

# AI 集群组网与网络拓扑：从交换机基础到双平面 ZCube

面向工程/研究人员的中文技术教程。读者不需要任何网络工程背景，我们将从"交换机里有什么"这类最基础的概念讲起，一步步深入到 Clos/fat-tree 的规模计算、成本与故障域分析、大模型流量特征，最后完整解剖智谱/驭训网络（Harnets.AI）的双平面 ZCube 方案，并对"省 1/3 成本、减 1/3 故障域"的说法做一次来源分级的批判性评估。

## 0. 引言：为什么要写这篇文档

### 0.1 缘起：一个听起来好得不像话的传闻

假设你是一位工程或研究人员，正在参与一个大模型推理集群的组网选型。某天你在技术社区刷到这样一条消息：

"智谱和驭训（Harnets.AI）搞了个叫 ZCube 的双平面组网，相比 Clos 节省 1/3 组网成本、减少 1/3 网络故障域，已经在 GLM-5.1 coding 的千卡生产集群落地了。"

下面还有一条高赞评论：

"这个拓扑特别适合大模型 EP（专家并行）。"

省三分之一的钱，还同时减少三分之一的故障域，性能还更好——这听起来违反"没有免费午餐"的工程直觉。你的第一反应应该是怀疑；第二反应应该是：要验证或驳斥这个说法，我需要懂哪些东西？

本文档就是为回答这个问题而写的。我们会从零开始，把"交换机""光模块""Clos""rail""故障域""EP all-to-all"这些概念逐一讲透，手把手带你做成本计算和故障域判断，最后回到 ZCube 本身，拆开它的数学结构，验证它的成本论证，标注哪些说法有原始论文支撑、哪些只是媒体演绎。

### 0.2 与前一篇文档的关系

本文是同系列第二篇。前一篇《NVIDIA GPU 微架构与 NVL72 AI 工厂》（stor-2.md）已经讲过以下概念，本文会直接引用、不再展开：

- **ECN / PFC / RoCEv2 / 无损网络**：以太网如何做到不丢包，以及 PFC 反压为什么会"传染"；
- **Scale-up vs Scale-out**：NVLink 域内纵向扩展 vs 跨节点横向扩展；
- **南北向 / 东西向流量**：进出数据中心的流量 vs 数据中心内部 GPU 之间的流量；
- **GB300 NVL72、ConnectX-7 / ConnectX-8 网卡、以及 双平面（dual-plane）** 的基本概念（每 GPU 的两个网口分接两套独立交换网络）。

如果你对这些概念还不熟，建议先读前一篇的对应章节；不读也能跟上本文，因为关键处我们都会做一句话级别的回顾。

### 0.3 本文的主要参考来源与"来源层级"

本文事实性内容分三个来源层级，正文和附录中会明确标注：

- **L1 一手原始资料**：ZCube 官方博客（z.ai/blog/zcube 及其中文版 zhipuai.cn/zh/research/160）、配套分析论文 arXiv 2607.21893（Fewer Paths, Better Performance: Understanding the ZCube Topology through Braess's Paradox）、原始论文 Yan et al. From ATOP to ZCube（ACM

SIGCOMM 2025)、Al-Fares et al. SIGCOMM 2008、Meta RSC 论文 (arXiv 2410.21680)、Google Jupiter 论文 (SIGCOMM'22)、阿里 HPN 论文 (SIGCOMM 2024) 等。这些是我们最信任的。

- **L2 行业经验口径/工程解读**: 光模块与交换机价格锚点、"光模块约占网络成本一半以上"这类行业经验比例; 以及公众号「驭驯网络」转载的 ZCube 系列技术文章 (作者: 疯狂的兔子tommy, 原文出自「算力网络架构手记」, 共三篇, URL 见附录 C) ——属于业内人士基于公开资料与工程经验的解读, 本文按 L2 引用并逐条标注。这些用于估算与工程直觉, 性质是"经验锚点", 不是精确报价或官方承诺。
- **L3 媒体转载口径**: 如"ZCube 不可达概率比双平面 Clos 低 50% 以上""102.4T 交换机支持 65,536 GPU"等出自腾讯新闻/冷月清谈等转载稿的说法。这些在原文中找不到直接出处, 本文一律标注"媒体口径, 未见于原文"。

读完本文你应该养成一个习惯: **任何惊艳的网络架构宣传, 先问"这句话出自论文第几节"**。

## 0.4 阅读路线图

全文按"由浅入深"组织为六部分加附录:

- **第一部分 交换机与组网基础**: 交换机内部是什么、端口与 radix、速率演进、光模块与铜缆、rail/plane/单轨/多轨、GPU:NIC 绑定、无阻塞与收敛比。网络小白从这里开始。
- **第二部分 经典网络拓扑课**: Clos 1952 的原始思想、fat-tree、两层/三层 spine-leaf 的规模公式推导 (手把手算)、rail-optimized fat-tree、Dragonfly/Dragonfly+、3D torus、hypercube, 以及一张横向对比表。
- **第三部分 成本计算与故障域入门**: 以 1024 卡和 16384 卡为例, 手把手算两层/三层 Clos 的交换机数、光模块数、总成本量级; 然后讲故障域的定义、各类故障的影响面、MTTF 模型与 Meta/Google 的实测统计。
- **第四部分 大模型流量特征**: DP/TP/PP/EP/SP 各产生什么通信, MoE 的 all-to-all 为什么难搞, PD 分离与 KV cache 传输, 以及为什么 Clos+ECMP 在这些流量下会难受 (Braess 悖论)。
- **第五部分 ZCube 深入解剖**: 架构总览、单轨+多轨双端口的数学 (用 32 卡小例子算给你看)、端口预算公式  $k + n/k \leq$  端口数与交换机总数  $2n/k$ 、直径 2 跳、成本 1/3 的精确来源 (自己算一遍验证 33%)、负载扩散定理、故障域分析、生产实测数字、ZCube Controller 与运维复杂度的转移。
- **第六部分 批判性评估**: ZCube vs Clos vs Dragonfly+ vs 阿里 HPN vs NVIDIA 双平面的取舍; ZCube 的局限与开放问题; "省 1/3"说法的来源分级表; 场景化选型决策建议。
- **附录**: 术语表 (55+ 条中英对照)、FAQ 汇总 (30+ 问)、参考链接。

每一部分末尾都有"小结"和"问答 FAQ"两个模块, FAQ 汇总在附录 B 中再完整列一次。

## 第一部分 交换机与组网基础: 网络小白起步

### 1.1 交换机到底是什么

拆开一台数据中心交换机, 你会看到三样决定性的东西:

**1. ASIC 交换芯片**。交换机不是"小型电脑", 它的核心是一颗专用交换芯片 (switch ASIC) ——比如 Broadcom 的 Tomahawk / Jericho 系列、NVIDIA 的 Spectrum 系列、Cisco 的 Silicon One。所有端口进出的数据包都在这一颗芯片内完成"查表转发": 读包头、查转发表、从正确的出端口发出去, 全程硬件流水线, 时延只有几百纳秒级。这就是为什么交换机转发比任何软件路由都快。

**2. 端口 (port) 与 radix。** 一台交换机能接多少根线，由它的端口数决定，这个数有个专门的名字叫 **radix (基数)**。端口数乘以端口速率就是这台交换机的**交换容量**。当前 AI 集群的主流旗舰是 **128 个 400G 端口、总容量 51.2Tbps** 的交换机 ( $128 \times 400G = 51.2T$ )。记住这个型号，它会在后文反复出现：128 口、51.2T。

**3. Buffer (缓存)。** 每个端口背后有共享缓存，用来吸收突发流量：多个发送端同时把包发给同一个接收端时，多出来的包先排队。buffer 打满之后会发生什么，取决于网络是"有损"还是"无损"：有损以太网直接丢包（靠上层重传）；无损 RoCE 网络会触发 **PFC (Priority Flow Control) 反压**，逐级"喊停"上游发送端。PFC 的细节在前一篇文档讲过，这里只需要记住一个直觉：**PFC 把局部的拥塞变成了向上游传导的压力波**，一个地方堵住，压力会顺着链路一路传回发送端，殃及同优先级的其他无辜流量。

还有一个隐藏角色：**ECMP (Equal-Cost Multi-Path, 等价多路径)**。当两台交换机之间存在多条成本相同的路径时，交换机按数据包的五元组（源 IP、目的 IP、源端口、目的端口、协议）做哈希，把每条流"钉死"在某一条路径上。注意两个要点：哈希是**静态的**（流一旦上路就不换道）；哈希是**拥塞盲的**（选路时不看哪条路径正在堵车）。这两点会在第四、五部分变成主角。

## 1.2 速率演进：从 100G 到 1.6T

以太网单端口速率在过去十年大致每两三年翻一倍：

| 代际   | 电气实现               | 大致时间                 |
|------|--------------------|----------------------|
| 100G | 4 × 25G NRZ        | ~2016                |
| 200G | 4 × 50G PAM4       | ~2019                |
| 400G | 8 × 50G / 4 × 100G | ~2021                |
| 800G | 8 × 100G           | ~2024                |
| 1.6T | 8 × 200G           | 2026 年进入部署           |
| 3.2T | 配合 CPO 交换机         | 2026+ (NVIDIA 已宣布计划) |

每一代的套路都一样：单 lane（通道）速率翻倍，或者用更多 lane 并行。形态上，可插拔光模块从 QSFP-DD 演进到散热更好的 **OSFP**——800G/1.6T 的 AI 集群基本都用 OSFP。一个市场规模锚点：光模块市场 2026 年约 154 亿美元，预计 2035 年到 481 亿（CAGR 13.5%，Gminsights 数据，L2 经验口径）。AI 集群是这场增长的最大买家。

## 1.3 光模块：组网成本里最容易被低估的部分

光模块（optical transceiver）是插在每个光纤端口两端、负责电-光转换的小盒子。它们按传输距离分档：

- **SR (Short Reach)**：多模光纤，~100 米，机房内短距；
- **DR (Data center Reach)**：单模，500 米，数据中心主力；
- **FR**：单模，2 公里；
- **LR**：单模，10 公里。

命名后缀的数字是 lane 数：800G DR8 = 8 条 100G lane。

技术上还有三个趋势值得知道：

- **TRX (传统模块)**：内含 DSP（数字信号处理器）做信号再生，成熟通用但耗电；
- **LPO (线性直驱)**：去掉模块内 DSP，靠交换机芯片直接驱动，省电约 30%、时延更低，适合 AI 集群短距场景（Meta、Amazon 在 1.6T 上采用），代价是互通性受限；

- **CPO（共封装光学）**：把光引擎和交换 ASIC 封在同一个封装里，功耗最低、密度最高，但不可热插拔、坏了维修难，预计 2027 年后规模商用。

功耗量级（L2 经验口径）：800G EML 模块 18–24W；1.6T 硅光 13–15W；1.6T LPO 10–12W；CPO 8–10W。一个万卡集群有几万只光模块，每只十几瓦，加起来就是几百千瓦——所以光模块不只是采购成本，还是功耗大头。

## 1.4 铜缆：DAC / ACC / AEC 的距离-成本阶梯

不是所有连接都要用光。距离够短时，铜缆便宜得多：

| 类型       | 原理               | 800G 典型距离           | 功耗/端  | 成本  | 典型场景        |
|----------|------------------|---------------------|-------|-----|-------------|
| DAC 无源铜缆 | 纯双绞线，无芯片         | ~3m (1.6T 仅 1–1.5m) | ~0W   | 最低  | 机柜内 GPU↔ToR |
| ACC 有源铜缆 | 端头 redriver 放大均衡 | ~4–5m               | 1–2W  | 低-中 | 机柜内/相邻柜     |
| AEC 有源电缆 | 端头 retimer 全信号再生 | ~5–7m               | 4–8W  | 中   | 服务器到交换机跨排   |
| AOC 有源光缆 | 电-光-电转换          | 30m+                | 8–12W | 高   | 排级及以上       |

经验法则：**3 米内 DAC 无敌，3–7 米用 ACC/AEC，再远必须上光纤**。注意速率越高铜缆窗口越窄——1.6T 时代 DAC 只剩 1–1.5 米。这就是为什么下一代集群的布线规划越来越“富光纤”，也是为什么任何能省掉一整层交换机（连同其长距离光纤）的拓扑都自带巨大的成本诱惑——记住这句话，第五部分会回收。

## 1.5 Rail、Plane、单轨、多轨

这是理解后面所有拓扑的四个词，务必分清：

- **Rail（轨道）**：一台 8 卡服务器有 8 张网卡（每 GPU 一张）。把集群里**所有服务器**的“**第 i 号网卡**”**接到同一组交换机**，就形成了第 i 条“轨道”。8 卡机 = 8 条 rail。同一条 rail 内的任意两张 GPU 只隔一台交换机（一跳）。集合通信库（NCCL）会尽量把流量约束在 rail 内。Rail 是一种**按网卡序号切分流量的逻辑分组**。
- **Plane（平面）**：整个交换网络的一套**独立物理冗余拷贝**。双平面 = 两套完整且互相独立的交换网络，每张网卡（或其两个端口）各接一套，流量一半走平面 A、一半走平面 B；一个平面整体故障只损失一半带宽。Plane 是**物理隔离**的概念，粒度比 rail 粗。
- **单轨（single-rail）接入**：相邻编号的一组 GPU 通过同一台交换机接入网络——“连续块”分组。同一台 8 卡机的 8 张 GPU 如果都走单轨接入同一台交换机，它们之间通信就是一跳。
- **多轨（multi-rail）接入**：每台 GPU 按“模位置”分散接入不同的交换机——类似 ROFT 的 rail 思想，同一位置的网卡进同一组。

ZCube 的天才之处在于：每张双端口网卡，一个端口用**多轨方式接入一组交换机**，另一个端口用**单轨方式接入另一组**——两种正交的分组方式叠加在同一张物理网上。这句话现在看不懂没关系，第五部分我们会用具体数字算给你看。

顺带回顾前一篇文档的一对概念（后面会反复用到）：**Scale-Up**（服务器/机柜内部：GPU↔GPU 走 NVLink/NVSwitch、GPU↔NIC 走 PCIe）与 **Scale-Out**（服务器之间：NIC↔交换机的 RoCE/IB/以太网）。本文讨论的 Clos、ROFT、ZCube 全部属于 Scale-Out 这一层——ZCube 的端点是每张 GPU 的双端口 NIC，它不替代 NVLink/NVSwitch 的域内互联，两层各司其职。



