# 利用智能网卡的数据路径加速器加速有序键值存储

Frederic Schimmelpfennig
约翰内斯·古腾堡
美因茨大学

Jan Sass
约翰内斯·古腾堡
美因茨大学

Reza Salkhordeh
约翰内斯·古腾堡
美因茨大学

Martin Kröning亚
琛工业大学

Stefan Lankes
亚琛工业大学
大学

André Brinkmann
约翰内斯·古腾堡
美因茨大学

arXiv:2601.06231v1 [cs.DC] 9 Jan 2026

**摘要**

远程内存键值（KV）存储是各种现代工作负载的基石，而高速范围扫描也常常是必需功能。然而，现有架构很少能同时兼顾极致效率、架构简洁性以及对有序操作的原生支持。传统的以主机为中心的框架受限于内核态网络协议栈和内部总线延迟。采用操作系统旁路或直接运行于智能网卡上的基于哈希的替代方案虽然吞吐量很高，却缺少支持范围查询所需的数据结构。基于 RDMA 的分布式系统兼具性能与范围操作能力，但往往依赖有状态客户端，导致扩展和错误处理更加复杂。另一种方案是在智能网卡上实现遍历位于主机内存中的树，但会受到较高的 DMA 往返延迟影响。

本文介绍了一种键值存储，它利用 BlueField-3 智能网卡上的在途数据路径加速器（DPA），在消除操作系统开销的同时支持无状态客户端和范围操作。这些 DPA 直接从网卡缓冲区接收网络请求，以遍历驻留在加速器本地内存中的无锁学习型索引。设计将从主机侧树副本中获取值的操作延迟到到达叶节点层级时执行，从而尽量减少 PCIe 传输。写操作暂存在 DPA 内存中，并批量迁移至主机；在主机上完成结构维护后，再以事务方式拼接回智能网卡。结合驻留于网卡的读缓存，该系统在点查找中可达到每秒 3300 万次操作（MOPS），在范围查询中可达到 1300 万次操作每秒。分析结果表明，该架构的性能可媲美或超越当今最先进的解决方案；我们还指出了可进一步提升性能的硬件改进方向。

## 1 引言

许多现代工作负载依赖支持查找、插入和范围查询的远程键值存储。如何而，远程性能通常受到传输瓶颈的限制。传统的远程键值存储（例如 Redis [37]和 Memcached [32] 通过以太网访问。它们使用操作系统（OS）内核网络协议栈，以及网卡与主机之间的 PCIe 瓶颈，从而限制吞吐量并增加延迟。

为降低操作系统开销，一些方法采用数据平面开发套件（DPDK）[11]，通过直接访问用户空间绕过网络协议栈。MICA 在单个服务器节点上可实现近 1 亿次操作每秒（MOPS）[27]。KV-DIRECT 等其他工作则通过消除主机参与、直接从 FPGA 智能网卡处理请求，实现了更高的吞吐量 [22]。由于键值存储离网络更近，这降低了延迟。然而，要取得这些成果，KV-DIRECT 在底层数据结构的复杂度和容量方面受到限制。MICA 和 KV-DIRECT 均采用基于哈希的点查找，且不支持范围查询。近期的研究将类似的卸载设计应用于更新一代的智能网卡，但也存在相同的不足 [14, 36]。HONEYCOMB 等方法 [28] 会遍历主机内存中的数据结构，并支持范围查询。然而，由于频繁进行 DMA 往返传输，其性能较低。

使用远程直接内存访问（RDMA）的键值存储客户端可以直接访问主机内存 [24, 25, 38, 44–46, 54]。这些方法带来了架构上的挑战，因为每个客户端最终都需要知道服务器上数据的远程地址，才能访问这些数据。SHERMAN 可尽量减少写操作的往返次数，但树遍历需要客户端缓存 [24]。类似地，ROLEX [25] 在客户端维护元数据索引，并尝试利用学习型索引优化访问。这些系统能够提供高性能，但依赖客户端逻辑和资源，从而增加了扩展的复杂性、带来一致性方面的顾虑，并需要额外的故障处理。

本文介绍*DPA-Store*，这是一种有序内存键值存储，利用 BlueField-3 智能网卡 [35] 及其数据路径加速器（DPA）[34] 绕过操作系统开销，并支持无状态客户端。DPA 是高度并行、可编程的计算单元，

嵌入在网络数据路径中。DPA 可直接访问网卡缓冲区和网卡侧动态随机存取存储器，并可执行直接内存访问操作来访问主机内存，从而实现低延迟、灵活的请求处理 [6]。

DPA-Store 围绕 DPA 子系统的高并发能力构建，该子系统包含 16 个核心，每个核心配备 16 个线程，共计 256 个线程。在接收请求时，这些线程无需主机参与，即可遍历存储在网卡内存中的学习型索引树 [19] 。为规避网卡的内存容量限制，值存储在主机上的树副本中。点查询和范围查询在到达叶节点并访问主机内存中的值时，都只需由 DPA 执行极少量的 DMA 操作。这样可以最大限度地减少网卡与主机之间跨越 PCIe 边界的次数。

收到插入请求后，负责处理的 DPA 线程会在网卡侧的叶节点层级将该条目追加到插入缓冲区中。这些条目会立即对后续查询可见。插入缓冲区填满后，会批量传输到主机。主机凭借更强的计算能力，使用执行代价高昂的结构更新，通过并发补丁 线程更新主机侧树：将插入内容应用到主机侧树、重新训练受影响的子树，并准备新节点。随后，一组拼接 线程将新节点复制到网卡侧树中，并通过指针交换使其可用。这在保证一致性的同时，使网卡侧遍历路径保持无锁。

我们为热点条目采用网卡侧读缓存，以减少树遍历和 DMA 操作的次数。请求根据其哈希值路由到 DPA 线程，使得用于保护缓存访问的小型布隆过滤器可以放入每个 DPA 线程剩余的缓存行空间中。这些缓存行几乎不会从 DPA 线程的 L1 缓存中被逐出，因此能够高效缓存热点键值对。

我们的学习型索引能够针对 DPA 子系统的内存特性、调度和并发进行细粒度优化。我们围绕最优缓存行访问设计遍历逻辑，并采用定点计算来弥补 DPA 缺少浮点运算单元的问题。我们的贡献如下：

- 我们证明，可以在 BlueField-3 Smart-NIC 上构建一个支持查找、插入和范围查询的键值存储；对于大多数操作，其性能已可与最先进的基于 RDMA 的键值存储相媲美，且无需依赖有状态客户端。DPA-Store 的 GET 操作可达 33 百万次操作每秒，范围操作可达 13 百万次操作每秒。
- 我们进行了大量实验来评估 DPA-Store。通过这些实验，我们揭示了 BlueField-3 的深层性能特征，超越了先前研究。我们还借助这些洞见解释了 DPA-Store 的插入性能为何仅为 1.7 百万次操作每秒。
- 我们展示了如何在不改变整体编程模型的情况下，对 BlueField-3 硬件进行少量修改，从而释放出显著的性能提升，轻松超越当今最快的基于远程直接内存访问的键值 Store。

本文结构如下。第 2 节介绍背景并综述相关研究。第 3 节详细介绍 DPA-Store 的架构与实现。第 4 节通过敏感性分析和对比基准测试评估 DPA-Store。最后，第 5 节总结全文。

## 2 背景与相关研究

本节概述学习型索引和 BlueField-3 DPU 的背景，并介绍 DPA-Store 与基于内核旁路、网卡辅助和远程直接内存访问的远程键值系统之间的关系。

### 2.1 经典索引与学习型索引结构

键值存储（KV）提供简洁的接口，支持 GET、插入、更新、删除，以及可选的范围操作。通常，键值存储依赖哈希表；若需要范围操作，则依赖有序树结构。现代非远程内存键值存储采用传统数据结构的优化变体，例如 $B^+$-树、字典树或哈希映射，为节点本地进程提供高度优化的遍历和并发控制 [20, 21, 29, 49]。然而，传统数据结构的内存访问模式往往并不理想 [19]。

为了解决这一问题，学习型索引 [8, 10, 12, 18, 19, 23, 26, 31, 48, 51] 使用轻量级模型捕获键分布，使查找操作能够在进行短暂扫描之前，直接跳转到节点内的相关区域，从而在树的每一层只需访问极少的缓存行（参见图 1）。近似键分布的一种常用方法是分段线性近似（PLA）。通过对累积分布的键拟合短线性段，可以利用参数$a$＝和$\cdot b$＋对键$k$ 的目标位置进行线性预测$p$，误差不超过 ε。

因此，与每个节点需要进行$O(\log n)$次内存访问的可能需要进行许多彼此独立的缓存行访问（例如，在 $B^+$-树中），而学习型索引中的查找操作则被聚合

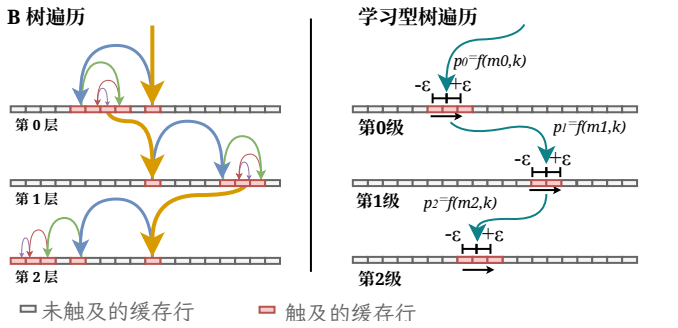



图 1：B树与学习型树中的访问模式。

在一个小型连续窗口中 $[p-\varepsilon, p+\varepsilon]$，以快速的中央处理器周期换取相对缓慢的内存访问。

误差界限 $\varepsilon$ 在容量效率与扫描开销之间进行权衡。界限越小，扫描范围越小，但模型数量越多；界限越大，保留在连续段中的数据越多，因此需要扫描更大的区域。因此，在将学习型索引映射到智能网卡等内存访问性能有限的加速器时，选择合适的参数 $\varepsilon$ 至关重要。

插入和模型重建策略会进一步影响运行时行为。ALEX [8], 例如，将键存储在带间隙数组中，并使用较小的缓冲区；只有在间隙填满后才需要重新训练。HYPER [51] 结合自底向上和自顶向下策略，并利用溢出缓冲区来限制内存开销，而 XINDEX [51] 则面向高并发工作负载。在所有情况下，缓冲区布局、大小和更新策略都会影响重新训练成本和并发特性。

