读懂 PCIe 与 RDMA 的网络基础

从一次 *KV Cache* 搬运出发，看清数据路径、完成语义和性能数字

当程序说“把这块 KV 数据传过去”，机器究竟做了什么？800G 是网络段的能力，应用吞吐由整条数据路径和软件流程共同决定。要把性能说清楚，需要分别理解网络协议、机内互连、内存访问和程序完成。

本文先回答三个问题：

1. 一块 KV 数据怎样穿过内存、PCIe 和网络，DMA、WRITE 与 READ 分别做什么？
2. 程序怎样通过 MR、QP、CQ 指挥网卡，什么时候才可以安全消费数据？
3. 为什么 800G 网卡不等于 800G 应用吞吐，怎样结合拓扑、并发和计时口径判断瓶颈？

贯穿例子是假设机器 A 把一块 8 MiB 的 KV 数据交给机器 B。拓扑和性能算例均为教学假设，不是作者测试或独立复现；本文不公开未经核验的部署配置和原始实验数字。

1. 字节怎样穿过内存 PCIe 和网络

1 先把四个层次分开

1.1 应用想做什么

在最上面，应用关心的是：保存一个 KV block、读回一个对象、把 prefill 的结果交给 decode。这里的 put/get 是应用 API；它们可能还包含找节点、分配空间、检查版本、更新索引和发布状态。

所以，**put** 不天然等于一次 **RDMA WRITE**，**get** 也不天然等于一次 **RDMA READ**。一种实现可以用 WRITE 推送数据，另一种可以通知对方用 READ 拉取，也可以用 SEND/RECV。需要看具体代码，不能从 API 名字倒推硬件操作。

1.2 网络负责怎样跨机器

网络把数据从一台机器的网卡送到另一台机器的网卡。网卡是 NIC，Network Interface Card；能执行 RDMA 的网卡常叫 RNIC。InfiniBand 文档也常用 HCA，Host Channel Adapter，描述主机侧适配器。

对以太网，可以先掌握这几个词，不必背完整的七层模型：

- Ethernet，以太网：链路上以“帧”传输数据，MAC 地址用于这层的交付；二层交换机通常据此转发
- IP：给跨网络传递的数据包提供寻址与路由。它不替你保证每个包都送达
- TCP：在 IP 之上提供可靠、有序的字节流。应用要自己约定一条消息在哪里结束
- UDP：提供数据报边界，但自身不保证送达、顺序或重传。上层协议可以另建可靠性

TCP 的“可靠”不是不会出错，而是协议会检测问题并重传，最终交付或报告失败。它也不意味着对方应用已经处理完数据。TCP 标准、UDP 使用指南

8 MiB 的 KV 对象一般不会作为一个巨大网络包发送。协议会把它分成多个包，接收端再按协议放入目标缓冲。MTU 是 Maximum Transmission Unit，即一层链路允许的最大传输单元；以太网 MTU、RDMA 的路径 MTU、应用对象大小是不同概念。包越小，包头和逐包处理占比通常越高；消息大也不意味着应用可以忽略分包与重传。

1.3 PCIe 负责怎样进出设备

PCIe 是机器内部的高速互连。网卡、GPU 和 CPU 的内存系统之间，常通过 PCIe 传递事务。它不是 TCP 下面再加的一层网络协议，而是网卡与本机其他部件交互的另一段路径。