# 没有 Spine，不等于没有 NVSwitch

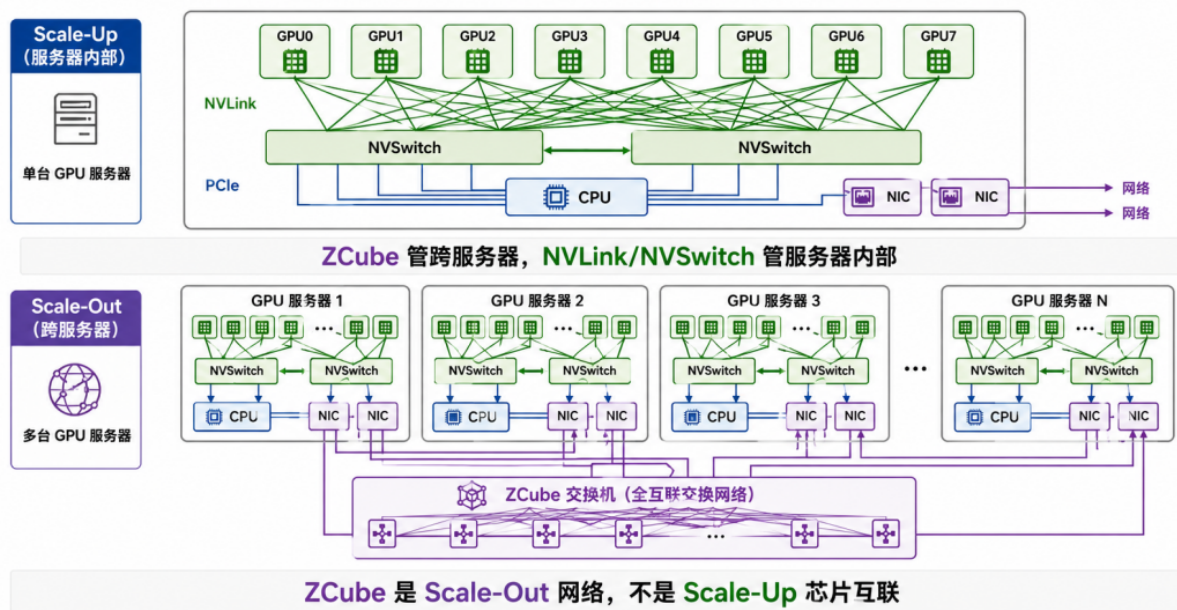


图 1-1（图源：微信公众号「驭驭网络」ZCube 系列文章三，L2）：Scale-Up（NVLink/NVSwitch 域内高速互联）与 Scale-Out（跨服务器网络）的分层——ZCube 属于后者。

## 1.6 GPU 与网卡的绑定模式

主流 AI 集群的后端计算网采用 **GPU:NIC = 1:1**：每张 GPU 配一张独立网卡（H100 配 ConnectX-7 400G，B300/GB300 配 ConnectX-8 800G），8 卡服务器就有 8 张网卡。好处是每张 GPU 有独立的网络带宽出口，配合 GPUDirect RDMA，网卡可以直接读写 GPU 显存、不绕 CPU。

特例是 NVL72：18 个 compute tray、每 tray 4 张单口 CX-8，每 tray 3.2T 上行；CX-8 集成的 PCIe switch 让每张 GPU 都有直达网卡的 RDMA 路径（GB200/CX-7 时代还要绕 Grace CPU 的 NoC——这是前一篇文档讲过的细节）。

也有些省钱场景用 2:1（两张 GPU 共享一张网卡），多见于推理或前端网。另外，前端网、存储网、带外管理网通常与后端 GPU 计算网**物理分离**，不要混为一谈——本文只讨论后端 GPU 计算网（东西向流量承载网）。

## 1.7 无阻塞与收敛比（oversubscription）

最后一个基础概念。

**无阻塞（non-blocking）**：网络中任意一对端点可以同时满速率通信，互不干扰。工程等价条件：每台交换机的下行端口带宽 = 上行端口带宽（1:1），整网也满足“全对分带宽”（full bisection bandwidth）。

**收敛比/超订（oversubscription）**：如果下行:上行 = 3:1，意味着上行带宽只有下行的三分之一，当超过三分之一的 GPU 同时对外通信时就要排队降速。省钱，但性能让位于概率。

经验：大模型训练的后端网几乎一律要求 **1:1 无阻塞**（all-reduce 是全员同时通信，没有“平均负载”可言）；前端网、存储网常做 3:1 甚至更高的超订。本文后面所有成本计算，默认无阻塞口径。

## 1.8 小结

- 交换机 = ASIC + 端口 + buffer；**radix（端口数）× 端口速率 = 交换容量**，当前旗舰  $128 \times 400G = 51.2T$ 。

- 光模块按距离分 SR/DR/FR/LR，按技术分 TRX/LPO/CPO；铜缆 DAC/ACC/AEC 在 3-7 米窗口内比光纤便宜得多，但窗口随速率收窄。
- **Rail** 是按网卡序号的逻辑分组，**plane** 是物理独立的整网冗余；**单轨**是连续块分组接入，**多轨**是模位置分散接入。
- 后端 GPU 网主流 GPU:NIC=1:1；训练网要求 1:1 无阻塞。
- ECMP 按五元组哈希静态选路，是"拥塞盲"的——这个缺陷是后面整个故事的引线。

## 1.9 问答 FAQ

### Q1：交换机和家用路由器有什么区别？

家用路由器是软件/低端芯片转发 + 少量端口；数据中心交换机是一颗专用 ASIC 硬件流水线转发几十上百个高速端口，时延几百纳秒，不支持复杂的 NAT/防火墙功能，只做一件事：快。

### Q2：51.2T 交换机的"51.2T"指什么？

所有端口速率之和：128 个 400G 端口  $\times$  400Gbps = 51.2Tbps 双向交换容量。

### Q3：光模块为什么那么贵？

里面要塞进激光器（EML 或硅光）、驱动、DSP（传统模块）、精密光学对准结构，还要过严苛的良率测试。800G 时代一只几百美元，量级锚点见第三部分。

### Q4：DAC 和 AEC 不都是"铜线"吗，差在哪？

DAC 是纯线缆没有芯片，超过 3 米信号就劣化得没法用；AEC 两端有 retimer 芯片把信号完整再生一遍，能到 5-7 米，但更贵、更耗电。

### Q5：rail 和 plane 到底怎么区分？

一句话：rail 是逻辑上"按网卡序号切蛋糕"（同一物理网内分组），plane 是物理上"烤两盘蛋糕"（两套独立网络）。实践中常组合：每个 plane 内部再做 rail 优化。

### Q6：为什么训练网不能用超订省钱？

因为训练的主力通信 all-reduce 是"全员同时开口说话"，不存在统计复用。超订省的钱会以每一步迭代都变慢的形式持续偿还，长期看极不划算。

## 第二部分 经典网络拓扑课：从 Clos 到 Fat-Tree 到 Dragonfly

### 2.1 Clos 1952：一个电话交换时代的省钱发明

1952 年，贝尔实验室的 Charles Clos 面对的问题是：电话交换机要连接  $N$  条线路，如果用交叉矩阵（crossbar），需要  $N \times N$  个交叉点——成本随用户数的平方爆炸。他的方案发表在 A Study of Non-Blocking Switching Networks（Bell System Technical Journal, 1953）里：**用三级交换级联（输入级 ingress  $\rightarrow$  中间级 middle  $\rightarrow$  输出级 egress）替代大矩阵**，交叉点数量大幅下降，而且能保证"任意空闲输入都能连到任意空闲输出"。

Clos 网络用三个参数描述  $(n, m, r)$ ： $r$  个输入级交换器、每个  $n$  入  $m$  出、 $m$  个中间级交换器。两个经典结论：

- **严格无阻塞条件： $m \geq 2n - 1$** 。直觉：最坏情况下，你想接入的输入所在的输入级有  $n-1$  个忙线占住中间级，你想接出的输出级也有  $n-1$  个忙线占住中间级，只要中间级再多出 1 个（共  $2n-1$ ），就永远能找到一条空闲通路。
- **可重排无阻塞条件： $m \geq n$** （允许重排已有连接的话可以放宽）。

ASCII 示意一个三级 Clos：

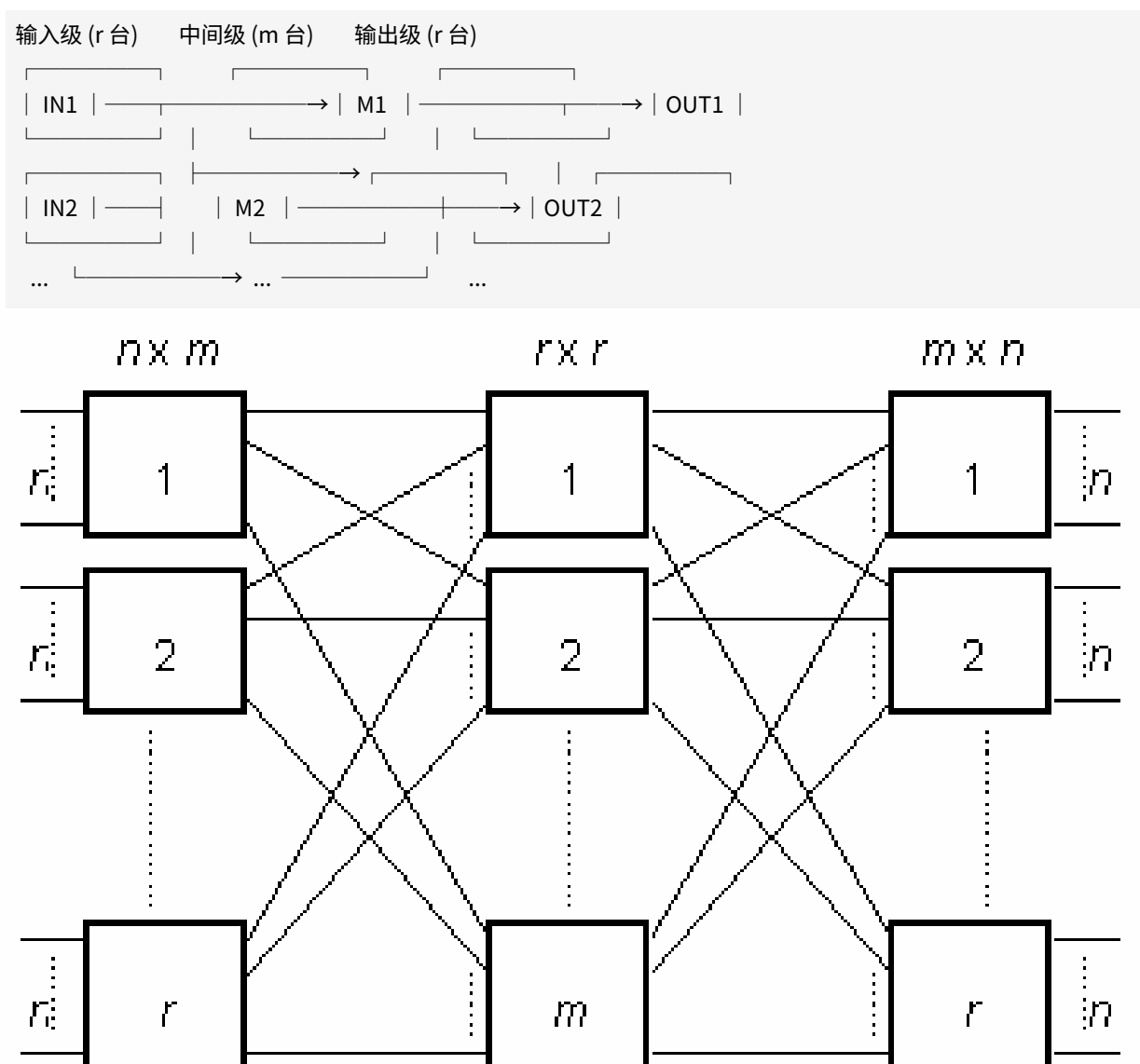


图 2-1（图源：Wikimedia Commons, Three-stage clos network；原图分辨率偏低但结构清晰）：三级 Clos 的参数化结构—— $r$  个  $n \times m$  输入级、 $m$  个  $r \times r$  中间级、 $r$  个  $m \times n$  输出级。

关键思想穿越了 70 年：**用多级小交换机的级联，代替一台大得造不起的交换机**。今天数据中心的一切 spine-leaf 架构都是这个思想的后代。

## 2.2 Fat-tree：把 Clos 种成一棵树

1985 年 Charles Leiserson 提出 fat-tree：一棵树，越靠近根链路越粗（“胖”在上头），保证任何子树之间都有充足带宽——它本质上是 Clos 的一个特例。

2008 年，Al-Fares、Loukissas、Vahdat 在 SIGCOMM 发表 A Scalable, Commodity Data Center Network Architecture，把 fat-tree 思想引入以太网数据中心：不用昂贵的大机框交换机，用一堆商用小交换机堆出无阻塞网络。k 元 fat-tree 的结构：k 个 pod，每个 pod 两层各  $k/2$  台交换机，核心层  $(k/2)^2$  台；可接  $k^3/4$  台主机（ $k=48$  时是 27,648 台）。

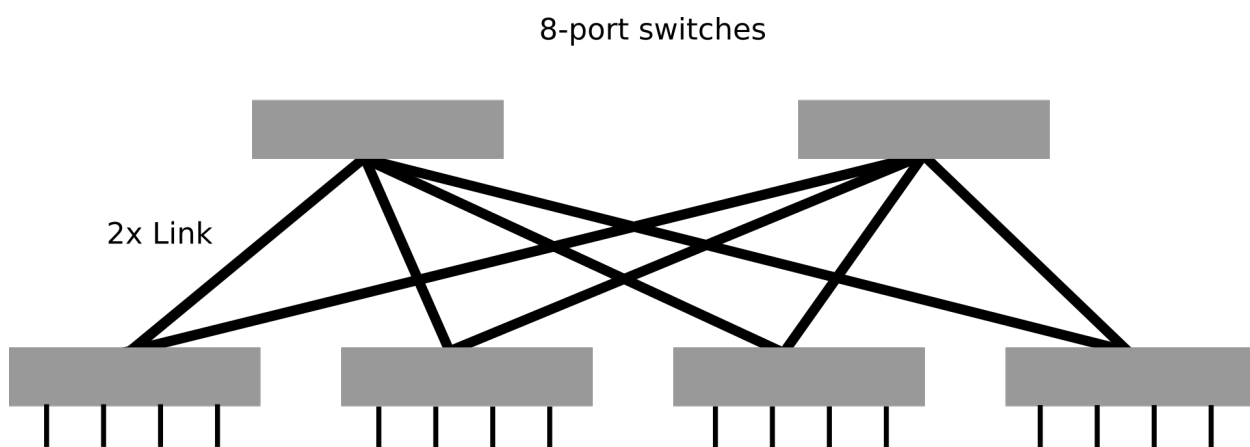


图 2-2（图源：Wikimedia Commons, Fat-tree.svg, CC 授权）：通用 fat-tree——越靠近根，聚合链路带宽越“胖”（图中上行 2 倍加粗）。