## 2.2 远程键值存储

远程键值存储允许通过网络协议进行访问，因此带来了额外的复杂性。广泛使用的基于以太网的键值存储 [9, 16, 32, 37] 适用于通用场景。然而，其请求吞吐量受主机网络协议栈及相关操作系统（OS）组件的限制，例如网络套接字、上下文切换以及网卡与主机之间的 PCIe 边界 [3, 33, 39]。

为解决这些问题，基于 RDMA（远程直接内存访问）的键值存储绕过操作系统和网络协议栈 [5, 13, 54]。它们采用传统的树架构 [24, 44, 45] 或学习型索引 [25, 46]，并提供完整的键值操作集，包括范围查询。

SHERMAN [24] 是一种构建于解聚内存之上的写入优化型分布式 B$^+$树，采用分层网卡片上锁、客户端索引缓存和无锁读取来提高写入吞吐量。SMART [38]通过解决 RDMA 网卡扩展瓶颈来优化这一方法。XSTORE [46] 将服务器端 B$^+$树与客户端上的学习型缓存相结合，客户端利用该缓存预测键值对在服务器上的位置，从而减少 RDMA 往返通信。这使范围查询最多只需两次 RDMA 操作即可完成。服务器在后台重新训练模型。ROLEX [25] 通过严格控制学习型树中的数据移动来应对动态工作负载，从而实现模型的异步重新训练。叶节点原子移位方案保持叶节点有序并将干扰降至最低。

这些系统表明，有序范围查询的低延迟服务能够实现高吞吐量。然而，它们要求客户端执行有状态的索引逻辑，或者每个请求依赖多次远程直接内存访问往返。

还可以通过尽量降低操作系统的影响或使用智能网卡来克服严重的瓶颈。MICA 将 DPDK 用户空间网络与并行数据分区相结合，以绕过内核开销，使得

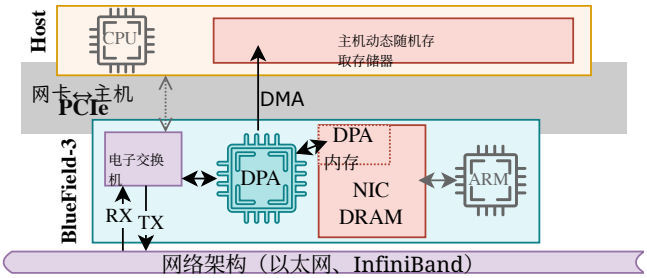



图 2：配置为网卡模式的 NVIDIA BlueField-3 架构，可通过直接内存访问访问主机内存。

单台服务器能够持续实现近 100 百万次操作每秒 [11, 27]。KV-DIRECT 通过扩展 RDMA 原语，将键值存储卸载到可由 FPGA编程的网卡上，提供驻留于网卡的点查找；随着每台服务器配备的网卡数量增加，其性能也能扩展，同时即使吞吐量很高，尾延迟仍可保持在微秒级，不过容量有限 [22]。MICA 和 KV-DIRECT 均采用基于哈希的数据结构来实现不受限制的吞吐量，且不支持范围查询。

一些近期研究 [14, 36, 47, 52] 也不支持范围操作，但深入揭示了当前智能网卡的能力，智能网卡能够卸载日益复杂的键值存储机制 [15, 17, 40–42]。

HONEYCOMB [28] 在智能网卡上的现场可编程门阵列中实现了 B 树遍历，并通过直接内存访问访问主机内存中的树。它提供键值存储接口，包括范围查询，但对于未缓存的键值对，需要执行多次开销高昂的直接内存访问。HIDPU [53]为解耦存储执行地址转换，并使用华为 Hi1823 智能网卡的 DPA 来映射部分连续的地址区域。HIDPU 将这些映射存储在专用分段中，并通过网卡侧内存中的学习模型进行访问。就功能和需求而言，这项工作与通用键值存储正交；它依赖哈希分段，且不支持跨多个映射的范围查询。尽管如此，其性能优势仍促使人们将键值存储卸载到智能网卡上。另一方面，DALDEX 使用智能网卡卸载持久内存学习索引的训练，以减少主机开销，而非用于网络服务 [43]。

## 2.3 BlueField-3 和 DPA 子系统

BlueField-3 智能网卡是一款基于 ConnectX-5 的网络适配器。除了高速网络功能外，它还配备了一个带外 ARM CPU、一个带内 DPA 集群以及 16 GIB DDR5 内存（见图2）。ARM CPU 可让 BlueField-3 运行专用操作系统（例如用于控制平面管理），并访问智能网卡上的加速器引擎（例如加密/解密引擎）。

DPA 子系统由 16 个物理 RISC-V 核心组成，每个核心的运行频率为 1.8GHz，并配备 16 个线程。目前，256 个线程中只有 189 个并发线程

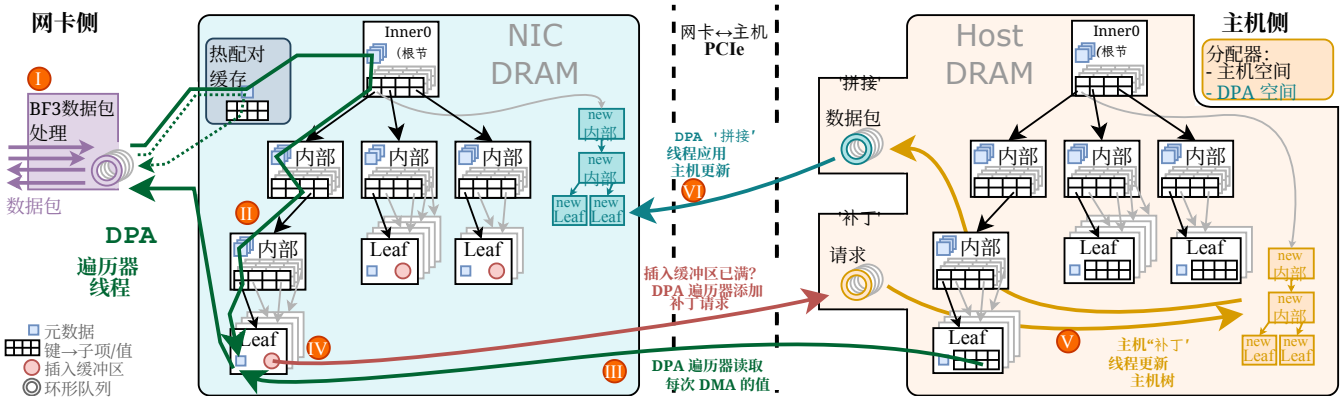



图 3：DPA-Store 架构：(I) 请求数据包在路径上被分配给一个 DPA 遍历器线程，该线程 (II) 遍历网卡侧学习型索引。(III) 使用叶级模型通过 DMA 获取值，(IV) 将插入操作放入叶级插入缓冲区。(V) 主机执行结构更新，并在不打断遍历器的情况下，(VI) 以事务方式将更新拼接 回网卡。

可供应用程序代码使用 [6]。DPA 线程由硬件以细粒度进行调度，从而实现高并发和数据包处理流水线化。

每个 DPA 线程都有独立的 L1 缓存，以及共享的 1.5 MiB L2 缓存和 3 MiB 三级缓存。缓存层次由 BlueField-3 DDR5 内存中一个专用的 1 GiB 区域提供支持，该区域称为*DPA* 内存。此外，根据配置，DPA 可以访问 BlueField-3 内存的其余部分或主机内存。如果将 BlueField-3 配置为允许 DPA DMA 访问主机，则会禁用 ARM CPU [6]。

DPA 线程的调用可以由主机或 ARM 上的应用程序控制，也可以通过为传入网络数据包定义传输接口接收（TIR）目标来控制。对于后一种方式，可以使用基于硬件定义字段（例如接收方或发送方地址、端口或 VLAN 标签）的数据包匹配规则，以线速筛选相关数据包。匹配的数据包会直接放入 DPA 的 L2 缓存，从而能够低延迟访问数据包数据。未匹配的数据包则会透明地进入默认网络路径。

先前的研究评估了 BlueField-3 的架构和 DPA 子系统 [6]。BlueField-3 通过支持路径内与路径外计算、专用加速器引擎和专用内存，旨在适用于广泛的应用。然而，开发高性能应用需要深入了解该架构的局限性。例如，对 DPA 内存的访问会产生近 500ns 的延迟，明显慢于标准中央处理器对动态随机存取存储器的访问。因此，在本研究的背景下，尽可能减少对树形数据结构的内存访问次数至关重要，这也促使我们采用学习型索引，因为它们在每个树层级的内存访问次数有限。

## 3 DPA-Store：架构与实现

本节将介绍 DPA-Store，这是一种支持范围查询并允许无状态客户端使用的远程键值存储。DPA-Store 利用 BlueField-3 DPU 的 DPA 子系统处理传入请求，避免所有操作系统开销，并充分发挥 DPA 的高度并行性。

我们选择了学习型索引树，以减少对延迟受限的 DPA 内存的访问次数。由于 DPA 内存容量仅为 1 GiB，受到限制，DPA-Store 使用大容量主机内存，在树副本中存储值，并通过尽可能少的网卡与主机之间的 DMA 操作来获取值。此外，主机还用于执行计算密集型或会造成阻塞的树操作，例如模型重新训练和节点分裂。主机通过读-复制-更新（RCU）语义将树更新传播到网卡侧，使 DPA 能够无锁遍历索引树，并始终保持一致性。

DPA-Store 为热点键值对采用线程本地缓存。插入和更新操作会批量缓存在叶级插入缓冲区中，从而减少传播到主机的树更新次数。图 3 展示了整体架构，本节将对此进行详细讨论。我们先介绍网卡侧请求路径（第 3.1 节），包括学习型索引和缓存的细节；随后介绍涉及主机的更新周期（第 3.2 节）。

### 3.1 DPA 子系统中的请求处理

DPA-Store 使用以太网之上的用户数据报协议（UDP）来传输请求，每个请求由单个 UDP 数据包组成。所有请求都终止于 BlueField-3 的 DPA 线程。我们定义了数据包匹配规则，将每个监听的 UDP端口映射到指定的 DPA 线程，从而有效利用硬件引导规则，将请求分发到可用的 DPA。虽然