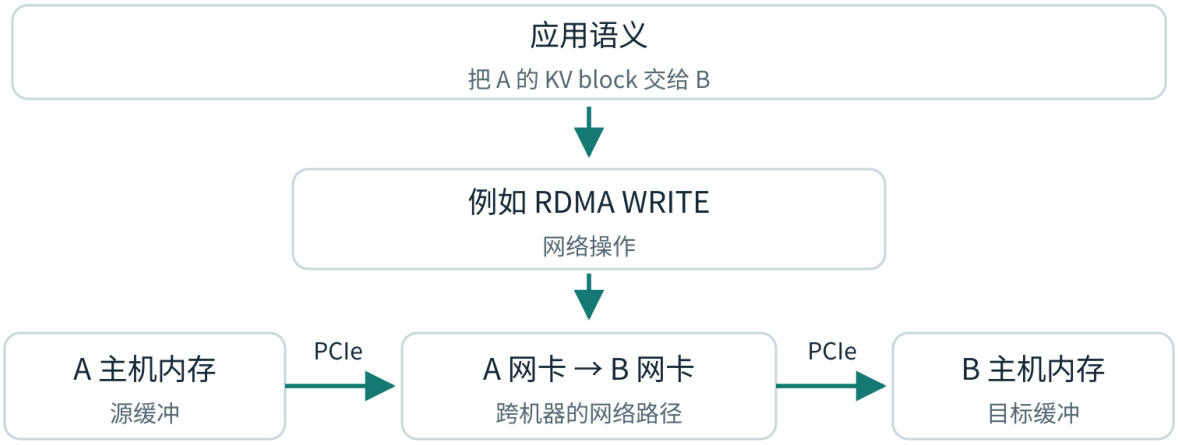


图 1 应用语义 网络操作 跨机路径与机内 **PCIe** 路径

网络交换机和 PCIe switch 都有 switch 这个词，但转发的是不同类型的包，属于不同的互连系统。后面说“交换机”，要先问是哪一种。

1.4 内存也有不同的位置

DRAM 在本文指主机内存；HBM 是 GPU 旁的高带宽显存。它们位于不同的物理内存与访问域；即使使用统一虚拟寻址，访问能力、路径与同步语义仍不同。跨主机、跨设备的搬运还涉及地址转换、访问权限和同步。

“经过 CPU”尤其容易歧义：经过 CPU 封装内的 I/O、缓存或内存控制器，不等于 CPU 核心执行了逐字节复制。先分清经过哪条硬件路径和谁执行搬运指令。

2 DMA 和 RDMA 到底省掉了什么

2.1 普通网卡原本就会 DMA

DMA 是 Direct Memory Access，直接内存访问。CPU 先准备缓冲区、描述信息和访问映射，再让设备发起内存读写；搬运大量 payload 时，不需要 CPU 核心为每个字节执行 load/store。

这里的“直接”不是没有控制器、没有地址转换，也不是设备能随意读全机内存。设备看到的 DMA 地址可能经过 IOMMU 映射，与程序的虚拟地址不同。Linux 的 DMA 文档专门区分了 CPU 虚拟地址、物理地址和设备使用的 DMA 地址。Linux DMA 地址说明

普通 TCP 网络也通常由网卡 DMA 收发。RDMA 的价值不能解释成“TCP 用 CPU 搬每个字节，RDMA 才让网卡搬”。常规 socket 路径还可能有内核缓冲与用户缓冲之间的复制、协议栈处理和系统调用；实际复制次数会受操作系统和零拷贝机制影响。

2.2 RDMA 把远端内存访问做成网卡能执行的操作

RDMA 是 Remote Direct Memory Access，远程直接内存访问。双方先准备资源、授权可访问的内存区域，再由网卡执行数据路径。普通单边 READ/WRITE 不需要远端应用线程逐次接收、解包、再复制这一块数据。

这就是“单边”的含义：每次数据操作不需要远端 **CPU** 对应执行一次收发动作。但远端仍要参与资源建立、内存授权、生命周期、元数据和错误处理。CPU 没有消失，只是不用守在每一块 payload 的搬运路径上。NVIDIA RDMA 操作概览

2.3 零拷贝是一个有边界的说法

零拷贝通常指省掉某个中间缓冲间的显式复制，不是字节不再移动：网卡仍要读源内存、发网络包、写目标内存。

因此读到“复用注册和直接使用用户缓冲”时，应拆成两个优化：

1. 复用注册：不再每次重新建立和撤销设备访问内存的资源
2. 直接使用用户缓冲：减少用户缓冲与暂存缓冲之间的复制

它们能减少软件成本和内存流量，但不会增加 PCIe 的物理容量。分析一个“零拷贝”方案时，最有用的问题是：到底省掉了哪一次、哪两个缓冲之间的复制？

3 跟着一次 WRITE 和 READ 走完数据路径

本节假设双方使用可靠连接型 RDMA，即 RC，Reliable Connection；缓冲、权限和连接都已经建立。先只考虑主机内存，GPU 在第三部分单独讨论。

3.1 A 发起 RDMA WRITE 就是在推数据

A 给本机网卡一份描述：从 A 的这个缓冲，取这么多字节，写到 B 授权的这个地址。



图 2 A 发起 RDMA WRITE 推送源内存数据到 B

这不是 CPU A 在执行一条能直接寻址 CPU B 内存的普通 store 指令。它是 A 请求本地 RNIC 执行一个远程操作，两端 RNIC 再把它变成本地内存事务和网络传输。