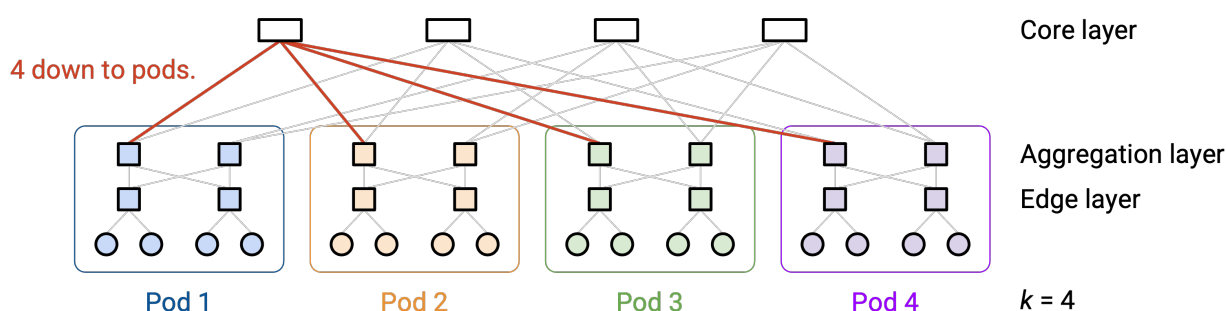


图 2-3（图源：Berkeley CS168 教材, Datacenter Topologies 一节）：k 元 fat-tree 的 pod 划分——每个 pod 内部两层，pod 之上再汇合到核心层；pod 是扩容与布线的基本单元。

工程上还有一个“折叠”的视角：经典三级 Clos 是左右对称的，沿中线对折，输出级叠回输入级，就得到 **folded Clos = leaf-spine**：

- **Leaf**（叶子）：最下层，直接接服务器/GPU（ $\approx$  ToR, Top-of-Rack）；
- **Spine**（脊）：中间层，只转发，不接端点；每台 Leaf 到每台 Spine 都有一条链路。

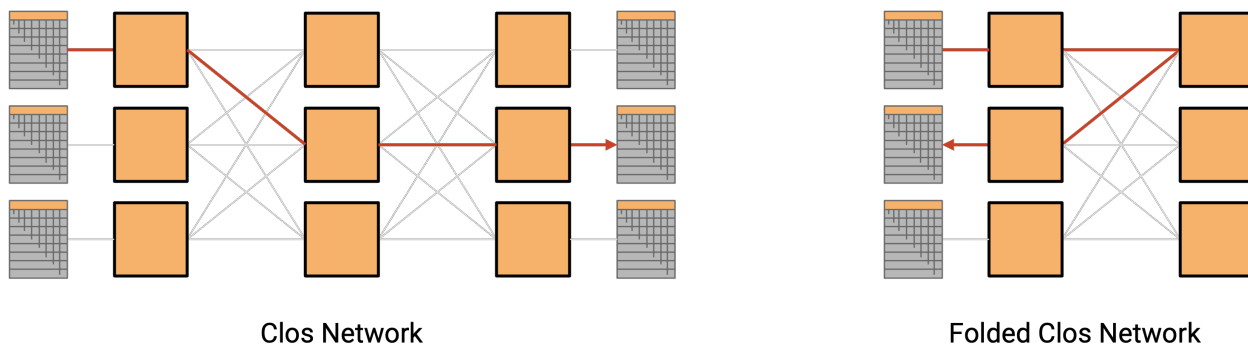


图 2-4（图源：Berkeley CS168 教材）：三级 Clos（左）沿中线折叠为 folded Clos（右）——输出级与输入级合并为双向端口，这就是 leaf-spine 的物理形态。

三个词辨析（面试常考）：**folded Clos** 指物理折叠形态；**leaf-spine** 是工程实现的俗称；**fat-tree** 强调“带宽向上汇聚、全对分带宽”的设计目标。在日常语境里三者基本是同一个东西。

## 2.3 两层 vs 三层：规模公式手把手推导

这是本节最重要的计算。设交换机 radix = R 个端口，要求无阻塞（一半端口向下接端点、一半向上）。

**两层（Leaf + Spine）：**

- 每台 Leaf:  $R/2$  口向下接 GPU,  $R/2$  口向上接 Spine;
- Spine 数量 =  $R/2$  (因为每台 Leaf 只有  $R/2$  个上行口, 每个口接一台不同的 Spine, 而每台 Spine 要接所有 Leaf) ;
- Leaf 数量  $\leq R$  (每台 Spine 有  $R$  个端口, 每个端口接一台 Leaf) ;
- 最大端点数 = Leaf 数  $\times$  每 Leaf 下行口数 =  $R \times R/2 = R^2/2$ 。

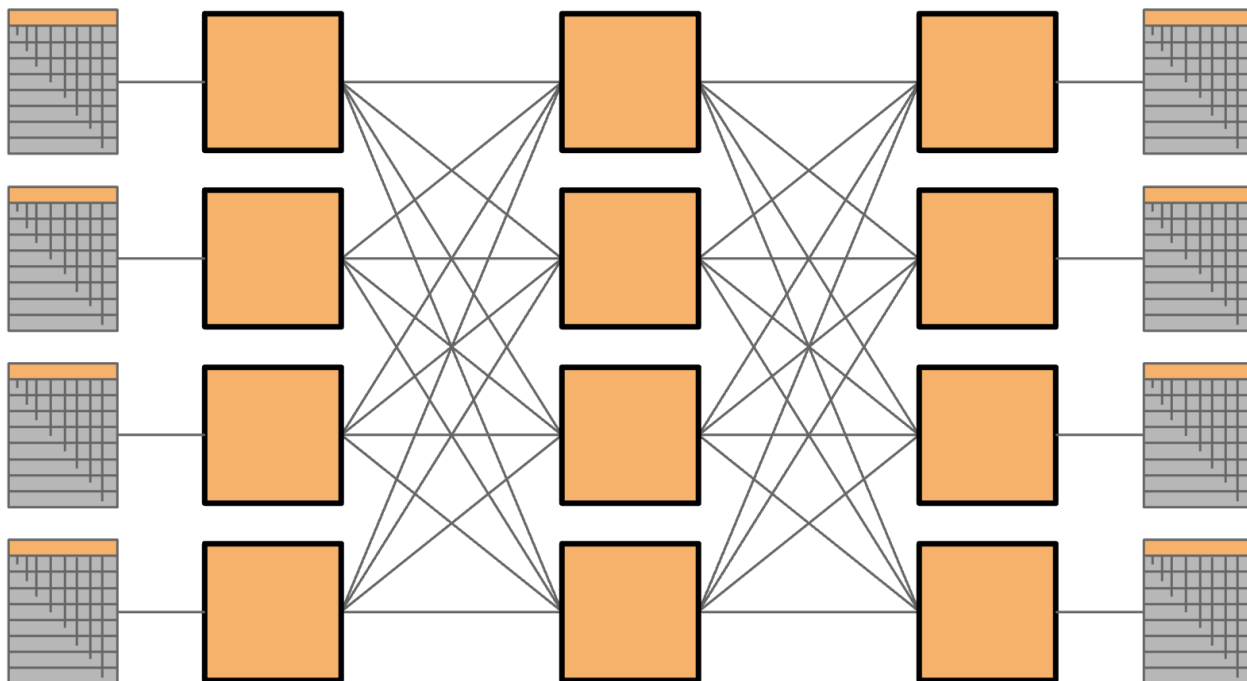


图 2-5 (图源: Berkeley CS168 教材) : 两层 leaf-spine 的展开形态——服务器接入 Leaf, 每台 Leaf 全互联到所有 Spine。

代数字:

- $R = 64$  (12.8T/25.6T 代际) :  $R^2/2 = 2048$  个端点;
- $R = 128$  (51.2T,  $128 \times 400G$ ) :  $R^2/2 = 8192$  个端点。

### 三层 (Leaf + Spine + Super-Spine, 即 5 级 Clos) :

把"Leaf+Spine"打包成一个 pod (一个 pod 就是一个两层 Clos, 规模  $R^2/2 \dots$  的一部分), 再用一层 Super-Spine 把 pod 互联。通用近似公式: **L 层 Clos 的无阻塞端点数  $\sim R \cdot (R/2)^{(L-1)}$ 。**

- $R = 64$  三层:  $64 \times 32^2 = 65,536$  端点;
- $R = 128$  三层:  $128 \times 64^2 = 524,288$  端点。

所以你会在公开资料里看到"两层万卡以内、超万卡上三层"的经验说法: 用 51.2T ( $R=128$ ) 交换机做两层, 单端口 400G 的 GPU 最多接 8192 张——万卡门槛刚好卡在两层和三层的分界线上。这不是巧合, 是  $R^2/2$  的算术结果。



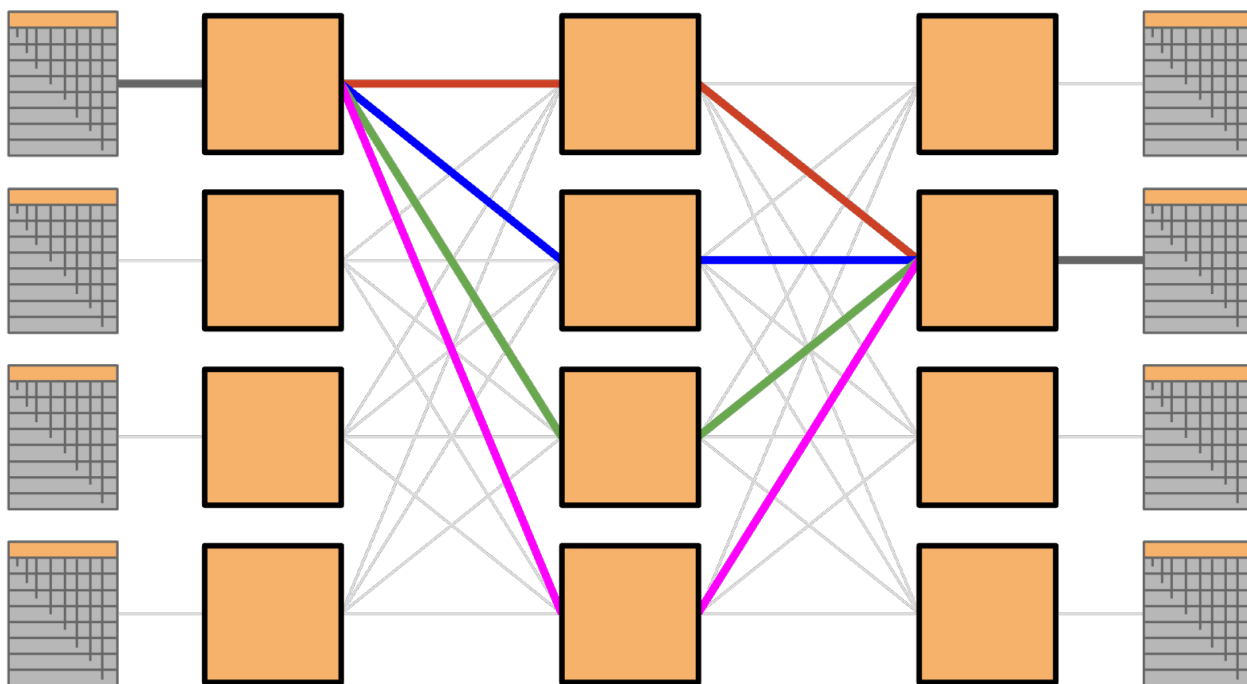


图 2-6（图源：Berkeley CS168 教材）：folded Clos 中任意一对服务器之间存在多条等价路径（彩色线）——多路径既是容错红利（第三部分），也是 ECMP 哈希极化的舞台（第四部分）。

## 2.4 Rail-Optimized Fat-Tree (ROFT)：为大模型训练做的优化

普通 fat-tree 对流量模式无假设。但大模型训练用的是 NCCL 集合通信，流量有强结构。ROFT 的做法（配合第一部分的 rail 概念）：

8 卡服务器的每张 GPU 各有一张网卡，把**所有服务器的第 i 号网卡**接到**同一组 Leaf**，构成第 i 条 rail。效果：

- 同 rail 内任意两 GPU 一跳直达（只过一台 Leaf）；
- NCCL 的 ring/tree 算法尽量把流量约束在 rail 内，跨 Spine 的流量最小化——Spine 层压力大幅减轻。

### Spine Switches

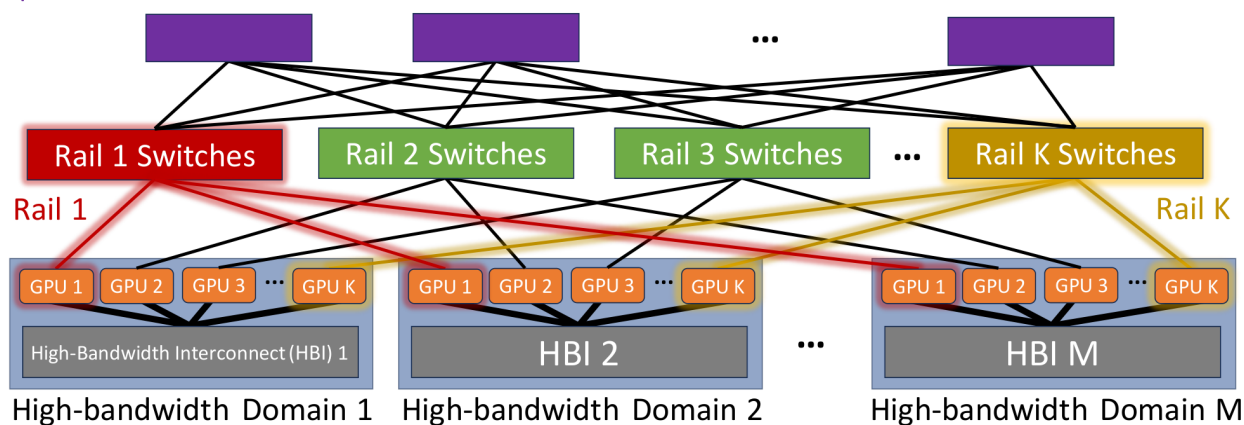


图 2-7（图源：Wang et al., HotI 2024, arXiv 2307.12169）：Rail-optimized fat-tree——Rail 1~K 各自的 Leaf 组 + 公共 Spine 层 + 服务器内高带宽域（HBD）的分层结构。