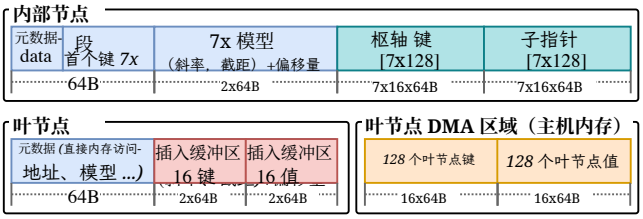



图 4：NIC 侧节点和叶节点 DMA 区域的布局。

每个请求都可以由任意 DPA 线程处理，但端口选择可供客户端用于实现负载均衡或提高缓存利用率。默认情况下，客户端使用相同的键哈希，将负载均匀分布到所有 DPA 线程上。

传入请求由负责的 遍历器 线程（总计 176 个）处理。这些线程沿学习索引树向下遍历，直到到达相应的叶节点。内部节点包含七个学习模型，每个模型由其区间上的分段线性近似（PLA）、枢轴键以及指向相应子节点的指针组成（见图 4）。我们将每个分段的枢轴键和子节点指针数量限制为 128，以匹配所选的 $\varepsilon$值。

将内部节点划分为七个分段后，每个分段的首个键和节点元数据都能容纳在单个缓存行中。遍历器线程执行二分查找，将操作键与各分段的键进行比较，同时预取分段模型，从而使计算与内存访问重叠。找到某个键对应的目标分段后，便对该分段的模型求值，以获得预测位置。同样，在计算过程中，我们会预取包含当前分段枢轴键的缓存行。随后利用预测结果扫描枢轴数组，找到索引及指向指定子节点的指针，之后遍历器线程再进入下一层。我们分别存储枢轴和子节点指针，以便在进行一次子节点访问之前，能够在每个缓存行中比较更多枢轴。

遍历器线程到达叶节点后，会执行请求的操作。插入、更新和删除请求仅操作插入缓冲区。键值对（对于删除操作，则为键删除标记）通过两个原子计数器追加：一次原子递增发生在写入之前，另一次发生在数据写入之后，从而允许并发写入者并行操作。我们确保并发查找先读取值，再读取相应的键，以保证映射正确。

当遍历器向插入缓冲区的最后一个槽位写入数据时，会通过直接内存访问向主机发送补丁请求。随后，主机可以执行所需的树操作，并通过拼接器线程将树更新传播回网卡侧树（参见第3.2节）。在此期间，网卡侧仍只能读取相应的插入缓冲区。如果遍历器因插入缓冲区已满而无法向其中追加内容，就会重新入队，以便稍后处理。

GET 请求会扫描插入缓冲区，查找最近更新或插入的目标键对应的值。每个叶节点都有一个独立的插入缓冲区，而 DPA 线程不太可能缓存该缓冲区。因此，在默认情况下计算模型预测时，我们会预取插入缓冲区。如果插入缓冲区中包含所请求的键值对，就会提前结束处理。否则，遍历器线程必须扫描预测位置附近的叶节点键数组。叶节点键数组位于主机内存中。为降低 DMA 延迟，我们还会在计算模型时预取键数组。找到数组中的键位置后，再通过直接内存访问从主机内存中读取对应的值。

范围请求会像 GET 请求一样，携带目标键 $k_{\min}$到达叶节点。随后扫描插入缓冲区，并将请求范围内的所有值添加到临时位置，该位置之后会成为响应。接着扫描 DMA 数组。扫描到叶节点末尾时，遍历器会通过重新向下遍历来继续处理下一个叶节点，所用的键是严格大于上一个返回键的最小键。

DPA 线程完成请求后，会向客户端发送响应数据包。响应数据包的布局与请求相同，但会在类型字段中加入状态码，并填充值字段或范围字段。对于范围响应，每个数据包最多携带 64 个键值对，以满足 1500 字节的最大传输单元（MTU）限制。

DPA 执行树遍历始终是无锁的，因为树结构只会在主机端发生变化，且这些更改会拼接回网卡，并遵循 RCU 语义（参见第 3.2 节）。因此，DPA 线程在树遍历期间不会停顿，树始终保持一致。

### 3.1.1 学习型索引参数

我们采用分段线性近似（PLA），其斜率为 $a$，截距为 $b$，预测值为 $p(k) = a \cdot k + b$。对于内部节点，每个段都包含一个独立的模型，用于预测下一层节点的枢轴。对于叶节点，模型预测值的位置。预测完成后，线性搜索会在 $[p-\varepsilon, p+\varepsilon]$ 范围内最多访问 $2\varepsilon$ 个键。我们为内部节点设置 $\varepsilon_{\text{inner}} = 4$ 的误差界（因此最多需要两次缓存行访问），为叶节点设置 $\varepsilon_{\text{leaf}} = 8$ 的误差界（因此最多需要访问三个由直接内存访问获取的缓存行）。

我们使用 PGM 提供的贪心算法 [10] 根据已排序键计算模型参数。选定的$\varepsilon$ 值在训练期间保持固定并强制执行。为防止内存碎片化并减少更新期间的分配开销，我们限制内部节点分段和叶节点的大小，使其最多容纳 128 个枢轴键和子节点指针。

BlueField-3 DPA 不支持浮点运算 [34]。因此，所有算术运算都使用定点数表示斜率和截距。为确保在完整的 64 位键空间中进行精确计算，我们在计算过程中暂时将其扩展到 128 位（另请参见 [53]）。

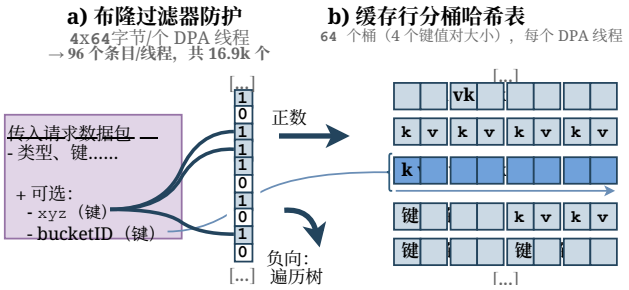



图5：DPA端热点条目缓存。

### 3.1.2 网卡上的热点条目缓存

对键值存储的访问通常呈偏斜分布 [1,4,7,50],，因此缓存热点条目可以减少树遍历和直接内存访问读取。因此，每个 DPA 遍历线程都会维护一个缓存（见图 5），其中包含：(i) 一个三路布隆过滤器；(ii) 一个哈希表，由一个包含多个桶的数组构成，每个桶包含四个键值对。每个桶的大小与缓存行相同，以提高内存利用率。

为确保相同的键始终由同一个 DPA 线程处理，客户端通过共享的键哈希选择该线程的 UDP 端口，并向指定的 DPA 线程发送数据包。客户端还会添加缓存查找所需的数据，例如布隆哈希索引和哈希桶索引，以减少 DPA 计算所需的工作量。每个布隆过滤器包含 256 位，这是能够放入遍历线程上下文剩余缓存行的最大尺寸。同一缓存行还包含用于控制接收/发送队列的重要元数据，因此不太可能从线程的 L1 缓存中被逐出。因此，布隆过滤器不会产生额外的内存访问开销。

我们将每个遍历线程的缓存容量设为 96 个条目，平均假阳性率为 31%。176 个遍历线程总共可缓存 16,896 个条目。假设数据集包含 2 亿个条目，且访问模式遵循参数为 $\alpha = 1$ 的齐夫分布，那么这些缓存条目涵盖了超过 50 % 的请求。随机选择要缓存的值，整体命中率为 25%。我们没有主动跟踪访问模式，因为这需要额外的计数器和内存访问。为保证一致性，缓存同时包含键和值，以检测键哈希冲突；而更新和删除请求则会使相应的缓存条目失效。

### 3.1.3 最大化传输速率

DPA-Store 使用 UDP 传输请求与响应。我们选择不使用更可靠的协议（例如传输控制协议（TCP）），因为DPA调用是基于事件的，因此不适合实现需要会话窗口、传输缓冲区、序列跟踪和基于时间的重传的 TCP 协议栈。相反，DPA-Store 通过监控客户端的响应速率来处理服务器上的缓冲区溢出。如果客户端未能及时收到响应，就会重新发送请求。这是网络拦截式设计中常见的问题，例如 KV-Direct and Mica [22,27]。

每线程 DPA 接收队列（每个遍历器有 256 个数据包）最多可容纳 45,056 个在途数据包，因此 DPA-Store 实例能够处理大量客户端。不过，当将数据包分配到 DPA 核心的键哈希方案产生严重倾斜，导致单个 DPA 线程不堪重负时，可能会发生 DPA 侧溢出。因此，如果某个客户端的响应速率超过阈值，客户端可以逐步采用替代哈希方案，将 GET 请求发送到不同的遍历线程。为防止此时热条目缓存不一致，客户端会从请求中移除哈希元数据，从而绕过缓存。

## 3.2 更新周期

本小节将说明如何在不阻塞遍历线程的情况下，将缓冲写入转化为结构性更新。我们介绍主机上的并发补丁线程，以及 DPA 子系统中的拼接线程。DPA-Store 采用 RCU 语义，确保网卡侧的树始终保持一致，因此节点绝不会就地修改，而是通过原子指针交换进行替换。更复杂的树更新会自底向上执行，直至到达最后一个需要修改的节点。随后通过指针交换更新其在父节点中的地址。此拼接过程对所有遍历线程都是透明的。

DPA 子系统的缓存一致性协议保证节点地址在所有 DPA 线程中始终一致 [34]。不过，遍历器可能会向下遍历过时的子树。为保持整体一致性，只有在所有遍历器都使用纪元处理完当前请求并转向下一个请求后，才会对过时节点和主机侧 DMA 位置进行垃圾回收。

### 3.2.1 主机侧补丁处理

DPA 遍历线程会将插入、删除和更新操作追加到每叶节点插入缓冲区中。当插入缓冲区满时，相应的遍历器会通过直接内存访问将补丁请求放入专用的主机内存队列。每个插入缓冲区只能触发一个补丁请求，因为只有填满该缓冲区的遍历器才允许发出补丁请求。在主机上，多个补丁处理线程（例如四个）负责处理补丁请求并更新主机侧树。如果补丁请求仅包含更新操作，修补器就会相应地修改值。在这种情况下，修补线程会通知 DPA 侧的拼接线程清空插入缓冲区，且不再执行其他操作。

