 2019:143-157.

[6] Yu Z, Zhang Y, Braverman V, et al. Netlock: Fast, centralized lock management using programmable switches[C]//Misra V, Schulzrinne H. SIGCOMM '20: Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications,Technologies, Architectures, and Protocols for Computer Communication. New York, NY, USA: ACM, 2020:126–138.

[7] Wang Q, Lu Y, Wang J, et al. Replicating Persistent Memory {Key-Value} Stores with Efficient {RDMA} Abstraction[C]//17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23). 2023: 441-459.

[8] Chen Y, Lu Y, Yang F, et al. Flatstore: An efficient log-structured key-value storage engine for persistent memory[C]//Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems. 2020: 1077-1091.

[9] Tsai S Y, Shan Y, Zhang Y. Disaggregating persistent memory and controlling them remotely: An exploration of passive disaggregated {Key-Value} stores[C]//2020 USENIX Annual Technical Conference (USENIX ATC 20). 2020: 33-48.

[10] Katsarakis A, Gavrielatos V, Katebzadeh M R S, et al. Hermes: A fast, fault-tolerant and linearizable replication protocol[C]//Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems. 2020: 201-217.

[11] Wei X, Xie X, Chen R, et al. Characterizing and Optimizing Remote Persistent Memory with {RDMA} and {NVM}[C]//2021 USENIX Annual Technical Conference (USENIX ATC 21). 2021: 523-536.

[12] Rumble S M, Kejriwal A, Ousterhout J. Log-structured Memory for {DRAM-based} Storage[C]//12th USENIX Conference on File and Storage Technologies (FAST 14). 2014: 1-16.

附录

1、RDMA全称是什么，与DMA相比的优势是什么？分别与计算机结构中的哪些硬件交互？

RDMA 全称是 Remote Direct Memory Access，它是一种计算机网络技术，允许一台计算机的内存直接访问另一台计算机的内存，而无需经过中央处理单元（CPU）的参与。这种直接的内存访问方式可以提高数据传输的效率和降低传输延迟。与 DMA（Direct Memory Access，直接内存访问）相比，RDMA 具有以下优势：

无主机参与： RDMA 可以在两台计算机之间直接传输数据，而不需要中央处理单元的介入。相比之下，DMA 通常还需要 CPU 的介入来启动、管理和终止数据传输。

低延迟： 由于 RDMA 允许直接内存访问，避免了 CPU 的参与，因此可以实现更低的数据传输延迟。这对于一些对延迟敏感的应用程序非常重要，如高性能计算和数据中心应用。

高吞吐量： RDMA 可以提供更高的数据传输吞吐量，因为数据传输过程中没有 CPU 的干扰，可以更有效地利用网络带宽。

RDMA 主要与计算机结构中的网络适配器（Network Adapter）和内存交互。RDMA 的实现通常依赖于支持 RDMA 功能的网络适配器，这些适配器通过网络连接两台计算机的内存，使它们能够直接进行数据传输，而无需主机 CPU 的干预

2、这篇文章的贡献是什么？

-将并发远程小写入转换为单个写入流

-基于 Rowan 的 KVS 实现了高性能，同时很大程度上消除了设备级别的写放大问题

3、一些关于旧瓶装新酒和新瓶装旧酒的思考

有时候，对经典理论、方法或技术的重新解释、改进或应用可以带来新的见解和创新。这种情况下，虽然使用了旧的概念或方法，但研究工作仍然具有独特性和价值。将已有的理论或方法应用到新的领域，或者通过新的角度对已有知识进行解释，同样可以带来有价值的成果。这样的研究可能为其他领域提供了新的思路和应用方向。持久性内存和 RDMA 这些新型硬件材料和计算机网络技术可以应用到当前的键值存储方向或许会出现新的收获，我个人认为这就是旧的瓶子（键值存储领域)装了新酒的例子（RDMA 和持久性内存）。