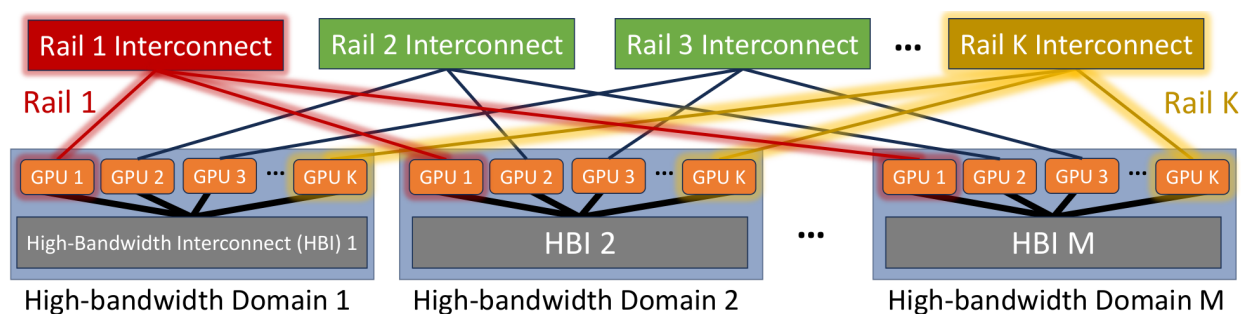


图 2-8 (图源: 同上, arXiv 2307.12169) : 对照极端形态 rail-only——干脆去掉 Spine, 各 rail 内部直连、rail 间不互通。成本最低, 但只能承载可以被严格按 rail 切分的流量。

NVIDIA DGX SuperPOD、NVL72 参考架构都是 rail-optimized 无阻塞 fat-tree: GB200 SuperPOD 用 32 台 Leaf 分 8 条 rail; GB300 NVL72 Enterprise RA 是双平面、每平面 4 台 Leaf, CX-8 的 800G 口拆成 2×400G 分接两个平面 (前一篇文档讲过)。

**ROFT 的局限** (这是 ZCube 的攻击点, L1 出自 ZCube 博客与论文): rail 映射是一种**静态的、事先画好的**流量规划。训练 all-reduce 的流量模式恰好和这个静态规划对齐, 所以 ROFT 在训练场景表现好。但推理场景 (尤其 PD 分离后) KV cache 的源和目的地是**运行时动态决定、强不对称的**, 静态 rail 映射不再等价于负载均衡——流量会集中到少数 Leaf 及其上行链路, 形成结构性热点。第四、五部分展开。

## 2.5 Dragonfly / Dragonfly+: HPC 来的省钱高手

Dragonfly 来自高性能计算 (HPC) 世界: **分组直连**拓扑。交换机分成若干组, 组内全互联 (或接近全互联), 组与组之间只用少量昂贵的"全局链路"连接。Dragonfly+ 允许组本身用路由器构成 (而非单台大路由器), 从而可以无限分组扩展, 彻底消灭 Super-Spine 层。

- 跳数: 组内 1-2 跳, 跨组最多 3-5 跳;
- 成本杀手锏: 一个公开测算 (Asterfusion/cloudswit.ch, 4096 rack、10 pod、64×100G Spine 口径, L2) 显示 Dragonfly+ 网络总成本约为三层 Clos 的 **66%** (\$5.61M vs \$8.49M), 对比四层 Clos 时仅 **32%**——省掉 Super-Spine 256 台、光模块 32,768 只;
- 代价: 组间全局链路是稀缺资源, 需要**自适应路由** (动态避开热点链路), 路由复杂度高; 组间割接/故障影响面大。

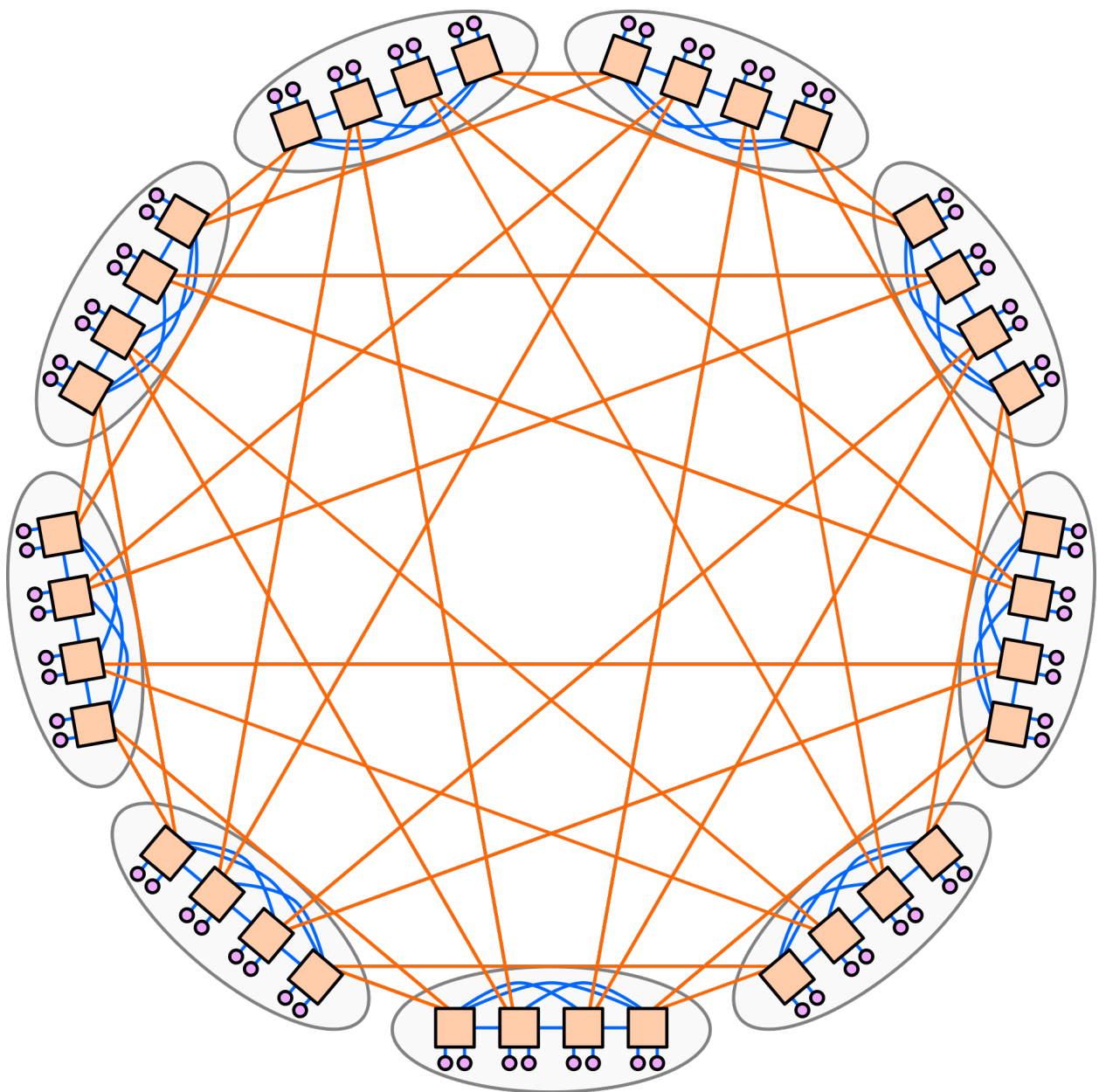


图 2-9（图源：Wikimedia Commons, Dragonfly-topology.svg）：Dragonfly 圆形布局——组内全互联（密），组间只留少量全局链路（稀）。

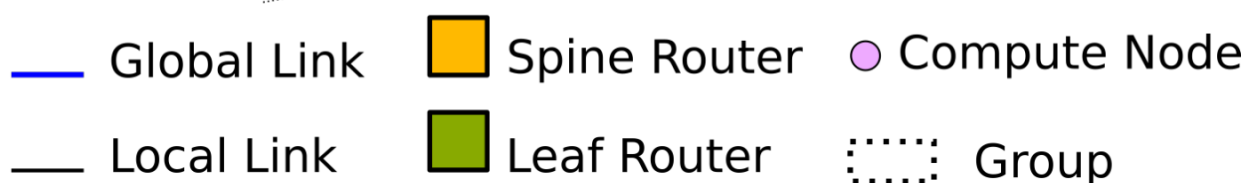
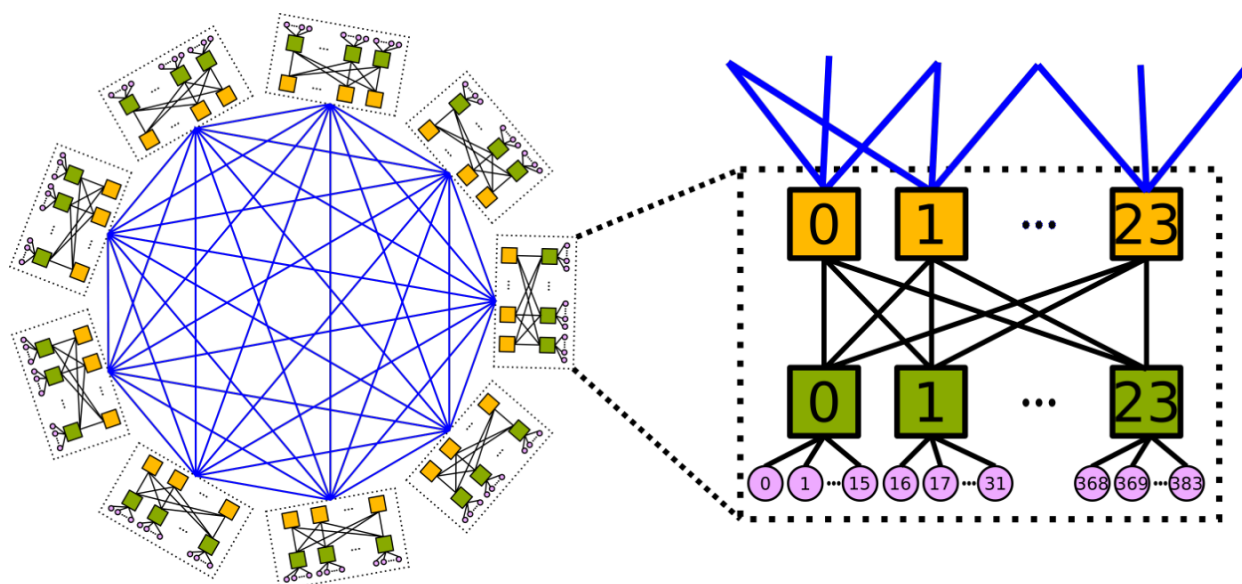


图 2-10（图源：arXiv 2406.15097, Dragonfly+ 应用干扰建模论文插图）：Dragonfly+——组本身由 Spine/Leaf 路由器对构成，可无限分组扩展，彻底消灭 Super-Spine 层。

典型用户：HPE Cray Slingshot、部分 Cisco/Microsoft 集群。

## 2.6 3D Torus 与 Hypercube：直连网络的另一极

**3D Torus（三维环面）**：每个节点在 x/y/z 三个维度上直接连接邻居，边缘回绕成环。Google TPU 走这条路：ICI 直连构成 2D/3D torus 并配 OCS（光路交换）做可重构；TPU v4 pod 4096 芯片，Ironwood 3D torus 达 9,216 芯片。Fugaku 用 6D Tofu，IBM BlueGene 用 3D torus。

- 优点：交换机和光模块用量极少（光模块:芯片约 1:3，靠 OCS），all-to-all 带宽利用好，成本低；
- 缺点：平均跳数随  $\sqrt{N}$  增长，时延高（训练可以容忍，推理 TTFT 敏感）；拓扑"定型"，扩容割接麻烦；规模有上限（TPU 系目前 ~9,600 芯片/pod）。

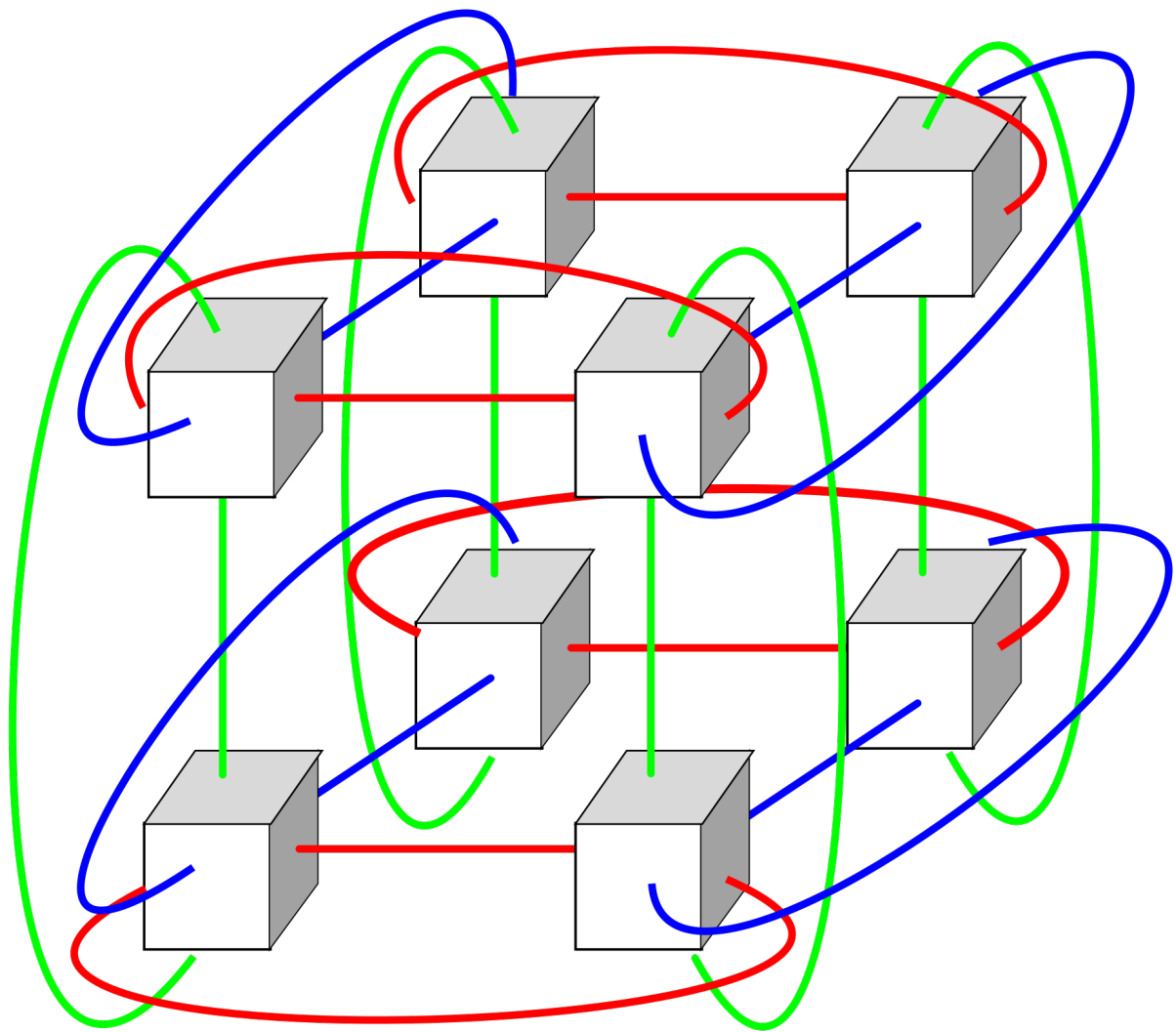


图 2-11 (图源: Wikimedia Commons, 2x2x2torus.svg) :  $2 \times 2 \times 2$  三维环面——每个节点在三个维度上直连邻居、边缘回绕成环。