如果补丁包含插入或删除操作，修补器会先将键值对与现有叶节点内容合并到一个临时排序数组中。得到的数组

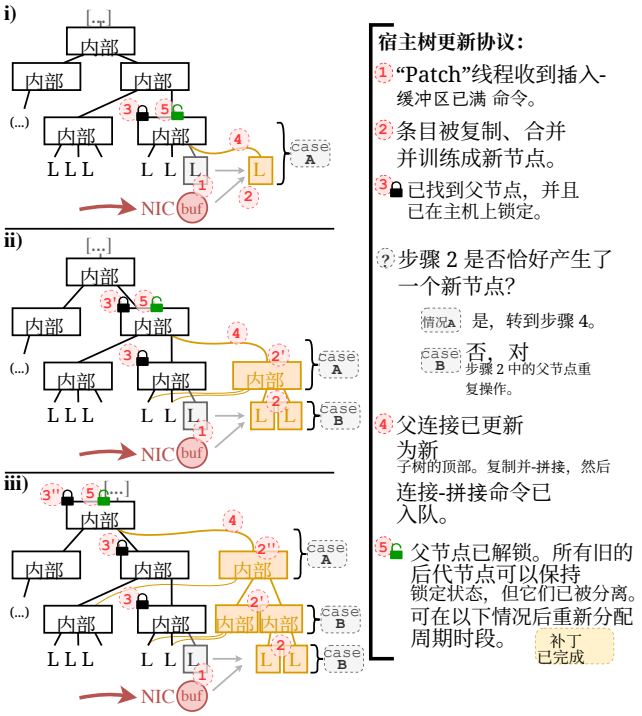



图 6：主机侧补丁协议。

随后使用 PLA 分段进行分区 $\varepsilon_{leaf}$，每个分段都会成为一个新的叶节点。这样得到的结果要么是一个符合叶节点容量的单独分段，要么是多个分段。需要分裂时，我们会使用重训练界限（0.25×容量）限制分段大小。由此产生的叶节点较为稀疏，后续的补丁请求可能无需再次分裂即可吸收，从而减少分裂操作的总次数。

主机上的叶节点更新后，修补线程从根节点向下遍历，以定位原叶节点的父节点。随后，它会锁定该父节点以进行独占访问，使得来自兄弟节点的并发更新必须等待当前更新完成。我们避免使用显式父指针，因为在并发环境下维护双向引用需要复杂的同步方案。

如果重训练恰好生成了一个叶节点，则只需在父节点中执行一次指针交换即可。否则，还必须重建父节点。我们将新的子节点枢轴与父节点条目合并，并使用 $\varepsilon_{inner}$ 应用 PLA 分段。此步骤中的 PLA 范围表示内部节点分段。如果分段产生的范围数超过每个节点允许的最大数量，则会创建多个内部节点，每个节点包含数量均衡的分段。与拆分叶节点类似，我们使用重训练界限限制内部节点中的分段数量。总体而言，对于树的每一层，该过程都会自底向上迭代；当父节点无需拆分或已到达根节点时才会停止。

一旦主机树中包含了新节点，我们就会传播

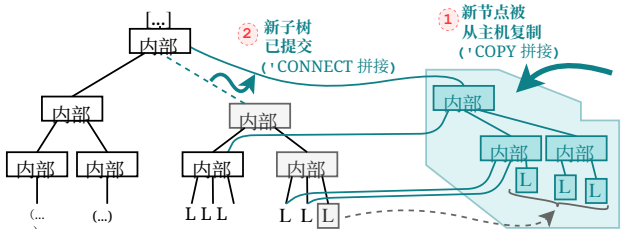



图 7：应用拼接操作时，NIC 树仍保持有效。

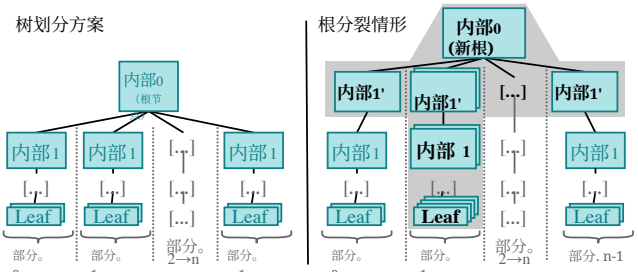



图 8：对树进行分区以支持并发拼接操作。

这些变更将通过拼接线程传播到 NIC 侧树。我们将各个节点更新作为 COPY 拼接操作加入拼接器队列。随后发出一条 CONNECT 拼接命令，该命令会转化为一次指针交换，从而使之前的 COPY 拼接操作生效（参见图6）。

### 3.2.2 NIC 端拼接

网卡侧 DPA 拼接线程执行通过拼接线程队列提交的拼接命令。线程按顺序处理传入的 COPY 和CONNECT 拼接操作，在网卡侧重做主机树上的更改（图 7）。主机已预先计算好 DPA 内存空间中的每个目标地址，因此拼接线程无需分配任何内存。相反，它们会对主机传递的指针执行所提供的命令。为实现并发拼接，我们在第一层内部节点下方对树进行分区（图 8，左）。并行拼接线程（默认四个）分别处理树中的专属分区。主机知晓拼接线程与分区之间的映射关系，并据此分配拼接命令。

拼接操作的顺序以及单线程所有权消除了大多数竞态条件。然而，根级更新可能导致更新冲突：(i) CONNECT 可能指向叶节点的内存地址，而该地址之前并未通过 COPY 拼接创建；或者 (ii) 新复制并连接的节点可能引用下方的节点，而这些节点尚未对网卡侧树可用。因此，我们实施了两项保护措施。首先，我们会延迟执行那些以根节点分裂为直接来源、且目标为相关节点的拼接操作，直到其目标可用为止。具体而言，我们使用 UID 来探测 CONNECT 拼接的目标节点是否已就位。其次，

安装更新后根节点的拼接操作会被队列栅栏阻塞，直到更早的更新完成。为保留分区方案，主机会确保根节点分裂维持均衡分区，并将新的顶层节点分布到所有分区中（图 8，右侧）。

#### 3.2.3 内存回收

网卡侧子树在使用 COPY 和 CONNECT 拼接安装新版本后会变得过时，此时必须最终释放叶节点内存。然而，遍历线程可能仍会进入包含受影响节点的子树。因此，我们使用所有 DPA 线程的传入和传出数据包计数器计算全局纪元值，并对过时节点执行基于纪元的内存回收。

#### 3.2.4 批量加载

批量加载会将一组已排序的键值对划分为 PLA 段，以便在主机内满足配置的叶节点误差界 $\varepsilon_{\text{leaf}}$。每个 PLA 段都会成为一个叶节点。所有叶节点就绪后，它们的前导键将构成构建上层的基础。我们使用子节点的首个键，自底向上应用相同的 PLA 构建来生成更高一级的节点，并强制采用 $\varepsilon_{\text{inner}}$ 的误差界，以调节树扇出度。该递归过程会持续进行，直到只剩下一个内部节点，该节点将成为根节点。

在整个批量加载过程中，主机将 COPY 和 CONNECT 拼接命令加入拼接器队列。拼接线程按照主机入队的命令组装初始树，最后执行根指针拼接，使遍历线程能够访问该结构。主机通过为每次树更新选择一个拼接器队列来确定树分区，从而隐式创建这些分区，因此网卡侧无需执行其他操作。

## 4 评估

本节首先概述实验设置，随后深入分析 DPA-Store。最后，我们将 DPA-Store 与 ROLEX 进行比较 [25],，后者是一种最先进的基于 RDMA 的键值存储。

### 4.1 实验设置

除非另有说明，我们的测试环境由一台服务器和六台客户端机器组成，它们通过一台 100Gb/s Dell PowerSwitch S5232F 交换机连接。该交换机基于以太网，并支持融合以太网上的远程直接内存访问（RoCE）。客户端使用配备 Mellanox ConnectX-5 网卡的双路 AMD Epyc 7301 CPU（32 个核心）。客户端利用 DPDK [11] 发送请求，以绕过操作系统瓶颈。服务器使用一块 BlueField-3 B3140L，运行于配备 32 核 AMD Epyc 9354P 和 128 GB DDR5 内存的系统上。在所有实验中，包括 DPA-Store 在内，BlueField-3 均以网卡模式运行，即允许 DPA 通过直接内存访问访问主机，并禁用 ARM 核心。当未使用 DPA 子系统时，BlueField-3 的行为与 ConnectX-7 网卡相同。这使我们能够在传统的主机/网卡配置上评估相关工作。同时，DPA-Store 在相同的硬件和软件栈上运行，从而确保比较公平。

许多学习型索引对键分布较为敏感。为严格测试我们的学习型索引，我们采用常用的 SOSD 数据集 [2,30]。这些数据集包括合成的稀疏和稠密数据集，以及源自 Facebook（人脸）、亚马逊（亚马逊）、维基百科（维基）和 OpenStreetMap（*OSMC*）工作负载的真实世界数据集。稀疏数据集从完整的 64 位范围中随机选取键，而我们的*dense4x* 数据集则从包含 200M 个键的连续范围中选取 50M 个键。除非另有说明，我们在每次实验前批量加载 25M 个条目，并根据齐夫定律，使用值 $\alpha = 0.99$ 来表示偏斜的键流行度。与大多数相关工作类似，除 64 位键外，我们还使用 64 位值。

对于长时间运行的基准测试（例如 GET），我们对四次运行的吞吐量取平均值；对于短时间运行的基准测试（例如插入新键），则对八次运行取平均值。延迟测量数据采集自所有客户端并进行汇总。通过 MPI 同步客户端的启动时间。测量结果显示，吞吐量的标准差低于 5%，延迟的标准差低于 9%。

### 4.2 DPA-Store 评估

本节中，我们研究内存消耗、学习型索引变化的影响，以及客户端参数和 DPA 线程数与主机线程数对吞吐量和延迟的影响。根据研究结果，我们推导出 DPA-Store 的默认值。此外，我们还分析了不同的 BlueField-3 型号或 BlueField-3 硬件的变更是否会影响性能。