3.2 A 发起 RDMA READ 就是在拉数据

A 发起 READ 时，要读的是 B 的源缓冲，结果落到 A 的目标缓冲：



图 3 A 发起 **RDMA READ** 请求方向与 **payload** 方向相反

READ 也从发起方的发送队列提交；“发送队列”说的是发送操作请求，不保证大块数据也从本机流向远端。由谁发起操作、请求往哪走、payload 往哪走，是三个不同问题。

READ 和 WRITE 不保证性能相同。READ 要先取回请求的数据，可并行的未完成读取数、响应路径和网卡资源都会影响吞吐。看到 put/get 速度不同，不要先认定是“网线一个方向更慢”。NVIDIA READ 与 WRITE 操作说明

3.3 PCIe 上的 READ 和 RDMA READ 不是同一件事

在一次 RDMA WRITE 里，A 网卡为了取源数据，会发起本地 PCIe Memory Read：小的读请求从网卡发往主机，带数据的 Completion 从主机返回网卡。B 网卡写目标时，使用携带数据的 PCIe Memory Write。

所以，WRITE 这个名字描述的是远程操作，内部仍然包含源端读取。PCIe 的 Completion 是对某类 PCIe 事务的回复；它与稍后要讲的 verbs Completion Queue 不是同一个东西。Intel I/O 事务与数据方向说明

为了画清路径，本文把主机内存简写成 DRAM；严格来说，具备 Intel DDIO 等机制的平台，DMA 数据也可能先进入 CPU 的末级缓存。**RDMA** 完成不能翻译成“已经写到 **DRAM** 芯片”，更不能翻译成“已经持久化到磁盘”。 Intel DDIO 说明

2. 程序怎样提交工作并安全消费数据

4 程序如何告诉网卡去做事

把 verbs 理解成“向 RDMA 设备提交工作的编程接口”。下面这些名词描述资源和工作队列，不是新的网络线缆。

4.1 MR 让网卡知道哪块内存可以碰

MR 是 Memory Region，内存区域。注册 MR 时，程序指定范围和权限，建立设备访问所需的映射与保护。常见注册方式涉及固定页面及设备地址映射，也存在按需分页等机制，所以不宜把所有注册方式都等同于永久锁住物理页。

- lkey 是 local key，本地访问使用的键，供网卡验证本地缓冲
- rkey 是 remote key，远端访问使用的键，与目标地址和长度一起描述授权访问
- PD 是 Protection Domain，保护域，用来隔离和约束相关设备资源

远端地址通常采用注册区域的地址表示，不能当作裸物理地址；verbs 也支持自定义设备虚拟地址或零基偏移。收到别人的普通 C/C++ 指针不等于获得 RDMA 访问权。发起方需要通过应用协议取得有效的远端地址、rkey、长度和使用约定。rkey 也不是网络加密密钥；它承担的是设备访问保护中的角色。

MR 不能在仍有操作使用它时随意注销，缓冲也不能过早释放或复用。这是 KV block 回收、引用计数和并发读取会与 RDMA 直接发生关系的地方。rdma-core 内存注册接口

4.2 QP 是工作入口 CQ 是完成出口

QP 是 Queue Pair，队列对，通常包含 SQ 与 RQ：

- SQ，Send Queue，发送队列：提交 SEND、RDMA WRITE、RDMA READ 等操作
- RQ，Receive Queue，接收队列：为需要接收请求的操作提前准备接收资源
- WR，Work Request，工作请求：应用提交的描述，例如地址、长度、操作类型
- WQE，Work Queue Element：设备工作队列中的工作项，可把它理解成 WR 对应的设备侧描述
- CQ，Completion Queue，完成队列：网卡写入操作完成结果的地方
- CQE，Completion Queue Entry：其中一条结果，程序据此检查成功或错误



图 4 程序提交工作 网卡执行并通过 CQ 报告结果

投递 RQ 是预备接收资源，真正的接收处理还要等对端操作到来。poll 是轮询：程序主动检查 CQ。也可以使用完成事件通知。轮询减少等待唤醒的成本，但会占用 CPU；这也是“RDMA 绕过远端逐次处理”不等于“系统不耗 CPU”的一个具体例子。

一条 QP 可以同时有多个在途工作；一个 CQ 也可以服务多个 QP。QP 数量、每个 QP 的队列深度、应用请求并发数不是同一个参数。创建 QP、提交 WR、轮询 CQ