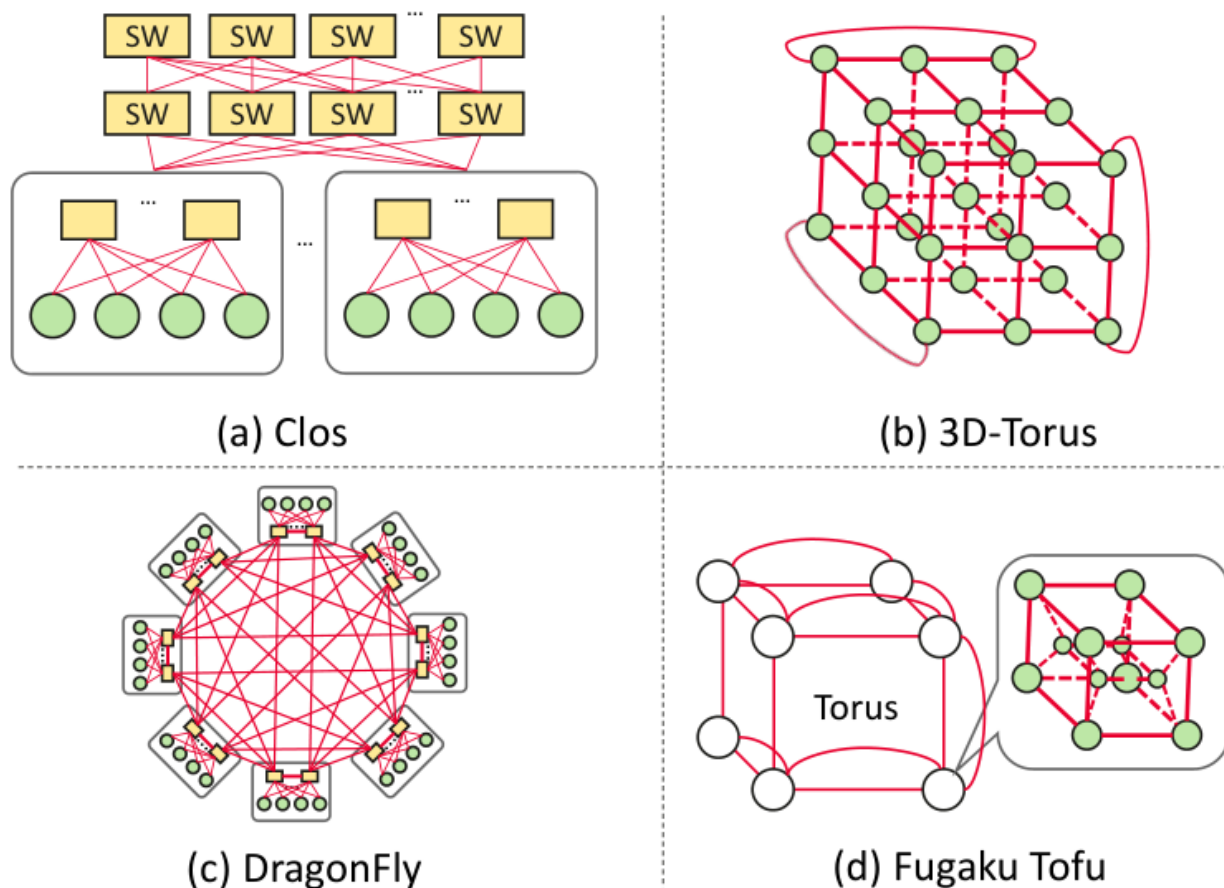


图 2-12（图源：arXiv 2503.20377，UB-Mesh 学术复现论文，非华为官方图）：Clos / 3D-Torus / Dragonfly / 富岳 6D Tofu 四种拓扑并排对比，可以直观看到"层级互联"与"节点直连"两大路线的形态差异。

**Hypercube（超立方体）**：N 个节点按二进制编号，编号差一位的节点直连，直径  $\log_2 N$ 。结构优雅但端口数随维度增长、布线噩梦，今天主要作为思想渊源存在（NVLink 域内互联有其影子）。

## 2.7 横向对比表

| 拓扑                         | 直径/跳数              | 扩展性            | 成本                       | 故障域特征                         | 路由复杂度          | 典型用户                          |
|----------------------------|--------------------|----------------|--------------------------|-------------------------------|----------------|-------------------------------|
| Fat-Tree/Clos (leaf-spine) | 3 跳（两层）            | 三层到 50 万+ 端点   | 高（交换机+光模块多）              | 单 Leaf 故障影响一个接入组；Spine 有多路径冗余 | 低（ECMP），但易哈希极化 | NVIDIA SuperPOD、Meta、多数 GPU 云 |
| Rail-Optimized FT (ROFT)   | 同 Clos, rail 内 1 跳 | 同 Clos         | 同 Clos                   | rail 级隔离                      | 低              | DGX SuperPOD、腾讯星脉、百度百舸        |
| Dragonfly/Dragonfly+       | 组内 2 跳、跨组 3-5 跳    | 超大（HPC 百万级）    | 约为 3 层 Clos 的 66%（L2 测算） | 组间全局链路少，割接影响大                 | 高（需自适应路由）      | HPE Cray Slingshot            |
| 3D Torus                   | 随 $\sqrt{N}$ 增长，高  | TPU ~9,600/pod | 低（大量直连+OCS）              | 局部化，割接/扩容难                    | 中（维度序+OCS 重构）  | Google TPU、Fugaku、Blue Gene   |
| Hypercube                  | $\log_2 N$ 跳       | 端口数随维度增        | 中高                       | 均匀                            | 中              | 早期并行机                         |

| 拓扑                    | 直径/跳数       | 扩展性                                     | 成本                             | 故障域特征                           | 路由复杂度             | 典型用户                           |
|-----------------------|-------------|-----------------------------------------|--------------------------------|---------------------------------|-------------------|--------------------------------|
| Rail-only (rail 间不互通) | rail 内 1 跳  | 受限                                      | 最低                             | rail 完全隔离                       | 极低                | 小规模训练集群                        |
| ZCube (扁平二部图)         | 2 跳         | 一层 51.2T 支撑 4,096 GPU (端口预算口径, 见 5.3 节) | 比 Clos 省 1/3 交换机+光模块 (L1 官方口径) | 核心链路故障影响 1/m <sup>2</sup> GPU 对 | 极低 (唯一路径) + 控制器   | 智谱 GLM-5.1 生产集群                |
| 华为 UB / UB-Mesh       | 超级节点内无阻塞全互联 | 目标百万级芯片                                 | 针对 Clos 超线性成本问题                | 分层故障域                           | 协议+拓扑一体化          | CloudMatrix38 4、Hot Chips 2025 |
| 阿里 HPN 7.0            | 两层          | "单层千卡、两层万卡"                             | 双 ToR 双平面                      | 双平面物理隔离消除哈希极化                   | 端网协同 (μFabb、HPCC) | 阿里云                            |

中国厂商的公开方案一句话各记一个关键词：**阿里 HPN** = 双平面物理隔离 + 端网协同；**腾讯星脉** = 多轨 Clos + TCCL/GOR；**百度百舸** = 多轨 + 两层 Clos + 自适应路由；**华为 UB-Mesh** = 统一总线 + 分层网格，直指"Clos 成本随规模超线性增长，上万节点后网络设备费可能超过计算芯片"（Hot Chips 2025）。

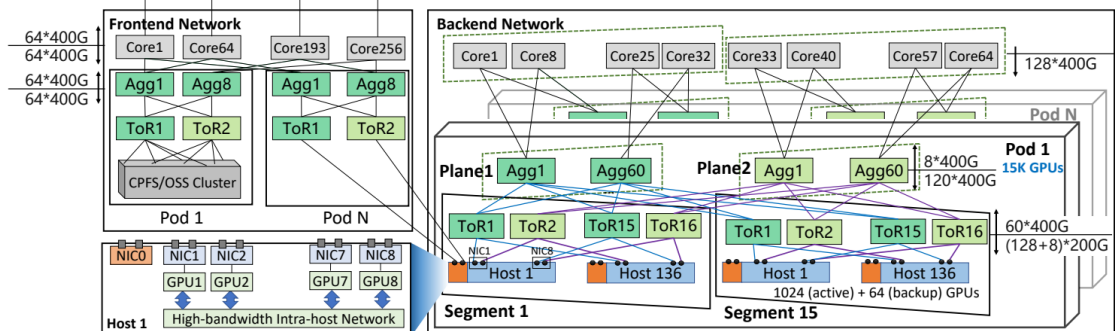


图 2-13（图源：Qian et al., SIGCOMM 2024 《Alibaba HPN》Figure 7, L1）：阿里 HPN 总体架构——双平面设计下每张网卡双口分接两个独立平面，Segment/Pod 两级组织，前后端网络分离，单集群 15K GPU 规模。

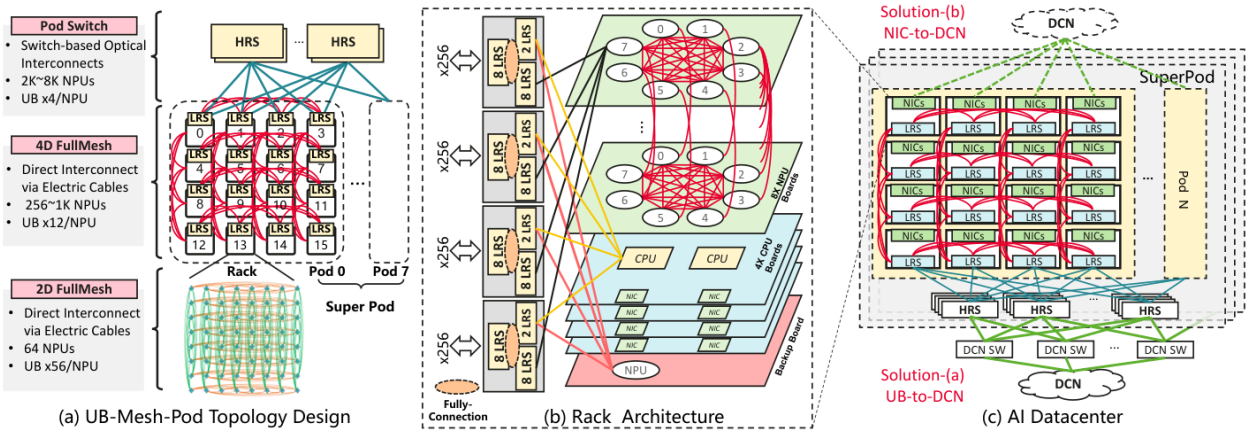


图 2-14（图源：arXiv 2503.20377，学术界复现论文而非华为官方发布，架构细节以华为 Hot Chips 2025 演讲为准）：UB-Mesh 分层结构——SuperPod/Pod/Rack 层级 + NPU 全互联的统一总线思路。

## 2.8 小结

- Clos (1952) 的核心思想：多级小交换机级联替代大矩阵；严格无阻塞条件  $m \geq 2n-1$ 。Fat-tree 是 Clos 的树形特例，folded Clos = leaf-spine。

- 规模公式：**两层无阻塞** =  $R^2/2$ ，**L 层**  $\approx R \cdot (R/2)^{(L-1)}$ 。R=128 时两层 8192 端点、三层 52 万端点——万卡恰好在两层/三层分界线上。
- ROFT 用静态 rail 映射讨好训练流量，但对动态推理流量失效。
- Dragonfly+ 靠分组直连省掉整层 Super-Spine（成本 ~66%），代价是路由复杂；3D torus 靠节点直连省光模块，代价是跳数多、规模封顶。
- 网络拓扑设计的永恒三角：**成本、直径（时延）、路由复杂度**，没有任何拓扑能同时最优。

## 2.9 问答 FAQ

### Q7: Clos 的"严格无阻塞 $m \geq 2n-1$ "在数据中心里还适用吗？

思想适用、参数不同。数据中心 folded Clos 的等价条件是每台 Leaf 的上下行带宽 1:1（下行  $R/2$ 、上行  $R/2$ ），这本质上就是  $m \geq n$  的可重排无阻塞口径——因为分组交换可以统计复用，不需要电路交换的"严格"保证。

### Q8: 为什么万卡是两层的极限？

R=128 的两层无阻塞上限  $R^2/2 = 8192$  个端口。万卡（10,000+）超过这个数，要么上三层，要么换更大的交换机 radix，要么换拓扑（比如 ZCube 的一层方案，第五部分见）。

### Q9: Spine 交换机为什么不直接接 GPU？

Spine 的每个端口都用来接 Leaf 才能最大化 Leaf 间带宽；让 Spine 接端点等于浪费宝贵的互联端口，违背了 Clos"分层换规模"的初衷。

### Q10: Dragonfly 那么便宜，为什么 GPU 云厂商不用？

三个原因：跨组跳数多导致时延不可控；需要自适应路由，与现有以太网 ECMP 运维体系不兼容；组间全局链路故障影响面大。它在 HPC（流量模式更规则、作业更长）里是合理的，在 LLM 推理（时延敏感、流量动态）里风险高。