#### 4.2.1 内存消耗

对于插入 5000 万个键值对的情况，我们评估了索引结构相较于主机侧原始 KV 数据的开销（表 1）。由于 人脸 和 *OSMC* 在 $\varepsilon_{\text{inner}} = 4$和 $\varepsilon_{\text{leaf}} = 8$取值下的内存消耗和开销最高，因此我们展示了使用 $\varepsilon = 16$可显著降低它们的内存影响。在以下基准测试中，对于这些数据集，我们为 $\varepsilon$ 选择较大的取值。

与其他学习型索引相比，DPA-Store 选择较小的 $\varepsilon$ 值，以尽量降低在延迟受限的 DPA 内存中，于预测位置附近查找键的成本。例如，ROLEX 使用 $\varepsilon \in \{128, 256\}$，因此能够维护大得多的节点，从而降低每节点内存开销。ROLEX 报告称，对于一个包含 5 亿条数据的数据集，若每个键值对为 16 B，则缓存数据的元数据开销为 6.5% [25]。然而，它要求所有客户端都持有

表 1：64 位键分布和 5000 万条目下的相对开销及网卡侧内存消耗。

| 数据集（5000 万条） | 相对开销索引的 | 总体 DPA 内存需求 |
|---|---|---|
| 稀疏 dense4x | 32% | 207 MB |
| | 26% | 169 MB |
| wiki | 23% | 147 MB |
| amzn | 54% | 346 MB |
| osmc | 74% | 472 MB |
| face | 104% | 672 MB |
| OSMC ($\varepsilon_{in} = \varepsilon_{lf} = 16$) | 35% | 228 MB |
| 人脸 ($\varepsilon_{in} = \varepsilon_{lf} = 16$) | 52% | 332 MB |

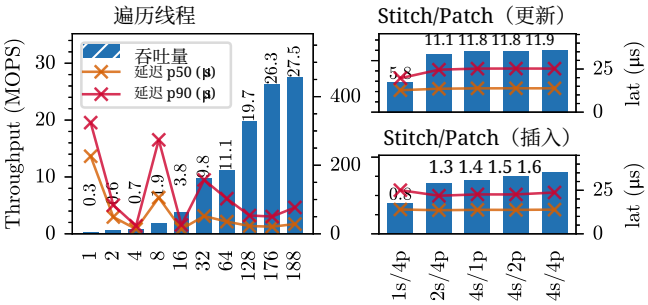



图 9：不同遍历线程、修补器线程和拼接线程数量下 DPA-Store 的吞吐量和延迟。左图展示了 GET 请求的遍历线程数量。右图展示了在包含 176 个遍历线程的纯插入和更新工作负载中，修补器／拼接器线程数量的变化。

一个单独的缓存，除了键的数量之外，还会使内存开销随着客户端数量成比例增加。

### 4.2.2 DPA 和主机线程数量

在本小节中，我们评估遍历线程、修补器线程和拼接线程数量变化对 DPA-Store 性能的影响。图 9 （左）展示了稀疏数据集上键流行度均匀分布的纯 GET 工作负载的吞吐量和延迟。我们观察到，吞吐量会随遍历线程数量成比例增长。遍历线程较少时，延迟的波动较大；线程数量增加后，延迟则趋于稳定。这是因为少量线程会被大量传入数据包压垮，无法及时处理请求。当线程数大于 16 时，会使用多个 DPA 硬件核心，从而降低延迟。

吞吐量在超过 176 个线程后略有提升，这相当于 11 个硬件 DPA 核心。剩余的 13 个 DPA 线程随后位于最后一个物理核心上。我们发现，对于插入操作，在最后一个可用硬件核心上同时运行遍历线程和拼接线程会导致吞吐量下降。这是因为硬件调度器会优先处理网卡门铃事件。这些事件会调用遍历线程

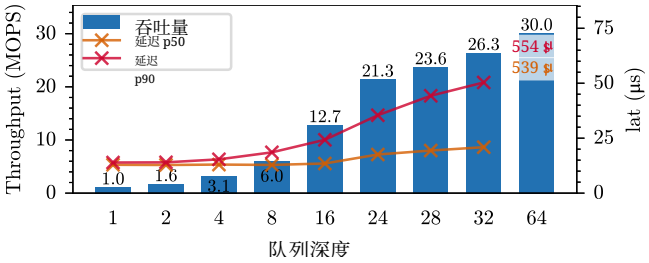



图 10：使用不同客户端队列深度时 DPA-Store 的 GET 吞吐量和延迟。

，因此会阻塞拼接器的执行。我们观察到，插入 操作中，两种类型的 DPA 线程混用一个核心时，吞吐量下降了 14 %。对于 插入 和更新 工作负载，当修补线程和拼接线程超过 4 个时，吞吐量趋于平稳。我们注意到，主机侧修补线程的吞吐量受网卡侧拼接线程限制（详细分析请参见第 4.2.8 节）。

**经验总结：**在 BlueField-3 上优先处理网卡门铃事件，需要将可用的 189 个 DPA 线程划分为：分配在 11 个物理核心上的 176 个遍历线程，以及专用核心上的 4 个拼接线程。此外，我们将 DPA-Store 配置为运行 4 个主机端修补器线程。

### 4.2.3 客户端队列深度

下文将探究客户端队列深度以及由此产生的在途请求总数对吞吐量和延迟的影响。6 个客户端节点各运行 31 个线程。线程发出的并发请求数为 1 到 64 个。网卡端哈希表确保传出数据包及其对应的传入数据包由同一线程处理，并在超时后重新发送未收到确认的请求。图 10展示了客户端队列深度对均匀分布的键分布下 GET 性能的影响。

吞吐量随着客户端队列深度增至 32（对应最多 5，952 个在途请求）而显著提升。此后，吞吐量仍持续上升；然而，由于 DPA-Store 的处理速率跟不上请求提交速率，延迟会升至不可接受的水平。因此，在后续所有实验中，GET 请求的队列深度均设为 32。类似地，我们将插入和范围查询工作负载的队列深度设为 18。集成自适应流量控制留待未来工作。

### 4.2.4 树深度和索引误差界的影响

我们评估树深度和误差界对 ε DPA-Store 吞吐量和延迟的影响。为了展示树深度的影响，本节纳入了 *sparseBig*数据集。批量加载 *sparseBig* （包含 5000 万个键值对）后，树深度为四；而 *sparse* 需要三个树层级。图11 展示了 GET 性能

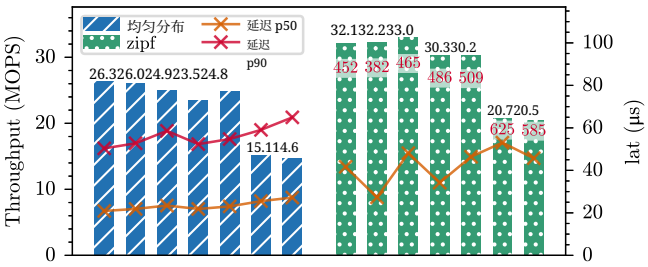



图 11：DPA-Store 在仅 GET 工作负载下的结果。

在均匀和偏斜的键流行度分布下，不同数据集的表现。在均匀访问下，树更深时，由于需要额外访问一个内部节点，我们观察到吞吐量略有下降。为数据集选择更大的 ε 值*face*和 *OSMC*时出现的更显著性能下降，体现了验证模型预测所需的额外缓存行访问所带来的影响（另见第 4.2.6 节中的 DPA-Store 性能模型）。

在键流行度倾斜的情况下，热条目缓存最多可将吞吐量提升 30%，这符合我们在第 3.1.2 节中的预期。不过，尾延迟有所增加，因为部分遍历线程接收的请求比例过高，导致在 DPA 队列中的等待时间更长。

### 4.2.5 B⁺-树对比

我们选择学习型索引作为 DPA-Store 的底层基础数据结构。为说明这一选择的合理性，我们对比学习型索引树与 B⁺-树在数据批量加载后立即执行 GET 操作时的吞吐量和延迟。因此，对于 B⁺-树，我们在构建树节点时设置 $\varepsilon = \infty$，使每个 2 千字节节点都被完全填满，各包含 128 个条目。对于节点内的查找，B⁺-树使用二分查找。两个变体都使用 176 个遍历线程。

图 12 显示，B⁺-树的延迟大多高于学习型索引。使用默认的 ε时，学习型索引在 *sparse*、*sparseBig* 和 *amzn*数据集上实现了更高的吞吐量，尽管 B⁺-树中分布更密集的节点带来了更低的树深度。 *osmc* 数据集配置为使用 $\varepsilon = 16$，从而使

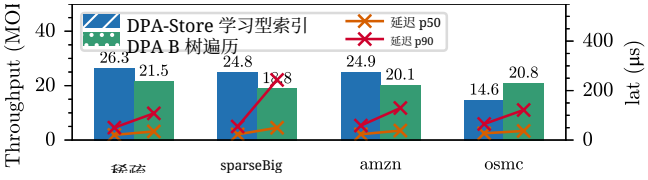



图 12：B⁺-树与学习型 DPA 遍历的对比。

B⁺-树获得了更高的吞吐量。这表明，要使 DPA-Store 的优化学习型遍历有效， ε 必须取较小的值。

### 4.2.6 GET 操作与内存访问分析

前两个小节表明，DPA-Store 的性能很大程度上取决于内存子系统，尤其是 DPA 可寻址 DDR5 内存的访问延迟。先前研究 [6] 报告称，DPA 内存的平均内存访问时间为 465 ns，而通过直接内存访问访问主机内存的平均时间为 910 ns。下面，我们假设默认参数为 $\varepsilon_{\text{inner}} = 4$ 和 $\varepsilon_{\text{leaf}} = 8$，并对完整树遍历的最短持续时间进行建模。

每个内部节点中访问的首个缓存行包含该节点的元数据以及该段的首个键。第二个缓存行包含每个段的模型。接下来的一个或两个缓存行（取决于节点的填充程度）包含分隔键，而第五个缓存行包含子节点指针。假设节点的平均填充率为 50%，则每个内部节点平均需要 4.5 个缓存行。同样，叶节点需要一个缓存行存放元数据，并且最多需要执行三次直接内存访问操作来获取键，再执行一次操作来获取值。请注意，在此分析中，我们假设插入缓冲区为空，因此不需要额外的内存访问。