4.3 三种操作对远端的要求不同

操作	谁决定数据落在哪	远端要预先投递接收 **WR** 吗	远端怎样获知
SEND/RECV	接收方投递的接收缓冲	要	成功接收会产生接收完成记录
普通 RDMA WRITE	发起方指定已授权的目标地址	不需要	操作本身通常不产生远端接收 CQE，需应用约定通知
普通 RDMA READ	发起方指定本地目标和远端源地址	不需要	远端应用通常不为此次 READ 收到接收 CQE

操作	谁决定数据落在哪	远端要预先投递接收 **WR** 吗	远端怎样获知
WRITE_WITH_IMM	数据写入指定远端地址，另带一个短通知值	要	远端消耗接收资源并得到带 immediate 值的完成记录

WRITE_WITH_IMM 的大块数据仍落在指定的远端地址，不是 RQ 中那个普通接收缓冲；它借用接收完成机制送通知。上表讨论的是普通 verbs 用法，不代表任何 RDMA 传输类型都支持所有操作。提交发送操作与 immediate、投递接收资源

还有一个常见误解：并非每个发送 WR 都必须单独生成成功 CQE。程序可以选择只给部分操作请求完成通知，以减少开销；资源何时能回收则要依照相应顺序与完成规则。没有看到某条 CQE，不一定等于那条 WR 没有执行。

5 已提交 已完成 已可用是三件事

最容易写错的 RDMA 程序，往往不是不会发数据，而是太早把缓冲交给下一个消费者。

5.1 提交成功不等于传输完成

`ibv_post_send` 成功返回，说明工作请求被接受，不说明 payload 已经到达远端。对于普通非 inline 发送，也就是没有把数据直接放进工作描述中的情况，源缓冲仍可能被网卡读取，不能马上覆盖。READ 的本地目标缓冲也要在对应成功完成之后再使用。提交发送的缓冲生命周期

5.2 RC 的完成不等于远端应用已经处理

成功 CQE 的含义要结合操作类型和传输类型。对本地 READ，成功完成表示请求的数据已按该操作的完成语义写入本地目标缓冲。对 RC WRITE，成功完成包含可靠传输所需的远端协议确认，即 ACK，Acknowledgment，不能把它解释成“只是本机把包发出去了”。

但网卡层的确认没有替远端应用执行“把这个 KV block 标记为 ready”“更新对象索引”“解除锁”“让 GPU 开始消费”。远端收到数据与应用对象可用之间，仍有发布协议。

普通 WRITE 没有自动的远端接收 CQE。系统可以安排后续 SEND、WRITE_WITH_IMM 或控制通道通知，但必须把“数据完成”与“通知允许消费”的先后关系设计清楚。RPC 是 Remote Procedure Call，远程过程调用；它可以承担这些控制动作，不必承载大块 payload。NVIDIA RC 及操作完成说明

5.3 顺序也有作用范围

同一 QP 的顺序规则很重要，但不要简化成“先 post 的所有事情在全系统都先完成”：RDMA READ/原子操作与后续操作之间的依赖可能需要 fence；不同 QP 之间不能凭提交先后推导统一顺序。CPU 的并发内存模型和 GPU 的执行流，又有各自的同步要求。WR fence 与操作标志

注册区域若启用了 relaxed ordering，即放宽写入顺序，也不能靠逐字写入的先后自行构造 ready 协议；接口规定的完成语义仍然成立。MR 的 relaxed ordering 说明

尤其不能让 QP 1 写数据、QP 2 写 ready 标记，然后仅因为 CPU 先提交了 QP 1，就断言任何观察者看见 ready 时一定能安全读取数据。

可以记住一条工程规则：生产者完成数据，按相应设备与内存模型建立顺序，再发布给消费者；消费者取得就绪条件后，才读并最终释放资源。这是一条设计原则，不是可直接替代 verbs/CUDA 同步 API 的伪代码。

6 InfiniBand RoCE 和 TCP 各是什么位置

6.1 RDMA 是操作模型 不等于某一种网络

常见的 RDMA 网络实现包括：

实现	承载方式	初学时应记住什么
InfiniBand，IB	InfiniBand 的链路与网络体系	使用专门的 IB fabric；不能直接当作普通以太网加 TCP
RoCE v1	RDMA over Converged Ethernet，直接承载在以太网二层	不使用 IP 路由承载该 RDMA 数据流
RoCE v2	以太网上的 IP + UDP 封装	可以进行 IP 路由；RDMA 数据路径不是 TCP
iWARP	基于 TCP/IP 的 RDMA 协议族	RDMA 也可以建立在 TCP 之上，所以“RDMA 必然不用 TCP”不成立