### Q11: rail 内一跳是怎么做到的？

所有服务器的第 i 号网卡都接同一台（组）Leaf，两张 GPU 同属 rail i 时，它们的通信路径是 GPU→Leaf\_i→GPU，只过一台交换机。

## 第三部分 成本计算与故障域入门：手把手算账

### 3.1 组网总成本公式

总成本 = 交换机（数量 × 单价）+ 光模块（数量 × 单价）+ 线缆/AOC + 布线施工 + 功耗（OpEx，按 3-5 年计）

价格锚点（2026 年市场行情，L2 经验口径，用于量级估算而非精确报价）：

- 800G SR8/DR8 光模块：约 \$360-380；800G LR8：\$420-450（接近 \$0.5/Gbps）；
- 1.6T OSFP：约 \$1,300-1,500，预计两年后降至 ~\$1,100；
- 400G DR4 模块：约 \$150-250 量级（按 \$0.5/Gbps 比例估）；
- 51.2T 白盒交换机（64×800G Tomahawk5 代际）：整机数万美元级，按 **\$3-6 万** 估算。

**一条重要行业经验（L2，务必记住其估算性质）：在 400G/800G AI 集群中，光模块+线缆常占网络硬件成本的 50-70%，且速率越高占比越高。** 也就是说，组网预算的大头不是交换机机箱，而是那些小小的光模块。

### 3.2 两层 Clos 的光模块数量：为什么是 4N

设 N 张 GPU，每 GPU 一个 400G 口，两层无阻塞 Clos：

- 下行链路：每张 GPU 一条链路接到 Leaf，共 N 条；
- 上行链路：无阻塞要求 Leaf 上行带宽 = 下行带宽，所以上行链路也是 N 条（按带宽等值计）；
- 每条链路两端各一只光模块 → 光模块总数  $\approx 2 \times (N + N) = 4N$ 。

万卡集群  $\approx 4$  万只光模块。这个数字在第三部分会反复用来算账。

### 3.3 算例一：1024 GPU 两层 Clos

参数：N = 1024, R = 128 (51.2T, 128×400G)。

- Leaf 数量 =  $N / (R/2) = 1024 / 64 = 16$  台；
- Spine 数量 =  $R/2 = 64$  台？——等一下，这里有个工程细节：Spine 数量 = Leaf 的上行口数 = 每台 Leaf 64 个上行口，所以理论上是 64 台 Spine。但实践中 Spine 只需要“每台 Spine 有一个口接每台 Leaf”：16 台 Leaf × 每台 Spine 用 16 口，Spine 用不满。工程上通常会减少 Spine 台数、用满端口：16 台 Leaf × 64 上行 = 1024 条上行链路，每台 128 口 Spine 可承载 128 条 →  $\text{Spine} = 1024/128 = 8$  台。
- 交换机合计：16 + 8 = 24 台；按 \$4 万/台  $\approx$  \$96 万。
- 光模块：下行 1024 条 + 上行 1024 条 = 2048 条链路 × 2 端 = 4096 只；400G DR4 按 \$200/只  $\approx$  \$82 万。
- 总网络硬件 CapEx  $\approx$  \$180 万量级，光模块约占 45%。

注意这个比例已经接近“一半”——这就是 L2 经验口径的微观体现。

### 3.4 算例二：16384 GPU 三层 Clos

N = 16,384 超过了 R=128 两层的 8192 上限，必须三层。结构：若干 pod（每 pod 是 Leaf+Spine 两层）+ Super-Spine。

- Leaf =  $N/(R/2) = 16384/64 = 256$  台；
- 每个 pod：按标准 fat-tree 打包，Spine 与 Leaf 的关系同上。把 256 台 Leaf 分成 64 个 pod，每个 pod 4 台 Leaf + 4 台 Spine（每 Spine 用 4 口接 pod 内 Leaf、剩余 64-4=60 口……这里用标准公式更干净：Spine 总数 = Leaf 数 = 256 台的一半口径）——为避免口径混乱，用全网链路数来算：
- Leaf 下行链路 = 16,384 条；Leaf 上行链路 = 16,384 条（无阻塞）；
- Spine 到 Super-Spine 层：Spine 的“上行”带宽同样 1:1，再加 16,384 条链路等值；
- 交换机数：Leaf 256 + Spine 256 + Super-Spine 128  $\approx 640$  台（每层的台数=该层链路数/R 推出）。

光模块：三层链路总条数  $\approx 3 \times 16,384 \approx 49,152$  条，× 2 端  $\approx 98,304$  只（约 6N，比两层的 4N 多了 50%——多一层就多一层链路）；

成本量级：交换机 640 × \$4 万  $\approx$  \$2,560 万；光模块 ~9.8 万只 × \$200  $\approx$  \$1,970 万；合计 \$4,500 万量级（约人民币 3 亿+），光模块占比约 44%。

记住两个比例：两层 Clos 光模块  $\approx 4N$  只；三层  $\approx 6N$  只；每多一层，光模块多 2N 只。而 ZCube 的核心卖点恰好是“砍掉一层”。

### 3.5 故障域入门：定义与分类

**故障域（fault domain）**：单一组件失效时，受影响（通信受损）的 GPU 范围。组网方案比较时，“故障域大小”和“成本”是同等重要的一阶指标。逐类分析（以两层 Clos、8 卡机、GPU:NIC=1:1、8 rail ROFT 为基准）：

- **单条接入链路故障**：只影响 1 张 GPU（若网卡单端口）；双端口/双平面设计下第二端口接管，影响为零到一半带宽。
- **单台 Leaf 故障**：影响其下联全部 GPU——64 口 Leaf 下接 64 张 GPU 失去一个平面的接入；ROFT 下是"该 rail 全挂"：全集群每张 GPU 损失 1/8 网卡带宽，NCCL ring 需绕路或整体降速。
- **单台 Spine 故障**：多路径冗余摊薄影响（损失约 1/m 的 Spine 带宽），ECMP 微秒级自动绕开——这是 Clos 多路径的**容错红利**。
- **rail 故障**：见上，rail 级隔离意味着 rail 故障是"全员轻伤"而非"局部死亡"。
- **plane 故障**：双平面各担 50% 流量，一个平面整体失效损失一半带宽，业务可继续——这是双平面方案的卖点。

### 3.6 MTTF 模型与实测统计

简化模型：单节点故障率  $r_f$ （次/节点·天），N 节点作业的平均无故障时间  $MTTF = 1/(N \cdot r_f)$ 。规模越大，故障越频繁——线性恶化。

**Meta RSC 实测**（L1，arXiv 2410.21680，11 个月、两个研究集群）：

- 集群故障率：RSC-1 = 6.50 次/千节点·天，RSC-2 = 2.34 次/千节点·天；
- 硬件故障只占作业数的 0.2%，但影响 18.7% 的 GPU 运行时间（大作业受害最重）；
- 最常见可归因硬件故障前三名：后端网络（IB 链路）、文件系统、GPU（显存错误为主）；
- 作业 MTTF：4,096 GPU  $\approx$  10 小时；16,384 GPU  $\approx$  **1.8 小时**；131,072 GPU  $\approx$  0.23 小时。十万卡级作业按 RSC-1 故障率 MTTF  $\sim$  15 分钟——checkpoint 间隔要压到  $\sim$  7 分钟才能保住 50% 有效训练时间比。

**Google Jupiter 的故障域设计**（L1，SIGCOMM'22 Jupiter Evolving）：把每个 block 的端口划分为 4 个故障域（各 25%），要求损失任一故障域后剩余拓扑保留  $\geq 75\%$  吞吐；OCS 让单 rack 故障的容量损失均匀摊薄（32 rack 坏 1 个 = 全网  $-1/32$ ）。

另有一个对后文重要的数据点（L1，arXiv 2607.21893 § 6.3 引用）：**数据中心网络硬件的年故障率一般  $< 1\%$** 。这意味着任何"为罕见故障付出每日性能税"的设计都值得重新审视——ZCube 的容错论证正是建立在这个数据上。

### 3.7 小结

- 组网成本 = 交换机 + 光模块 + 线缆 + 施工 + 功耗；**光模块约占一半以上**（L2 经验口径），速率越高占比越大。
- 两层 Clos 光模块  $\approx$  4N 只，三层  $\approx$  6N 只；16384 卡三层 Clos 网络硬件成本是人民币数亿量级。
- 故障域大小排序：plane  $>$  rail  $\approx$  Leaf  $>$  Spine（多路径摊薄） $>$  单链路。
- 万卡以上作业的 MTTF 以小时甚至分钟计（Meta 实测）；故障域设计与 checkpoint 策略必须联动。
- 网络硬件年故障率  $< 1\%$ ——容错机制的成本应按这个频率来定价。

### 3.8 问答 FAQ

**Q12：为什么光模块比交换机还烧钱？**

链路数随 GPU 数线性增长（4N~6N 条链路），而交换机台数增长较慢；单只模块虽只几百美元，乘上几万只就是千万美元级。速率翻倍时模块价格降幅慢于带宽增幅（\$0.5/Gbps 的经验线下降缓慢），占比进一步上升。



### Q13：三层 Clos 比两层贵多少？

链路数多 50%（6N vs 4N），交换机台数多出一层。对 16k 卡量级，CapEx 大约翻倍——这就是"万卡要不上三层"永远是预算会议上的争论点的原因。

### Q14：Spine 故障为什么影响小？

两层 Clos 有 R/2 台 Spine 并行，单台故障只损失 1/m 的跨层带宽，ECMP 在微秒级把流量哈希到剩余路径上。多路径在容错上是真金白银的红利——后面会看到 ZCube 放弃它换来了什么。

### Q15：MTTF 1.8 小时是不是说 16k 卡集群每两小时就全挂一次？

不是"全挂"，是"作业每 1.8 小时遇到一次需要处理的故障事件"（某节点/链路/GPU 出问题）。靠 checkpoint + 热备 + 快速隔离把损失控制在分钟级。但频率之高足以让"故障域大小"成为一阶设计目标。

### Q16：Google Jupiter 的"4×25% 故障域"思路通用吗？

思想通用：把网络切成 N 个对称故障域，让单域损失的吞吐  $\leq 1/N$ ，并用软件（OCS 重构/流量工程）摊薄。它是"把冗余做在拓扑内"的代表——第五部分会看到 ZCube 走了相反的路。

## 第四部分 大模型训练/推理的流量特征：网络到底在扛什么

### 4.1 五种并行策略，五种通信模式

大模型把计算切到成千上万张 GPU 上，切法不同，网络要扛的流量就完全不同：

| 并行策略           | 切法               | 通信模式                                             | 流量特征                                                      |
|----------------|------------------|--------------------------------------------------|-----------------------------------------------------------|
| DP 数据并行        | 每卡完整模型副本，切 batch | 梯度 <b>AllReduce</b> (ring/tree)                  | 同步大象流：全员同时、单流带宽大、周期性强；通信量 $\propto$ 模型参数                  |
| TP 张量并行        | 单层矩阵乘切到多卡        | 层内 AllReduce/AllGather                           | 频繁、消息中等、 <b>时延极敏感</b> → 一般限制在 NVLink 域内（单机 8 卡或 NVL72 一柜） |
| PP 流水线并行       | 按层切分             | 相邻 stage 间 <b>P2P</b> 传激活                        | 通信量小、模式固定可预测                                              |
| SP/CP 序列/上下文并行 | 切 sequence 维     | AllGather/RS                                     | 常与 TP 组合                                                  |
| <b>EP 专家并行</b> | MoE 不同专家放不同 GPU  | 每层 MoE <b>两次 all-to-all</b> (dispatch + combine) | 小消息、目的地运行时决定、逐层变化、倾斜突发——本文主角                              |

前四种的流量要么规则（PP）、要么全员同步对称（DP）、要么被关在 NVLink 域内（TP）——传统的 rail-optimized Clos 对付它们绰绰有余。**EP 不一样**。

### 4.2 MoE 与 EP 的 all-to-all：为什么它是最难扛的流量

以 DeepSeek-V3 为代表（671B 总参、37B 激活、256 路由专家 + 1 共享专家、top-8 路由、hidden 7168、58 个 MoE 层），EP 把每层 MoE 变成这样一轮通信：

1. **Dispatch**：每张 GPU 把本地 token 按 gating 网络的路由结果，发送给持有目标专家的 GPU；
2. 专家计算；
3. **Combine**：把专家输出收回，按原 token 序合并。

通信量估算（L2，glennklockwood.com 口径，4096 token/序列、bf16）：每 token 隐状态 14,336 B；每 GPU 每前向发出约 468 MB、收回 468 MB；58 层累计 **每前向 54 GB all-to-all 流量，反向再来 54 GB**。



它与传统 all-reduce 的本质区别有四：

- **小消息**：token 激活粒度小，单条消息 ~7KB 量级（UCCL-EP 论文口径）；dispatch/combine 极其频繁（每 token 选 8 个专家）。
- **目的地运行时决定**：通信矩阵由 gating 在运行时算出，**逐层、逐 step 变化**——没有任何静态拓扑映射能事先对齐它。
- **突发与倾斜**：热专家接收的 token 远超均值（负载倾斜）；研究进一步指出逐 NIC round-robin 调度在倾斜 MoE 流量下会产生"指数级 incast"——最热接收端的到达率可增长到线速的 N 倍，工作负载几乎全堆在交换机 buffer 里。
- **不可被计算掩盖**：EP all-to-all 难以与计算重叠；未优化时 DeepSeek-V3 训练中通信时间可占 >50%（NVIDIA 开发者博客口径）。EP 程度越高（Wide-EP，跨节点大 EP 域），跨节点通信越主导运行时。

一句话总结（值得抄在笔记本上）：**EP 把每层 MoE 变成一次倾斜的、运行时决定目的地的 all-to-all；它要的不是平均带宽，而是"任意流量矩阵下都不出结构性热点的拓扑 + 能吸收 incast 的端到端机制"。**