对于深度为 3 的树，我们访问两个内部节点和一个叶节点，因此总内存访问时间为

$$2 \cdot \delta_{inner} + \delta_{leaf} = 2 \cdot 4.5 \cdot 0.465\mu s + 0.465\mu s + 2 \cdot 0.91\mu s = 6.47\mu s$$

请注意，叶节点键的缓存行在主机上是连续的，并会合并为一次直接内存访问。假设 DPA 调度会将一个线程的计算与另一个线程的内存访问重叠，则在相关缓存行均未被缓存的情况下，最大吞吐量为 $176/6.47\mu s = 27.2$百万次操作每秒。

如果进一步假设根节点的前几个缓存行（节点元数据、段键和模型）已为所有 DPA 线程缓存，并将两次内存访问替换为每次 64*ns* 的三级缓存访问时间，则可计算得出 $\delta_{\text{root}} = 1.2905$ 并获得 31。05百万次操作每秒的最大吞吐量。尽管该模型是一种近似，但我们对 DPA-Store 的评估得出了相似的结果（例如，图 9）。

如第 3.1 节所述，我们优化了遍历路径，以乐观地预取缓存行，使内存访问与计算重叠。由于每个线程的计算能力相对较弱，我们测得这项优化带来了 19% 的改进。

**经验总结：** GET性能受限于对 DPA 内存的访问延迟；如果内存访问延迟能够与标准中央处理器的延迟相当，GET性能将显著提升。当内存延迟为 100ns 时，GET延迟可降至 2.82 $\mu$s 以下，吞吐量可提升至 62 百万次操作每秒以上。降低内存

延迟还可以增加 ε 值，并提高内存利用效率。

### 4.2.7 批量加载性能

我们使用一个包含 5000 万条稀疏数据的数据集和四个拼接线程评估了批量加载吞吐量。批量加载期间会禁用遍历线程。完整的批量加载在主机上完成，平均运行时间为 1,643 毫秒。复制和处理拼接请求累计耗时 1,605 毫秒。批量加载会将 192 MB 的树数据复制到 DPA 内存中，因此主机到 DPA 内存带宽仅为 120 MB/s。带宽较低的原因在于，无法直接从主机向 DPA 内存高效写入目标地址各不相同的细粒度数据。相反，拼接线程必须通过直接内存访问轮询传入的拼接操作，然后先将数据加载到 DPA 线程中，再加载到 DPA 内存中。

**经验总结：** 如果 BlueField-3 支持从主机内存向 DPA 内存进行高效传输，批量加载性能以及下一节所示的插入性能都将大幅提升。

### 4.2.8 插入/更新性能

我们评估了 插入 和 更新操作的吞吐量和延迟，并混合了不同比例的GET 操作，使用 稀疏、亚马逊和 *OSMC* 数据集（见图13）。

对于仅包含更新的补丁，拼接请求不需要将任何数据复制到 DPA 内存。相反，只需更新主机侧的值，并通过拼接命令重置原始 DPA 叶节点的插入缓冲区。在这里，我们的吞吐量最高可达 12.1 百万次操作每秒。我们特别观察到，三个数据集上的插入操作吞吐量都较低，最高仅为 1.7 百万次操作每秒。一旦从叶节点开始向上在主机侧重新训练节点，就会传输 COPY 拼接。对于叶节点，只传输模型参数和直接内存访问地址；而内部节点则会复制完整的分隔键和子指针。正如我们在采用相同拼接方法进行运行时更新的批量加载性能中已经看到的那样，BlueField-3 无法高效地将数据从主机写入设备，这一限制对我们影响很大。运行时拼接的数据带宽

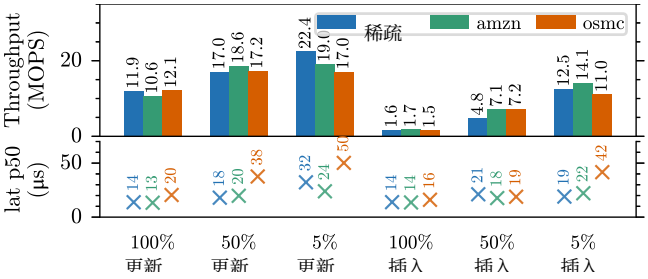



图 13：DPA-Store 更新和插入操作的性能。

在插入密集型场景下，无法超过我们测得的批量加载低速率。尽管轮询拼接器的表现低于预期，但我们认为，后续几代智能网卡有望解决这一问题，并将重点转向最大限度地提升 GET 和范围查询的性能。

### 4.2.9 BlueField-3 型号比较

我们在另一套配置上进行了测试，以评估 BlueField-3 系列的 B3220 模型。与采用单通道内存且配备一个网络端口的 B3140L 不同，B3220 配备双通道 DPA 内存和两个网络端口。不过，DPA 核心的数量和类型保持不变。遗憾的是，当时无法获得可比的客户端节点，因此我们不得不通过 DPA 程序，使用另一块 BlueField-3 B3220 网卡生成请求。这使我们能够将 DPA-Store 的吞吐量提升至饱和，但由于客户端网卡的计算性能不足，无法评估响应延迟。我们还取消了对 B3140L 在途请求数量的限制。因此，两个模型的 DPA-Store 吞吐量均已达到饱和，从而能够进行公平比较。

图14 展示了两块网卡在不同数据集上的仅 GET 工作负载，以及均匀和偏斜的键流行度。我们观察到，对于键分布均匀的稀疏数据集，两块网卡的吞吐量相同。这表明 GET 吞吐量主要受 DPA 内存延迟支配，从单通道内存切换到双通道内存不会直接影响 DPA-Store。DPA-Store 的插入、更新和范围查询吞吐量也没有显著差异。

然而，在仅 GET 工作负载采用倾斜键分布时，B3220 的吞吐量高于 B3140L，达到 48.5 百万次操作每秒，而 B3140L 达到 39.9 百万次操作每秒。此外，我们还运行了 ping 测试，其中接收端 DPA 线程无需访问内存即可返回数据包。对于 ping，B3140L 型号达到 44.9 百万次操作每秒，而 B3220 的吞吐量高出 69%。由于缓存请求只触发一次内存访问，且均匀访问保持不变，这些结果表明，双端口 B3220 配备了更强的数据包匹配硬件，因此能够更快地处理小型请求。

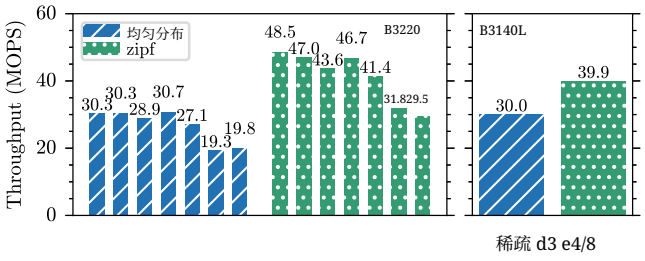



图 14：不同 BlueField-3 模型的 GET 吞吐量。

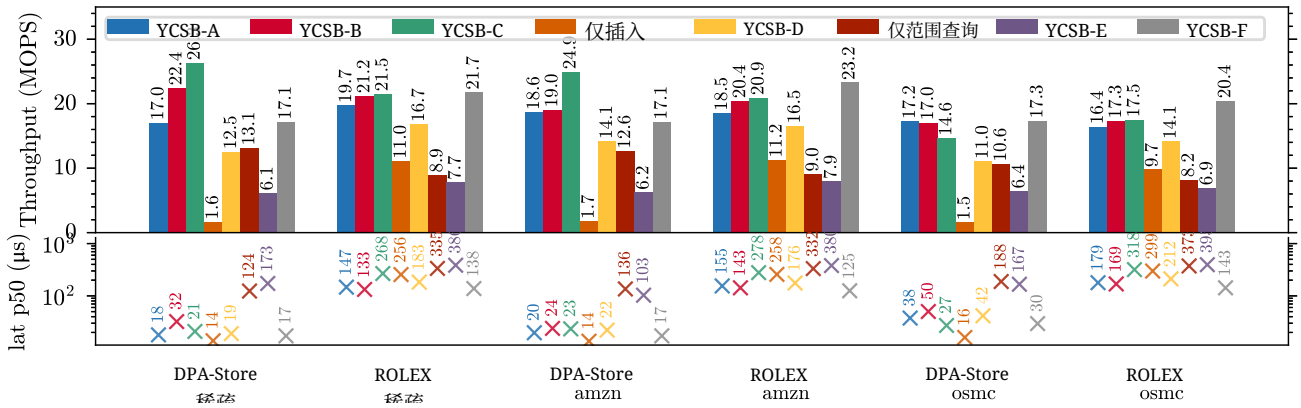



图15：DPA-Store 和 ROLEX 在稀疏、亚马逊和*OSMC* 数据集上的*YCSB*工作负载。

## 4.3 与 ROLEX 的比较

我们将 DPA-Store 与 ROLEX 进行比较 [25], ROLEX 是能够在现代硬件上运行且目前可用的最快 RDMA键值存储，此次比较旨在突出不同架构选择的影响。我们在评估 DPA-Store 时采用的配置相同，对 ROLEX 进行了评估。与 ROLEX 已发表的结果相比，这使其吞吐量更高，但也增加了其延迟。

我们使用 YCSB 工作负载评估了两种方案 [7]，其中包含六种场景：A（50 % 读取，50 % 更新）、B（95 % 读取，5 % 更新）、C（100 % 读取）、D（95 % 读取，5 % 插入）、E（95 % 范围查询，5 % 插入）和 F（50 % 读取，50 % 读-修改-更新）。此外，我们还进行了 100 % 插入和 100 % 范围查询的实验，每次范围查询覆盖 10 个相邻键。实验使用 稀疏、亚马逊 和 *OSMC* 数据集进行，这些数据集使 DPA-Store 具有不同的树深度和误差界配置（参见图 11），并采用均匀键分布进行测量，以降低热条目缓存的影响。

图 15 显示，对于 亚马逊 和 *OSMC* 数据集，在 YCSB-A 和所有范围查询专属工作负载中，DPA-Store 的吞吐量均超过 ROLEX。对于GET专属工作负载，与 ROLEX 相比，DPA-Store 在 稀疏和亚马逊数据集上实现了更高的吞吐量和更低的延迟。然而，由于其较大的 值更适合该数据集，ROLEX 在 *OSMC* 数据集上取得了更好的结果。