RoCE v2 中，UDP 是承载外壳，可靠连接的序号、确认与重传由 RDMA 传输层机制承担。不能从“底下有 UDP”推出“这个 RC RDMA 操作不可靠”。同样，IB/RoCE 上还有不同传输服务，不能说所有 RDMA 操作都天然可靠。NVIDIA RoCE 说明、IETF RDMAP 标准

常见混合形态是：控制面用 TCP/RPC 交换地址与元数据，数据面用 RDMA 搬大块 KV。因此日志中看见 TCP，不足以证明大块数据走 TCP；看见 RDMA 连接，也不说明所有流程都绕过 CPU。

6.2 交换机为什么会把延迟拉长

假设多台机器同时向一台机器发送 KV 数据，输入流量超过某个出口的服务能力，包会进入队列。队列先增加延迟，缓冲耗尽后可能丢包。这种“多发一收”在分布式推理里值得特别关注。

需要分清两种控制：流控处理接收资源或相邻链路能否继续收；拥塞控制让发送端根据路径压力调节发送速率。它们相关，但不是一回事。

RoCE 部署中常见 PFC，即 Priority Flow Control，按优先级暂停上游发送；也常用 ECN，Explicit Congestion Notification，让交换机标记拥塞，接收端 RNIC 再生成 CNP，Congestion Notification Packet，把信号反馈给发送端降速。PFC 可能传播暂停、扩大阻塞，不能当作没有代价的加速开关。具体是否使用 PFC、如何实现丢包恢复和拥塞控制，要看网卡与网络方案，不能一概认定“RoCE 必须绝不丢包”。NVIDIA RoCE 网络控制

这解释了为什么单流裸测很快，多机一起跑却出现长尾：平均带宽、拥塞队列、重传和应用排队可能同时变化。只看网络口是否是 800G，无法解释这些差异。

3. 怎样从拓扑和计时口径判断性能

7 沿假想拓扑理解 Gen5 上行和 GPU HBM

7.1 带宽要沿实际路径算

Gen 表示 PCIe 代际，x16 表示链路宽度为 16 条 lane。每条 lane 有两个相反方向的传输能力；全双工允许两个方向同时工作，并不让一次单向搬运自动拥有两倍带宽。

对 Gen5 x16，可以计算单方向的编码后理论带宽：

32 GT/s × 16 lane × 128/130 ÷ 8
≈ 63.02 GB/s
≈ 504.12 Gb/s

这是扣除 Gen5 编码开销后的单方向理论值，尚未扣除事务与协议开销。GT/s 是传输速率单位，不能跨代际无条件当作 payload Gb/s；Gen6 使用 PAM4 和 FLIT 等不同机制，不应照搬 128/130。
PCI-SIG 代际速率与编码资料