## ZCube 不替代 NCCL，而是给 NCCL 提供更均衡的物理底座

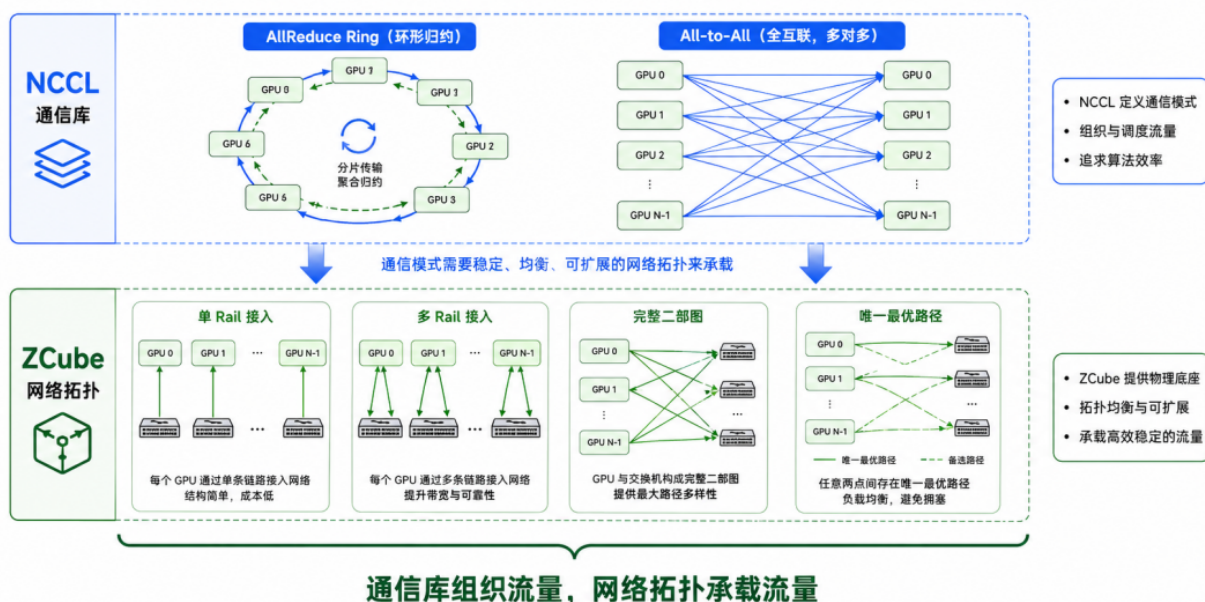


图 4-1（图源：微信公众号「驭训网络」系列文章二，L2）：NCCL（AllReduce Ring / All-to-All 算法层）与网络物理底座的分工——无论是 ZCube 还是别的拓扑，它都不执行规约、不替 NCCL 做集合通信算法，只负责让通信关系落到更合理的物理路径上。

## 4.3 PD 分离与 KV Cache 传输：推理侧的新流量

推理系统近年主流化了一个架构变化：**PD 分离**（Prefill/Decode 分离，Splitwise ISCA'24、DistServe OSDI'24、Mooncake FAST'25）。Prefill 阶段算密集、Decode 阶段带宽/显存密集，两类节点分开部署、各自弹性伸缩。代价是：**KV cache 必须从 Prefill 节点跨网络传到 Decode 节点**。

这种流量的特征（L1，ZCube 博客的生产观测）：

- **源-目的强不对称**：某个时刻可能少数 Prefill 节点向大量 Decode 节点发送，或反之；
- **流量规模动态变化**：请求到达率、上下文长度（差数十倍）都在变，几乎不可预测；
- **源和目的各占连续 GPU 块**：Prefill/Decode 池通常按连续编号分配——在 ROFT 下，连续块恰好映射到少数几台 Leaf 的上行链路，**静态 rail 映射彻底失效**，形成结构性热点（这点第五部分配图详解）。

一个具体例子（L2，微信系列文章二）：P1→D3 200MB、P2→D3 300MB、P4→D3 500MB、P5→D7 50MB——某个 Decode 节点、某张 NIC、某台 Leaf 会突然成为热点，而下一分钟热点又搬家了。

为什么结构性热点杀伤的是**尾时延**：一次请求的 TTFT（首 token 时延）取决于它最慢的那段 KV 传输；只要网络里有一条链路持续拥塞，总会有请求恰好踩中它——P99 被拖垮。ZCube 生产实测中 TTFT P99 改善 40.6%，针对的就是这个尾巴。

## ROFT 适合规则通信，却可能被动态 KV 流量打出热点

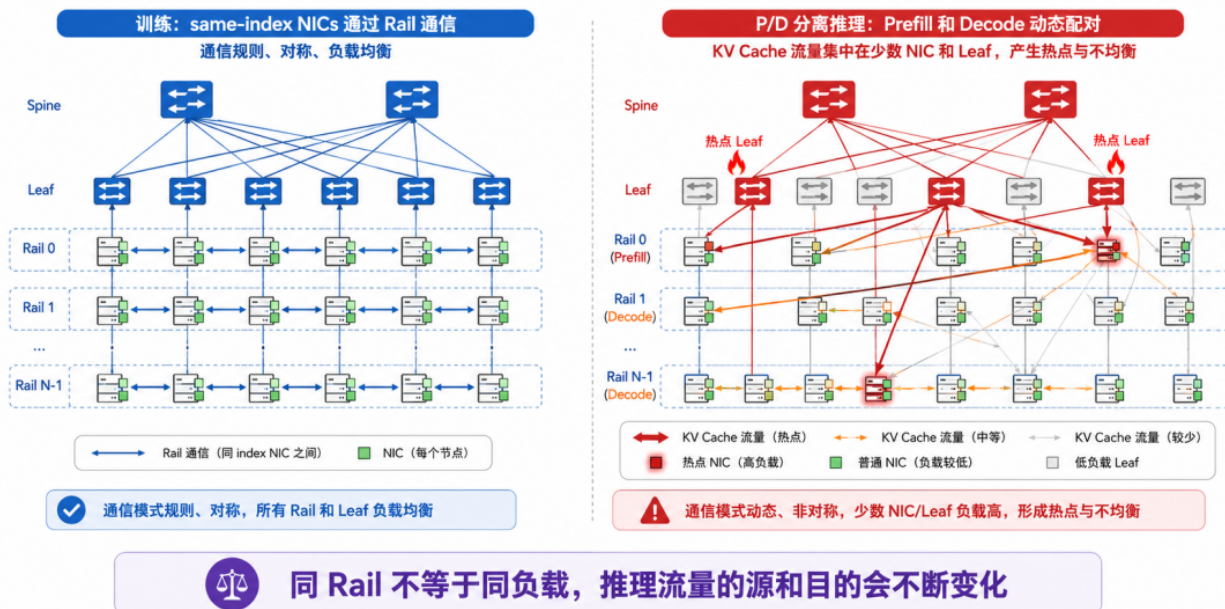


图 4-2（图源：微信公众号「驭驯网络」系列文章三，L2）：左——训练 all-reduce 的规则 rail 通信恰好与 ROFT 的静态映射对齐；右——P/D 分离后 KV cache 流量的源/目的动态不对称，少数 NIC 与 Leaf 被打成热点。一句话对比："ROFT 优化规则的 rail 通信，ZCube 面对动态而不对称的 GPU 对 GPU 通信"。

## ZCube 优化的是 KV Cache 流量如何在网络中相遇

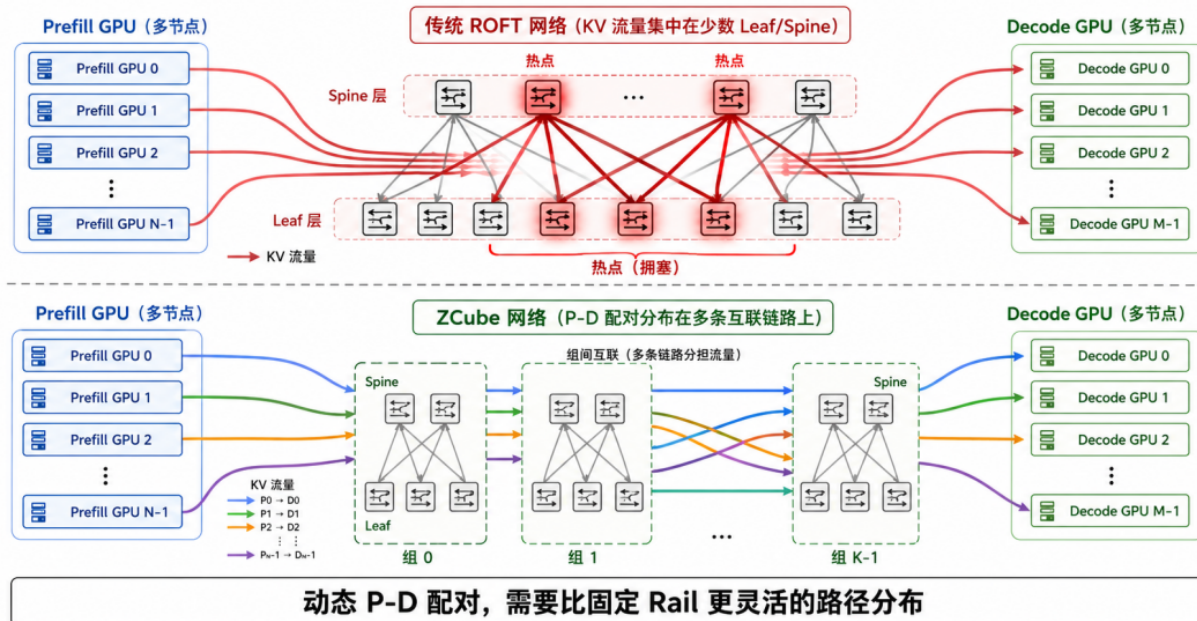


图 4-3（图源：微信公众号「驭驯网络」系列文章二，L2）：同一组 P→D KV cache 大流在 ROFT 中挤过少数 Leaf/Spine 路径（左），在 ZCube 中被奇偶二部图结构分散到更宽的组间链路空间（右）。原



话金句："优化的不是 KV Cache 本身，而是 KV Cache 在集群里的相遇方式。"

## 4.4 为什么 Clos + ECMP 扛不住：Braess 悖论的阴影

把前三节拼起来，传统方案的病根就清楚了。Clos/ROFT 的设计前提是"多条等价路径 + 静态哈希  $\approx$  负载均衡"。这个前提在 LLM 流量下三处失效：

1. **哈希极化**：LLM 通信是少数几条大象流（而不是互联网的海量小流），五元组哈希在流数少时很容易把多条大象流钉到同一条路径上，其他路径空转；
2. **静态 rail 映射失效**：流量矩阵逐 step 变化，事先画好的映射反而成为热点制造机（4.3 节）；
3. **拥塞盲选路**：ECMP 选路时不看拥塞，两条流挤在一条 400G 链路上各跑 200G，旁边一条 400G 链路空着——网络总带宽宽裕，局部却在堵车。

### Spine 提供多路径，但不保证大流真的均衡

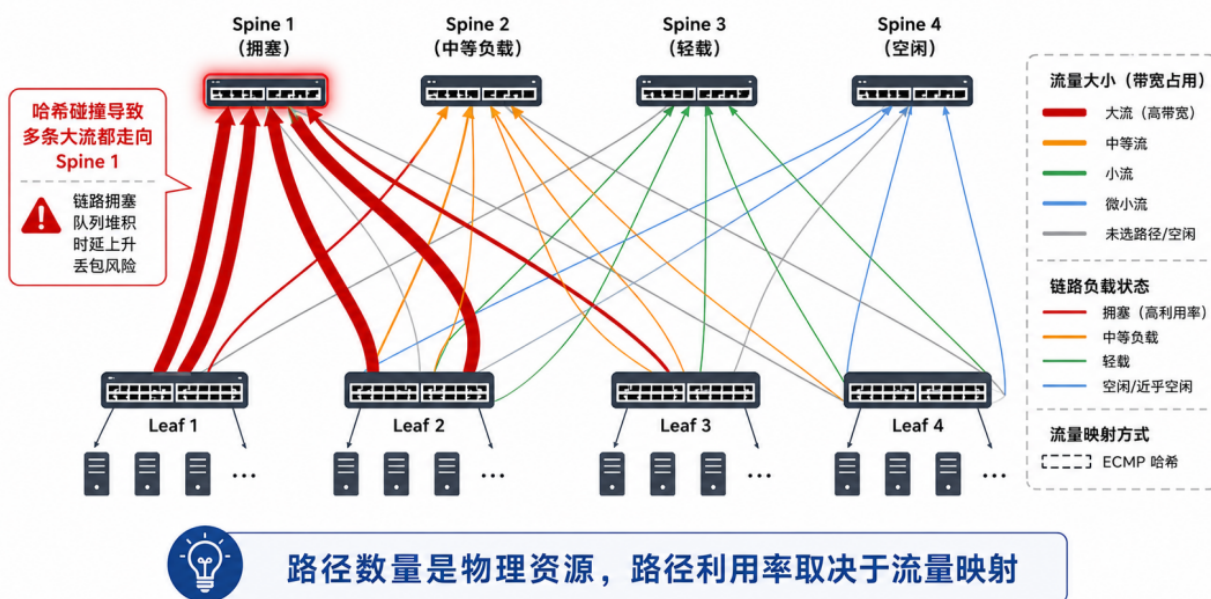


图 4-4 (图源：微信公众号「驭驯网络」系列文章三，L2)：Spine 层多路径但大流不均——ECMP 哈希不知道流有多大、链路是否正承载 KV 大流、源-目的对是否相撞，碰撞使多条大象流挤上同一台 Spine，其余路径空转。

ZCube 配套论文 (L1, arXiv 2607.21893) 把这个现象上升到了理论高度：**多路径 + ECMP 静态哈希属于"拥塞盲选路"，在 LLM 结构化流量下处于 Braess 悖论阴影中**——Braess 悖论原本说的是"给路网加一条新路，所有人的通行时间反而变长"，对应到这里就是"多给路径 + 盲选路，可能比少给路径 + 确定选路更差"。论文的量化结论：静态哈希的最坏失衡可达  $128\times$  (vs 自私路由博弈的  $4/3$  PoA 上界)；而 ZCube 因为每对 GPU 恰有一条最优路径 (PDI=1, 无选路自由度)，**天然免疫 Braess 悖论**。仿真对照：集群化 Prefill 流量下 ROFT 负载失衡 4.50，ZCube 仅 1.33 (计入空载链路)，在受载链路上恰为 1.00。

## 4.5 小结

- DP/TP/PP 的流量规则、可预测，ROFT 应付自如；**EP all-to-all 与 PD 分离的 KV cache 传输是传统架构的盲区**。
- EP 流量四毒：小消息、目的地运行时决定、倾斜 incast、难以被计算掩盖；DeepSeek-V3 每前向 54 GB all-to-all。

- PD 分离的 KV cache：源-目的动态不对称 + 源/目的各占连续 GPU 块 → ROFT 静态映射制造结构性热点 → 杀伤 TTFT 尾时延。
- 理论解释：ECMP 静态哈希是拥塞盲选路，最坏失衡  $128\times$ ，处于 Braess 悖论阴影中；拓扑层面"唯一最优路径"反而是出路。

## 4.6 问答 FAQ

### Q17: all-to-all 和 all-reduce 差在哪，为什么前者难？

All-reduce 是规则的全员归约，通信模式固定、可事先对齐 rail 映射；all-to-all 是"每人给每人发不同的东西"，且 EP 里收件人列表每步都变。网络要为"任意可能的流量矩阵"做准备，而不是为"一种已知模式"做优化。

### Q18: 7KB 小消息为什么可怕？

小消息意味着同样的字节量要拆成成千上万条流，每条流都是哈希表的一次抽奖——极化概率大增；同时小消息的传输时延中固定开销（包头处理、RDMA 建链/排队）占比高，对拥塞更敏感。