ROLEX 在 INSERT工作负载下展现出更高的吞吐量。其客户端能够通过远程直接内存访问将插入请求直接写入服务器内存，性能仅受服务器内存性能的限制。由于采用有状态客户端，ROLEX 可以延迟进行模型重新训练，并将树更新的负载转移到客户端。这种方法能更均匀地分散负载。不过，它需要复杂的客户端逻辑和状态维护。另一方面，DPA-Store 不会给客户端增加状态，只需进行键哈希以分配线程。此处展示插入工作负载仅作比较。在实际应用中，值会在运行前批量加载，而 YCSB工作负载则代表典型用法，其中插入操作最多占 5%。

在新键插入量仅占 GET 或范围请求很小一部分的工作负载中，例如 YCSB-D 和 YCSB-E，DPA-Store 的性能几乎可以达到 ROLEX 的水平。由于更新现有键不会改变内部节点，且无需发送额外的 COPY 拼接，因此原始更新受主机到DPA的瓶颈限制较小。因此，即使更新比例较高（如 YCSB-A），DPA-Store 在亚马逊和*OSMC*上也能超越 ROLEX。

图 15 显示，在所有实验中，DPA-Store 的延迟都低于 ROLEX。DPA-Store 执行 GET 操作时，可能需要经过相对较慢的 PCIe 总线访问主机以进行直接内存访问，但热缓存和插入缓冲区减少了直接内存访问操作的总次数。ROLEX 则要求每项操作都执行远程直接内存访问操作；在未缓存的情况下，甚至需要多次往返通信，因此在途请求越多，争用延迟就越明显。

**经验总结：** DPA-Store 虽然需要更复杂的网卡硬件，但在大多数工作负载下都比最先进的 ROLEX 键值存储更快，不过 INSERT 吞吐量较低。然而，如前文所示，如果下一代 Bluefield 代际配备更好的 DPA 内存接口，并能够高效地将数据从主机传输到 DPA 内存，那么 DPA-Store 在所有工作负载下的吞吐量都可能显著超过 ROLEX。

## 5 结论

我们提出了 DPA-Store，这是一种运行于 SmartNIC 上的键值存储，可从所有操作中消除操作系统延迟。通过仔细分析 BlueField-3 的内部硬件，我们设计了 DPA-Store，以最大限度地提高操作吞吐量。我们将计算密集型的学习型索引维护任务卸载到主机，同时使树遍历保持无锁且高度并发。我们证明了 DPA-Store 的性能可与 ROLEX 相媲美，并且对 Bluefield-3 稍作改动即可显著超越这些成果。

## 致谢

本研究得到了以下项目的资助：大气物理学中的大数据*(BINARY)*，该项目由卡尔蔡司基金会 (P2018-02-003) 资助；此外，本研究还得到了以下项目的资助：*ScalNEXT*：面向超级计算的计算节点数据管理与控制流优化(16ME0688)。

此外，本研究使用了亚琛工业大学复杂电力系统自动化研究所的硬件。

## 参考资料

[1] Berk Atikoglu、Yuehai Xu、Eitan Frachtenberg、Song Jiang 和 Mike Paleczny。大型键值存储的工作负载分析。载于 *ACM SIGMETRICS/PERFORMANCE*计算机系统测量与建模联合国际会议（*SIGMETRICS*），英国伦敦，*6*月*11*日至*15*日，第53–64页，2012年。doi:10.1145/2254756.2254766。[2] Lorenzo Bellomo、Giuseppe Cianci、Luca de Rosa、Paolo Ferragina 和 Mattia Odorisio。整数数据压缩索引、学习型索引与传统索引方法的比较研究。载于 第*23*届实验算法国际研讨会（*SEA*），意大利威尼斯，*7*月*22*日至*24*日，第5:1–5:23页，2025年。doi:10.4230/LIPICS.SEA.2025.5。[3] Qizhe Cai、Shubham Chaudhary、Midhul Vuppalapati、Jaehyun Hwang 和 Rachit Agarwal。了解主机网络协议栈开销。载于 *ACM SIGCOMM* 会议，线上会议，美国，*8*月*23*日至*27*日，第65–77页，2021年。doi:10.1145/3452296.3472888。[4] Zhichao Cao、Siying Dong、Sagar Vemuri 和 David H. C. Du。Facebook的RocksDB键值存储工作负载的特征分析、建模与基准测试。载于 第*18*届*USENIX*文件与存储技术会议（*FAST*），美国加利福尼亚州圣克拉拉，*2*月*24*日至*27*日，第209–223页，2020年。[5] Haibo Chen、Rong Chen、Xingda Wei、Jiaxin Shi、Yanzhe Chen、Zhaoguo Wang、Binyu Zang 和 Haibing Guan。使用远程直接内存访问和硬件事务内存进行快速内存事务处理。 *ACM*计算机系统汇刊，35(1):3:1–3:37，2017年。 doi:10.1145/3092701。[6] Xuzheng Chen、Jie Zhang、Ting Fu、Yifan Shen、Shu Ma、Kun Qian、Lingjun Zhu、Chao Shi、Yin Zhang、Ming Liu 和 Zeke Wang。揭秘数据通路加速器增强型旁路智能网卡。载于 第*32*届*IEEE*国际网络协议会议（*ICNP*），比利时沙勒罗瓦，*10*月*28*日至*31*日，第1–12页，2024年。doi:10.1109/ICNP61940.2024.10858560。

[7] Brian F. Cooper、Adam Silberstein、Erwin Tam、Raghu Ramakrishnan 和 Russell Sears。使用 YCSB 对云服务系统进行基准测试。载于第 1 届 ACM 云计算研讨会（SoCC）论文集，美国印第安纳州印第安纳波利斯，2010 年 6 月 10 日至 11 日，第 143–154 页。doi:10.1145/1807128.1807152。[8] Jialin Ding、Umar Farooq Minhas、Jia Yu、Chi Wang、Jaeyoung Do、Yinan Li、Hantian Zhang、Badrish Chandramouli、Johannes Gehrke、Donald Kossmann、David B. Lomet 和 Tim Kraska。ALEX：一种可更新的自适应学习型索引。载于数据管理国际会议（SIGMOD）论文集，线上会议，6月14日至19日，2020年，第 969–984 页。doi:10.1145/3318464.3389711。[9] Dragonfly。Dragonfly——无限制的内存数据存储。访问日期：2025年8月14日。网址：https: // www.dragonflydb.io/。[10] Paolo Ferragina 和 Giorgio Vinciguerra。PGM 索引：一种具有可证明最坏情况界的全动态压缩学习型索引。VLDB基金会会议论文集，13(8):1162–1175，2020年。doi:10.14778/3389133. 3389135。[11] Linux 基金会。数据平面开发套件（DPDK），2015年。网址：http://www.dpdk.org。[12] Jiake Ge、Huanchen Zhang、Boyu Shi、Yuanhui Luo、Yunda Guo、Yunpeng Chai、Yuxing Chen 和 Anqun Pan。SALI：一种基于概率模型的可扩展自适应学习型索引框架。ACM 数据管理论文集，1(4):258:1–258:25，2023年。doi: 10.1145/3626752。[13] Anuj Kalia、Michael Kaminsky 和 David G. Andersen。高效使用远程直接内存访问（RDMA）实现键值服务。载于 ACM SIGCOMM 会议，美国伊利诺伊州芝加哥，2014 年 8 月 17 日至 22 日，第 295–306 页。doi:10.1145/2619239. 2626299。[14] Arjun Kashyap、Yuke Li 和 Xiaoyi Lu。DPU-KV：探讨 DPU 卸载对边缘内存键值存储的益处。载于第 34 届高性能并行与分布式计算国际研讨会（HPDC）论文集，美国圣母大学，2025 年 7 月 20 日至 23 日，第 11:1–11:14 页。doi:10. 1145/3731545.3731571。[15] Arjun Kashyap、Yuke Li、Darren Ng 和 Xiaoyi Lu。理解新兴 BlueField DPU 的特性。载于第 39 届 ACM 超级计算国际会议（ICS）论文集，美国犹他州盐湖城，2025 年 6 月 8 日至 11 日，第 807–821 页。doi:10. 1145/3721145.3725780。

[16] KeyDB。KeyDB——更快速的 Redis 替代方案。访问日期：2025-11-22。网址：https://docs.keydb.dev。

[17] Elie F. Kfoury、Samia Choueiri、Ali Mazloum、Ali AlSabeh、Jose Gomez 和 Jorge Crichigno。《智能网卡综合综述：架构、开发模型、应用与研究方向》。*IEEE Access*，12:107297–107336，2024年。doi:10.1109/ACCESS.2024.3437203。