以下是一种假想拓扑，用来说明路径约束，不代表任何已核验部署：

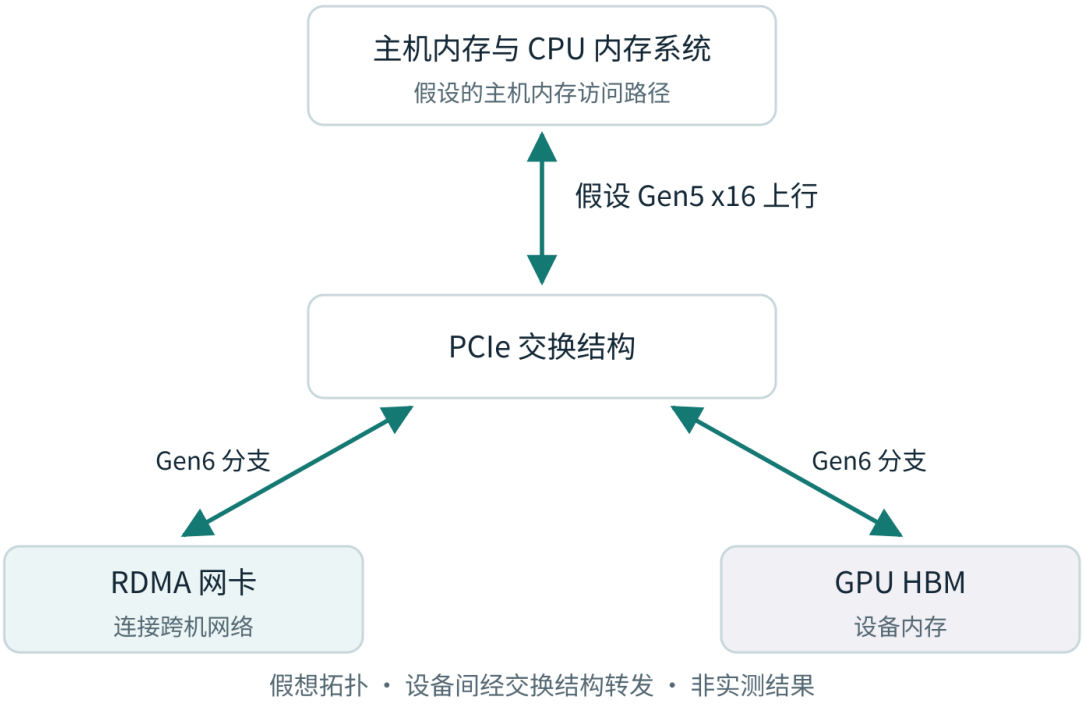


图 5 用假想拓扑区分主机内存上行与设备间路径

假设网卡侧连接是 Gen6，读写主机内存仍要经过 Gen5 上行。于是 800 Gb/s 的网络能力无法直接转换为同样的主机内存吞吐。把某个端口升级得更快，不会让路径上的另一段一起升级。

例如，若某次同条件测量得到 450 Gb/s，它约等于 504.12 Gb/s 的 89%。这只是解释比值的假想数字，并非本文实测。这样的结果可以与 Gen5 上限约束相容，但不能把全部差额精确分配给 PCIe 包头，更不能排除内存、网卡或软件同时限制吞吐。比较数据还需保证方向、对象大小、并发和计时口径一致。

7.2 NUMA 说明内存不是到处一样近

NUMA 是 Non-Uniform Memory Access，非一致内存访问。多路服务器里，不同 CPU 插槽各有其相连的内存资源。网卡挂在一个 NUMA 节点附近，而缓冲分配在另一个节点，访问就可能还要跨 CPU 插槽互连，带来额外延迟和共享带宽压力。

所以需要同时看三件事：网卡在哪里，数据页在哪里，提交与轮询线程在哪里。把线程绑到本地节点，不自动证明缓冲也分配在本地节点。

即使每张 NIC 各有独立的 Gen5 上行，多卡扩展时仍可能竞争同一节点的内存系统。单卡数据不能直接乘以卡数得到可靠的整机吞吐预测。

7.3 GPUDirect RDMA 改变的是机内这段路

GPUDirect RDMA 允许兼容网卡通过 PCIe 对 GPU 内存进行直接访问。在适合的拓扑和配置下，payload 可以在网卡与 GPU 之间走设备到设备的路径，避免先暂存到主机 DRAM。它不是一种新的跨机网络协议，也不等于 NVLink。NVIDIA GPUDirect RDMA 概览

拓扑工具中的 PXB 表示两个设备之间经过多个 PCIe bridge，而不经过 PCIe Host Bridge。这描述的是拓扑关系；它不证明某次程序已经用上了这条路径，更不证明实际带宽。NVIDIA 拓扑标记

要建立更强结论，至少需区分三个问题：GPU 缓冲是否真正注册为 RDMA 目标；实际搬运是否避开主机暂存；在正确同步下，大尺寸和多次传输的有效吞吐是多少。例如，小缓冲写入后读回一致，只能支持该测试条件下的正确性，不能支持“HBM 路径跑满 800G”。

另一个独立问题是 GPU 可见性。一个正在运行的 GPU kernel，不会仅因远端完成了 RDMA WRITE，就自动获得安全的新数据视图。必须使用适合平台与 API 的 CUDA 工作提交、同步或可见性机制。NVIDIA 明确提醒并发 kernel 可能观察到旧数据、部分数据或乱序数据。GPUDirect RDMA 同步与内存顺序

8 用四个量读懂性能

8.1 容量 吞吐 延迟 并发

- 链路容量：这一段理论上每秒能通过多少数据
- 有效吞吐：单位时间真正完成了多少应用 payload，不含你不计入成果的包头与控制流量
- 请求延迟：一项操作从开始到完成用了多久；起止点必须明确
- 在途并发：有多少项操作已经开始但还没完成