[18] Andreas Kipf、Ryan Marcus、Alexander van Renen、Mihail Stoian、Alfons Kemper、Tim Kraska 和 Thomas Neumann。RadixSpline：单遍学习型索引。载于第三届利用人工智能技术进行数据管理国际研讨会论文集（*aiDM@SIGMOD*），美国俄勒冈州波特兰，*6*月*19*日，第5:1–5:5页，2020年。doi:10.1145/3401071.3401659。[19] Tim Kraska、Alex Beutel、Ed H. Chi、Jeffrey Dean 和 Neoklis Polyzotis。学习型索引结构的价值。载于数据管理国际会议论文集（*SIGMOD*），美国德克萨斯州休斯敦，*6*月*10*日至*15*日，第489–504页，2018年。doi:10.1145/3183713.3196909。[20] Viktor Leis、Alfons Kemper 和 Thomas Neumann。自适应基数树：面向主存数据库的巧妙索引。载于第*29*届*IEEE*数据工程国际会议（*ICDE*），澳大利亚布里斯班，*4*月*8*日至*12*日，第38–49页，2013年。doi:10.1109/ICDE.2013.6544812。[21] Viktor Leis、Florian Scheibner、Alfons Kemper 和 Thomas Neumann。实用同步的ART。载于第*12*届新型硬件上的数据管理国际研讨会（*DaMoN*）论文集，美国加利福尼亚州旧金山，*6*月*27*日，第3:1–3:8页，2016年。doi:10.1145/2933349.2933352。[22] Bojie Li、Zhenyuan Ruan、Wencong Xiao、Yuanwei Lu、Yongqiang Xiong、Andrew Putnam、Enhong Chen 和 Lintao Zhang。KV-DIRECT：采用可编程网卡的高性能内存键值存储。载于第*26*届操作系统原理研讨会（*SOSP*）论文集，中国上海，*10*月*28*日至*31*日，第137–152页，2017年。doi:10.1145/3132747.3132756。[23] Pengfei Li、陈宇、Yu Hua 和 Pengfei Zuo。Finedex：面向可扩展并发内存系统的细粒度学习型索引方案。*VLDB*学报，15(2):321–334，2021年。doi:10.14778/3489496.3489512。[24] Pengfei Li、Yu Hua、Pengfei Zuo、Zhangyu Chen 和 Jiajie Sheng。一种高性能、面向远程直接内存访问的面向解耦内存系统的学习型键值存储。ACM 存储学报，19(4):30:1–30:30，2023年。doi:10.1145/3620674.[25] Pengfei Li、陈宇、Pengfei Zuo、Zhangyu Chen 和 Jiajie Sheng。ROLEX：面向解聚内存的可扩展 RDMA 定向学习型键值存储。载于第21届 USENIX 文件与存储技术会议（FAST），美国加利福尼亚州圣克拉拉，2月21日至23日，第99–114页，2023年。[26] Pengfei Li、Hua Lu、Rong Zhu、Bolin Ding、Long Yang 和 Gang Pan。DILI：分布驱动的学习型索引。VLDB基金会会议论文集，16(9):2212–2224，2023年。doi:10.14778/3598581.3598593。[27] Hyeontaek Lim、Dongsu Han、David G. Andersen 和 Michael Kaminsky。MICA：一种快速内存键值存储的整体方法。载于第11届 USENIX 网络系统设计与实现研讨会（NSDI）论文集，美国华盛顿州西雅图，4月2日至4日，第429–444页，2014年。[28] Junyi Liu、Aleksandar Dragojevic、Shane T. Fleming、Antonios Katsarakis、Dario Korolija、Igor Zablotchi、Ho-Cheung Ng、Anuj Kalia 和 Miguel Castro。Honeycomb：基于现场可编程门阵列的智能网卡上的有序键值存储加速。IEEE计算机汇刊，73(3):857–871，2024年。doi:10.1109/TC.2023. 3345173。[29] Yandong Mao、Eddie Kohler 和 Robert Tappan Morris。快速多核键值存储的缓存优化技巧。载于欧洲计算机系统会议论文集——第七届 EuroSys 会议（EuroSys），瑞士伯尔尼，4月10日至13日，第183–196页，2012年。doi:10.1145/2168836.2168855。[30] Ryan Marcus、Andreas Kipf、Alexander van Renen、Mihail Stoian、Sanchit Misra、Alfons Kemper、Thomas Neumann 和 Tim Kraska。学习型索引基准测试。VLDB基金会会议论文集，14(1):1–13，2020年。doi:10.14778/3421424.3421425。[31] Ryan Marcus、Emily Zhang 和 Tim Kraska。Cdf-shop：探索并优化学习型索引结构。载于数据管理国际会议（SIGMOD）论文集，线上会议，6月14日至19日，第2789–2792页，2020年。doi: 10.1145/3318464.3384706。[32] Memcached。Memcached——一种分布式内存对象缓存系统。访问日期：2025年8月14日。网址：https://memcached.org。[33] Rolf Neugebauer、Gianni Antichi、José Fernando Zazo、Yury Audzevich、Sergio López-Buedo 和 Andrew W.

Moore。了解端主机网络的 PCIe 性能。载于 *ACM* 数据通信特别兴趣小组会议（*SIGCOMM*）论文集，匈牙利布达佩斯，*8*月*20*日至*25*日，第327–341页，2018年。doi:10.1145/3230543.3230560。

[34] 英伟达。NVIDIA BlueField-3 数据路径加速器（DPA）。访问日期：2025年8月14日。网址：https://docs.nvidia.com/doca/sdk/dpa+subsystem/index.html。

[35]英伟达。NVIDIA BlueField 网络平台。访问日期：2025年8月14日。网址：https://www.nvidia.com/en-us/networking/products/data-processing-unit/。

[36] Boris Pismenny、Liran Liss、Adam Morrison 和 Dan Tsafrir。通用网卡上内存的优势。载于第 *27* 届 *ACM* 编程语言与操作系统架构支持国际会议（*ASPLOS*），瑞士洛桑，第1130–1147 页，2022年。 doi:10.1145/3503222.3507711。[37] Redis。Redis——实时数据平台。访问日期：2025年8月14日。网址：https://redis.io。[38] Feng Ren、Mingxing Zhang、Kang Chen、Huaxia Xia、Zuoning Chen 和 Yongwei Wu。借助 SMART 扩展内存解耦应用程序。载于 第 *29* 届 *ACM* 编程语言与操作系统架构支持国际会议论文集，第 *1* 卷（*ASPLOS*），美国加利福尼亚州拉霍亚，第 351–367 页，2024年。doi:10.1145/3617232.3624857。[39] Henry N. Schuh、Arvind Krishnamurthy、David E. Culler、Henry M. Levy、Luigi Rizzo、Samira Man-abi Khan 和 Brent E. Stephens。CC-NIC：面向网卡的缓存一致性接口。载于 第 *29* 届 *ACM* 编程语言与操作系统架构支持国际会议论文集，第 *1* 卷（*ASPLOS*），美国加利福尼亚州拉霍亚，第 52–68 页，2024年。doi:10.1145/3617232.3624868。[40] Shangyi Sun、Chunpu Huang、Rui Zhang、Lulu Chen、Yukai Huang、Ming Yan 和 Jie Wu。使用数据处理单元优化系统的全面研究。*CoRR*，abs/2301.06070，2023年。 doi:10.48550/ARXIV.2301.06070。[41] Shangyi Sun、Rui Zhang、Ming Yan 和 Jie Wu。SKV：由智能网卡卸载的分布式键值存储。载于*IEEE* 集群计算国际会议（*CLUSTER*），德国海德堡，*9*月*5*日至*8*日，第 1–11 页，2022年。 doi:10.1109/CLUSTER51413.2022.00016。

[42] Lasse Thostrup、Daniel Failing、Tobias Ziegler 和 Carsten Binnig。面向数据库管理系统的蓝色领域 DPU 在高速网络上的评估。载于《使用现代处理器和存储架构加速分析与数据管理系统国际研讨会》（ADMS@VLDB），澳大利亚悉尼，9月5日，第1–10页，2022年。[43] 童奥阳、Yu Hua 和陈梦磊。DALDEX：通过增量学习实现的 DPU 加速持久化学习型索引。载于第39届 ACM 超级计算国际会议（ICS）论文集，美国犹他州盐湖城，6月8日至11日，第535–549页，2025年。doi:10.1145/3721145.3725750。[44] 王静、王庆、张宇昊和舒继武。Deft：一种适用于解聚内存的可扩展树索引。载于第二十届欧洲计算机系统会议（EuroSys）论文集，荷兰鹿特丹，第886–901页，2025年。doi:10.1145/3689031.3696062。[45] 王瑞宏、王建国、Prishita Kadam、M. Tamer Özsu 和 Walid G. Aref。DLSM：一种用于内存解聚的 LSM 索引。载于第 39 届 IEEE 国际数据工程大会（ICDE），美国加利福尼亚州阿纳海姆，2023年4月3日至7日，第2835–2849页，2023年。doi:10.1109/ICDE55515.2023.00217。[46] Xingda Wei、Rong Chen、Haibo Chen 和 Binyu Zang。XSTORE：利用远程学习型缓存实现的高速基于远程直接内存访问的有序键值存储。《ACM 存储学报》，17(3):18:1–18:32，2021年。doi:10.1145/3468520。[47] Xingda Wei、程荣新、杨宇涵、Rong Chen 和 Haibo Chen。探究旁路智能网卡加速分布式系统的特性。载于第17届 USENIX 操作系统设计与实现研讨会（OSDI），美国马萨诸塞州波士顿，2023年7月10日至12日，第987–1004页。[48] 吴家成、张勇、陈世敏、陈宇、王进和邢春晓。具有精确位置的可更新学习型索引。《VLDB基金会会议论文集》，14(8):1276–1288，2021年。doi:10.14778/3457390.3457393。[49] 吴兴波、倪凡和 Song Jiang。Wormhole：一种用于内存数据管理的高速有序索引。载于第十四届 EuroSys 会议论文集，德国德累斯顿，2019年3月25日至28日，第18:1–18:16页。doi:10.1145/3302424.3303955。[50] 杨俊成、岳尧和 K. V. Rashmi。对推特数百个内存键值缓存集群的大规模分析。《ACM 存储学报》，17(3):17:1–17:35，2021年。doi:10.1145/3468521。

[51] Shunkang Zhang、Ji Qi、Xin Yao 和 André Brinkmann。HYPER：一种通过混合构建实现的高性能、内存高效学习型索引。 *ACM* 管理数据会议论文集，2(3):145，2024年。 doi:10.1145/3654948。

[52] Yiwen Zhang、Guokuan Li、Jiguang Wan、Junyue Wang、Jun Li、Ting Yao、Huatao Wu 和 Daohui Wang。Dow-kv：一种基于 DPU 卸载、针对写入优化的解耦持久内存键值存储。在*IEEE* 集群计算国际会议 *(CLUS-TER)*，美国新墨西哥州圣菲，*10* 月 *31* 日至 *11* 月 *3* 日，第 271–283 页，2023年。 doi:10.1109/CLUSTER52292.2023.00030。

[53] Wenbin Zhu、Zhaoyan Shen、Qian Wei、Renhai Chen、Xin Yao、Dongxiao Yu 和 Zili Shao。HIDPU：一种面向解耦存储系统、由 DPU 驱动的混合索引方案。在第 *23* 届 *USENIX*文件与存储技术会议 *(FAST)*，美国加利福尼亚州圣克拉拉，*2* 月 *25* 日至 *27* 日，第 271–285 页，2025年。

[54] Tobias Ziegler、Sumukha Tumkur Vani、Carsten Binnig、Rodrigo Fonseca 和 Tim Kraska。为支持快速 RDMA 网络设计分布式树形索引结构。发表于国际数据管理会议（*SIGMOD*）论文集，荷兰阿姆斯特丹，*2019* 年 *6* 月 *30* 日至 *7* 月 *5* 日，第 741–758 页，2019 年。 doi:10.1145/3299869.3300081。