800 Gb/s = 100 GB/s，因为 1 byte = 8 bit。GB 常按十进制；MiB 按二进制。8 MiB = 8,388,608 字节，在 400 Gb/s 的有效传输速率下，仅 S/B 一项约为 0.168 ms。这不是完整 put 的预测延迟。

8.2 路径上限与端到端耗时

对持续、充分流水化的大传输，可用下面的式子做第一层检查：

稳定有效吞吐 ≤ min(源内存读取、源 PCIe、网络、目标 PCIe、目标内存写入)

它只是上界模型，不是保证能达到的值；实际还有共享资源、包开销、读请求窗口和软件供给等限制。

不同段可以同时处理不同的数据块，因此不要给整个对象在每一段都算一个 S/B，再机械相加来预测稳态吞吐。反过来，小请求和单次操作仍有启动、传播、排队以及控制处理的固定成本。

一个便利 API 的端到端耗时可以概念性地分成：

准备缓冲 + 注册/复制 + 准入 RPC + 排队 + 数据传输
+ 完成处理 + 发布对象/解除读保护

哪些项能重叠、哪些在关键路径上，只有分阶段计时才能回答。单路慢、多路快说明并发能隐藏某些等待，不能单凭这一现象断言“就是 RPC 慢”。

8.3 并发让管道有事可做 不会把管道变粗

若每次都等上一个请求全部结束才发下一个，网卡可能时而忙、时而空。多个在途请求让等待与搬运重叠，吞吐因此提高。

一个量级估算是：要把带宽 B 的路径维持忙碌，在途数据量需要覆盖 B × L。这里 L 是该层限制窗口周转的时间，可能接近网络往返时延，也可能包含设备或软件等待，不能随手拿任意 RTT 代入。

假设 B = 50 GB/s、L = 10 微秒，则对应约 500 kB 在途数据。这只是量级示例，不是机器测量值。单个 QP 如果有足够深度，也可能提供这些在途工作；增加 QP 只是扩大并行度的一种手段。

并发已经足够后再继续增加，可能只让队列更长、延迟更大。判断调优是否有效，要同时看吞吐、平均延迟和尾延迟，例如 p99，而不是只看最高 Gb/s。

8.4 吞吐的倒数通常是完成间隔

假设对象大小为 8 MiB，总吞吐为 48 Gb/s：

```
每个对象的平均完成间隔
= 8 MiB × 8 ÷ 48 Gb/s
≈ 1.398 ms
```

如果稳定时期平均有 8 个这样的对象在途，并且吞吐和在途数按同一请求边界统计，则由 Little 定律，平均请求延迟约为 8 × 1.398 = 11.18 ms。不能因为 1.398 ms 的间隔就声称每个请求只花 1.398 ms。

“配置了 8 个 worker”也不自动等于“平均正好 8 个请求在途”。准确延迟仍应记录每个请求的开始与结束；分位数更不能从平均吞吐反推。

9 给常见性能说法补上证据边界

将机制用于判断性能时，需要逐项检查以下边界。

800G 网卡不保证 **800G DRAM** 吞吐。正确。应继续追问是哪张卡、哪块内存、哪个方向、哪些共享资源，以及测试是不是充分流水化。

QP 多了可能更快。正确的可能性，不是定律。需要区分应用并发、QP 数量、每 QP 深度，以及 READ 的未完成请求能力；不要把八路请求直接理解成八条独立链路。

网卡通过 **CQ** 报告完成。这是入门概括。应补充不是每个发送请求都有单独成功 CQE，post 成功也不是完成；完成之后能否发布给 CPU/GPU 消费者还取决于操作与同步语义。

单向应用流量可能占用 **PCIe** 两个方向。正确。同机循环搬运的 payload 先从内存读到 NIC，再由 NIC 写回内存；它会使用两个方向。两个方向若能充分重叠，不必然使吞吐减半；也不能把同一份 payload 的两段搬运加起来，冒充两倍应用有效吞吐。是否真的经过外部线缆，应由端口或交换机计数器验证。

零拷贝后应用吞吐提高。这与软件开销或内存流量得到改善相容，但需要分阶段计时来确认原因。不同大小、方向、并发或缓存状态的结果不能无条件横比，更不能把上限差额全算作某一个软件阶段。

HBM 正确性通过且拓扑为 **PXB**。这是进一步测量的基础，不是 800G 性能验证。正确性、实际路径和持续吞吐需要分别建立证据。

10 再读 KV 传输代码时带着这六个问题

不用先把网络课本全部学完。阅读 SGLang、Mooncake 或其他 KV 传输实现时，可以先在代码里找这条链：

1. 数据在哪里：CPU DRAM、GPU HBM，还是中间暂存池？谁拥有这块缓冲？
2. 操作是什么：应用 put/get 最终调用 WRITE、READ 还是 SEND？谁推、谁拉？
3. 权限怎么来：谁注册 MR，谁交换地址和 rkey，注册与缓冲能活多久？
4. 并发在哪里：应用请求数、QP 数、队列深度分别由哪里控制？
5. 何时可消费：谁检查 CQ，谁发布 ready，GPU 通过什么同步条件开始读？
6. 性能在测什么：只测 payload 搬运，还是包含分配、RPC、发布和释放？

这里没有断言这些项目的某个版本一定采用何种实现；它是一份阅读代码时的检查顺序。

若以后在机器上验证，先做只读拓扑检查，再按顺序比较：同条件的主机内存裸传输、GPU 内存传输、应用完整 put/get；逐项变化消息大小、并发和 NUMA 位置，避免一次改动太多参数。应同时记录实际字节数、总时间、每请求延迟、CPU 占用，以及必要的端口/拥塞计数器。

最后用四个问题自测：

- A 对 B 发起 READ，大块数据主要向哪边走？答案是 B → A；请求方向相反
- Gen5 x16 双向合计约 126 GB/s，能让一次 A → B 传输达到 126 GB/s 吗？不能，空闲的反方向不能借来用
- 普通 WRITE 在本地完成后，B 的应用会自动收到一次接收 CQE 吗？通常不会，需要约定通知与发布
- 同时有 8 个对象在途，平均每 1 ms 完成一个，能说请求延迟是 1 ms 吗？不能；稳定且口径一致时平均约 8 ms

理解到这里，再回看一篇 RDMA 性能分析，最重要的变化应当是：你不再只盯着“800G”“Gen6”“零拷贝”几个标签，而能说清楚哪个字节从哪里出发，经过哪段链路，由谁发起，何时才允许下一个消费者使用。

术语速查

缩写	完整名称或含义	本文中的角色
NIC / RNIC	Network Interface Card / RDMA-capable NIC	网卡 / 支持 RDMA 的网卡
DMA / RDMA	Direct Memory Access / Remote Direct Memory Access	设备直接访问本地内存 / 远程内存操作
PCIe	Peripheral Component Interconnect Express	主机内部高速互连
DRAM / HBM	Dynamic RAM / High Bandwidth Memory	主机内存 / GPU 高带宽显存
MR / PD	Memory Region / Protection Domain	注册内存区域 / 保护域
lkey / rkey	local key / remote key	本地 / 远端访问保护键
QP / SQ / RQ	Queue Pair / Send Queue / Receive Queue	队列对 / 发送 / 接收工作队列
WR / WQE	Work Request / Work Queue Element	应用工作请求 / 设备队列工作项
CQ / CQE	Completion Queue / Completion Queue Entry	完成队列 / 一条完成记录
RC	Reliable Connection	本文数据路径示例采用的可靠连接服务
RPC	Remote Procedure Call	跨进程或跨机请求操作，常承担控制流程
NUMA	Non-Uniform Memory Access	不同内存位置的访问代价不一致
PFC / ECN / CNP	Priority Flow Control / Explicit Congestion Notification / Congestion Notification Packet	暂停机制 / 拥塞标记 / 拥塞反馈包

来源与核验边界

概念与 API 行为依据下列官方文档和标准整理，核验日期为 10月 8, 2026。本文由 AI 辅助整理并进行技术审阅；拓扑、对象大小与性能算例均明确作为教学假设，不代表作者完成的机器检查、硬件实验或独立复现。

1. TCP 标准
2. UDP 使用指南
3. Linux DMA 地址说明
4. NVIDIA RDMA 操作概览
5. Intel I/O 事务与数据方向说明
6. rdma-core 内存注册接口
7. 创建 QP
8. 提交 WR
9. 轮询 CQ
10. 投递接收资源
11. NVIDIA RoCE 说明
12. IETF RDMAP 标准
13. NVIDIA RoCE 网络控制
14. PCI-SIG 代际速率与编码资料
15. NVIDIA GPUDirect RDMA 概览
16. NVIDIA 拓扑标记
17. GPUDirect RDMA 同步与内存顺序