### Q19: 热专家 incast 为什么"不可避免"？

多个发送端同时向同一个热专家 GPU 发数据，最后一跳（进入该 GPU 网卡的链路）带宽是物理上限，流量超过它就必然排队——这是"最后一跳竞争"，任何拓扑都消除不了，只能靠端侧限速、buffer 吸收、专家负载均衡算法缓解。ZCube 自己也不声称解决它（见 5.7 两类拥塞的区分）。

### Q20: PD 分离不分不行吗？

不分（PD 融合）就没有 KV cache 跨节点传输，但 Prefill 和 Decode 的资源需求互相干扰、无法独立扩容，吞吐和成本都吃亏。主流生产系统（Splitwise/DistServe/Mooncake 一脉）都选择了分离，KV cache 传输随之成为一等公民。

### Q21: Braess 悖论用一句话怎么记？

"路多了，如果每个司机都盲目自选，所有人可能更慢。"对应网络："路径多了，如果选路不看拥塞（ECMP），可能比唯一路径更慢。"ZCube 的选择是把"选路"这个自由度从系统里删掉。

## 第五部分 ZCube 深入解剖

### 5.1 它是什么，谁做的

ZCube 是由智谱（Z.ai）、驭驯网络（Harnets.AI）、清华大学联合研发的全扁平 GPU 互联架构，面向大模型推理（PD 分离场景）并兼容训练（L1）。原始论文是 Yan et al. 的 From ATOP to ZCube（ACM SIGCOMM 2025），配套分析论文 arXiv 2607.21893。它已在 GLM-5.1 coding 的千卡生产集群落地：原 ROFT 组网迁移到 ZCube，GPU、软件栈、应用零改动，唯一变量是拓扑（L1，官方博客）。

架构总览（p=2 双端口网卡、n=32 GPU、每台交换机接 k=8 GPU 的最小示例）：

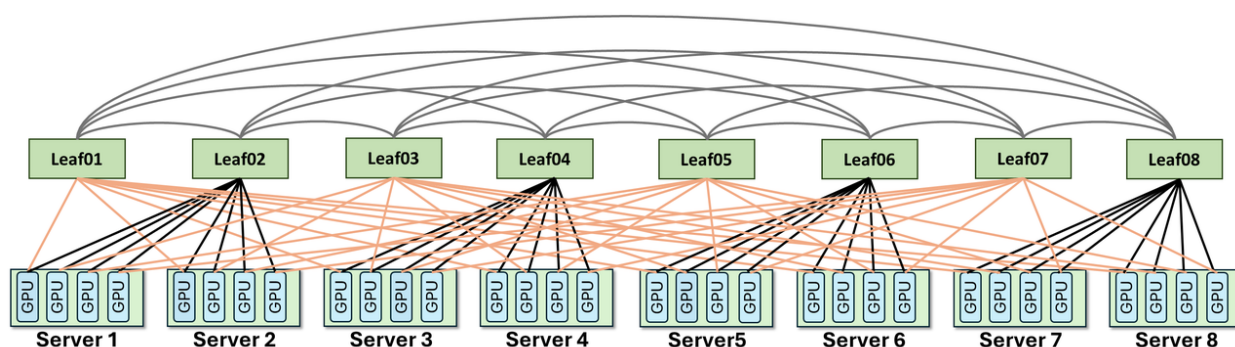
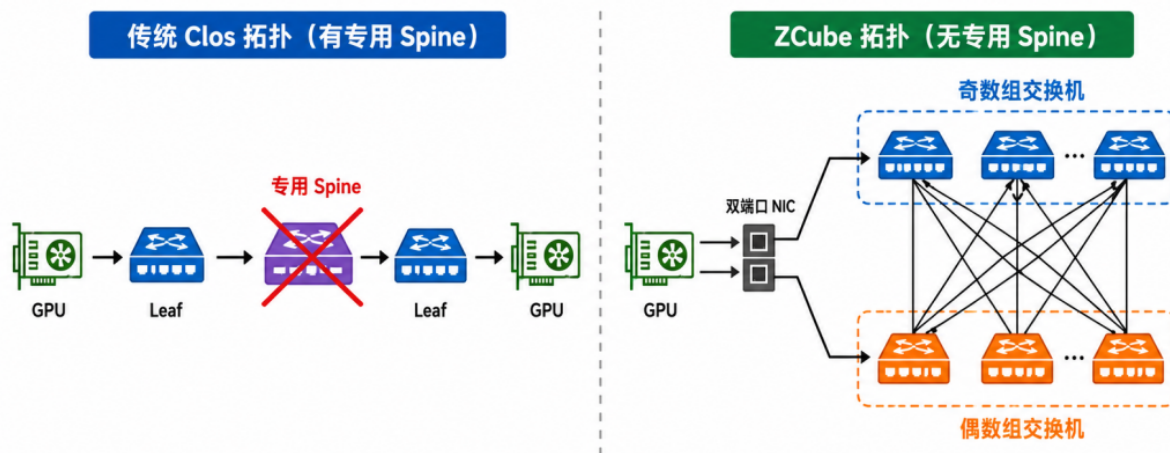


图 8（ZCube 官方博客 Figure 7）：8 台 Leaf 交换机排成一层，顶部灰色弧线是奇数交换机与偶数交换机之间的完全二部图互联；下方橙色（多轨）与黑色（单轨）两类接入线分别把每张 GPU 的两个网卡端口接入两组交换机。没有 Spine，没有第二层。

## ZCube 取消的是专用 Spine，不是交换机间转发



### 两组交换机同时承担 GPU 接入和跨组中转

图 5-1（图源：微信公众号「驭驯网络」系列文章三，L2）：传统 Clos 跨 Leaf 路径（服务器→Leaf→Spine→Leaf→服务器，3 跳）vs ZCube（服务器→接入交换机→接入交换机→服务器，2 跳）——注意 ZCube 取消的是“只负责中转的专用 Spine 设备”，不是跨交换机转发本身；转发工作被摊到了每一台接入交换机身上。

## 5.2 拓扑的数学：四条规则与两个公式

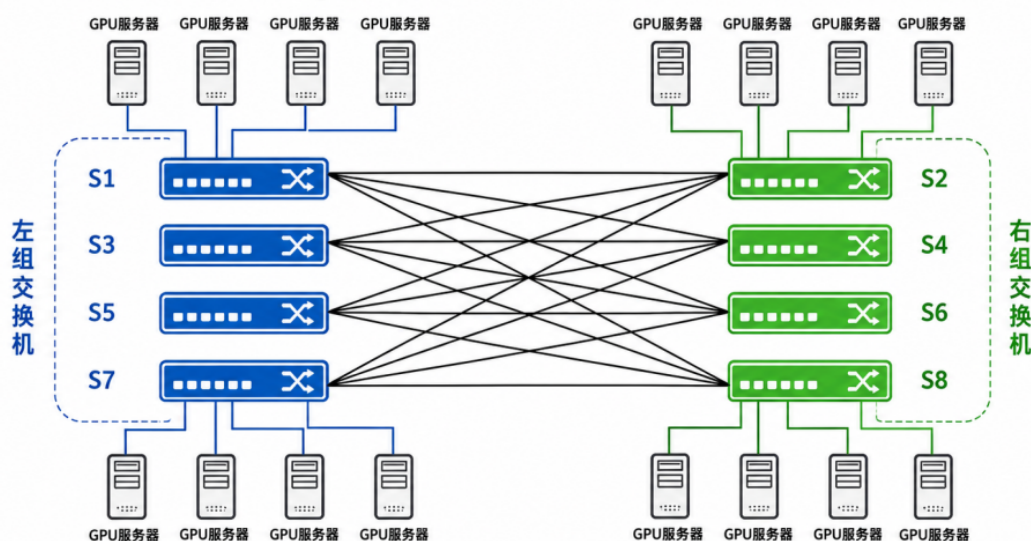
ZCube 的构造规则（L1，博客 § 3.1）：

1. 取消 Spine 层，只保留一层 Leaf 交换机；
2. Leaf 按序号奇偶分为两组，各  $m = n/k$  台（ $n$  = GPU 数， $k$  = 每台交换机接入的 GPU 数）；
3. 两组之间构成完全二部图  $K_{\{m,m\}}$ ：每台奇数交换机与所有偶数交换机直连，反之亦然；
4. 每张 GPU 的网卡有两个端口（ $p=2$ ），一个端口以“多轨”方式接入奇数组，另一个端口以“单轨”方式接入偶数组。

先看规则 3 的直观成本：为什么两组之间用完全二部图，而不是全部交换机全 Mesh？以 8 台交换机分两组为例（奇数组 S1/S3/S5/S7、偶数组 S2/S4/S6/S8）：完全二部图需要  $4 \times 4 = 16$  条组间链路；8 台全 Mesh 需要  $8 \times 7 \div 2 = 28$  条。二部图省掉的是组内链路（奇-奇、偶-偶）——而组内链路本来就是纯浪费：每张 GPU 的两个端口分属两组，跨交换机流量永远是“奇→偶”或“偶→奇”，组内直连没有任何流量需要它。推广到  $m+m$  台：二部图  $m^2$  条 vs 全 Mesh  $m(2m-1)$  条， $m=64$  时是 4,096 vs 8,128，省约一半。这就是“分组后的结构化全连接”：任意跨组交换机对恰有一条直连，链路数却接近减半。



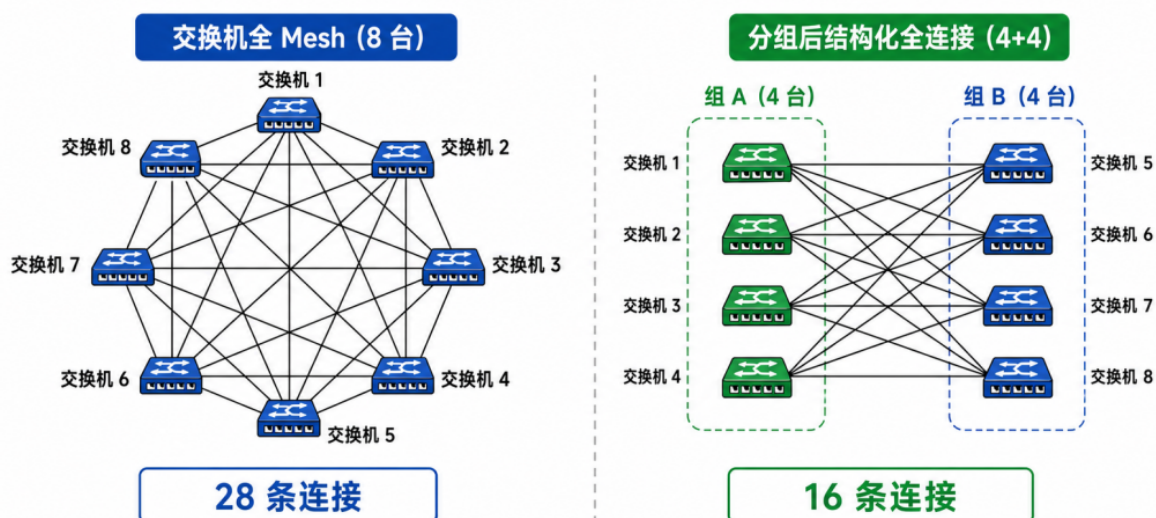
## 没有 Spine，靠两组交换机的完整二部互联



每台交换机既是接入节点，也是跨组流量的转发节点

图 5-2（图源：微信公众号「驭驭网络」系列文章三，L2）：8 台交换机分奇偶两组、组间完全二部互联（S1-S8 示例），组内不互联。

## ZCube 不是交换机全 Mesh，而是分组后的结构化全连接



只做跨组连接，避免无约束全互联

图 5-3（图源：同上）：8 台全 Mesh 需 28 条连接（左），4+4 完全二部图只需 16 条（右）——省链路但不损失“任意跨组交换机对一跳可达”的性质。

形式化映射公式（GPU 编号  $i$  从 1 起）：

端口 1（多轨，接奇数交换机）：编号  $((i-1) \bmod m) \times 2 + 1$

端口 2（单轨，接偶数交换机）：编号  $\text{ceil}(i/k) \times 2$

其中  $m = n/k$ 。多轨 = 按模位置分组（编号余数相同的 GPU 接同一台奇数交换机——同一台服务器内的 8 张 GPU 被均匀打散到所有奇数交换机）；单轨 = 按连续块分组（连续  $k$  张 GPU 接同一台偶数交换机——同一台服务器的 GPU 聚在一起）。

手把手算一遍  $n=32, k=8$  ( $m=4$ , 奇数交换机 4 台: 1,3,5,7; 偶数 4 台: 2,4,6,8) :

| GPU i | 端口1 (多轨) → 奇数交换机                | 端口2 (单轨) → 偶数交换机                 |
|-------|---------------------------------|----------------------------------|
| 1     | $(0 \bmod 4) \times 2 + 1 = 1$  | $\text{ceil}(1/8) \times 2 = 2$  |
| 2     | $(1 \bmod 4) \times 2 + 1 = 3$  | $\text{ceil}(2/8) \times 2 = 2$  |
| 5     | $(4 \bmod 4) \times 2 + 1 = 1$  | $\text{ceil}(5/8) \times 2 = 2$  |
| 8     | $(7 \bmod 4) \times 2 + 1 = 7$  | $\text{ceil}(8/8) \times 2 = 2$  |
| 9     | $(8 \bmod 4) \times 2 + 1 = 1$  | $\text{ceil}(9/8) \times 2 = 4$  |
| 16    | $(15 \bmod 4) \times 2 + 1 = 7$ | $\text{ceil}(16/8) \times 2 = 4$ |
| 17    | $(16 \bmod 4) \times 2 + 1 = 1$ | $\text{ceil}(17/8) \times 2 = 6$ |
| 32    | $(31 \bmod 4) \times 2 + 1 = 7$ | $\text{ceil}(32/8) \times 2 = 8$ |

用微信系列文章三的另一组语言把这个 32 卡例子再描述一遍 (两种等价说法, 任选顺手的记, L2) :

- **偶数组 (单轨, 连续分组)** : S2 接 GPU 1-8, S4 接 GPU 9-16, S6 接 GPU 17-24, S8 接 GPU 25-32——看"连续范围", 保持局部性, 同机 8 卡聚在同一台交换机上;
- **奇数组 (多轨, 交叉分组)** : S1 接 GPU 1/9/17/25, S3 接 GPU 2/10/18/26, S5 接 GPU 3/11/19/27, S7 接 GPU 4/12/20/28——看"相同相对索引", 把不同服务器同位置的网卡聚到一起 (这正是 ROFT 的 rail 思想)。

## 双口 NIC 不是主备，而是两种互补接入视角

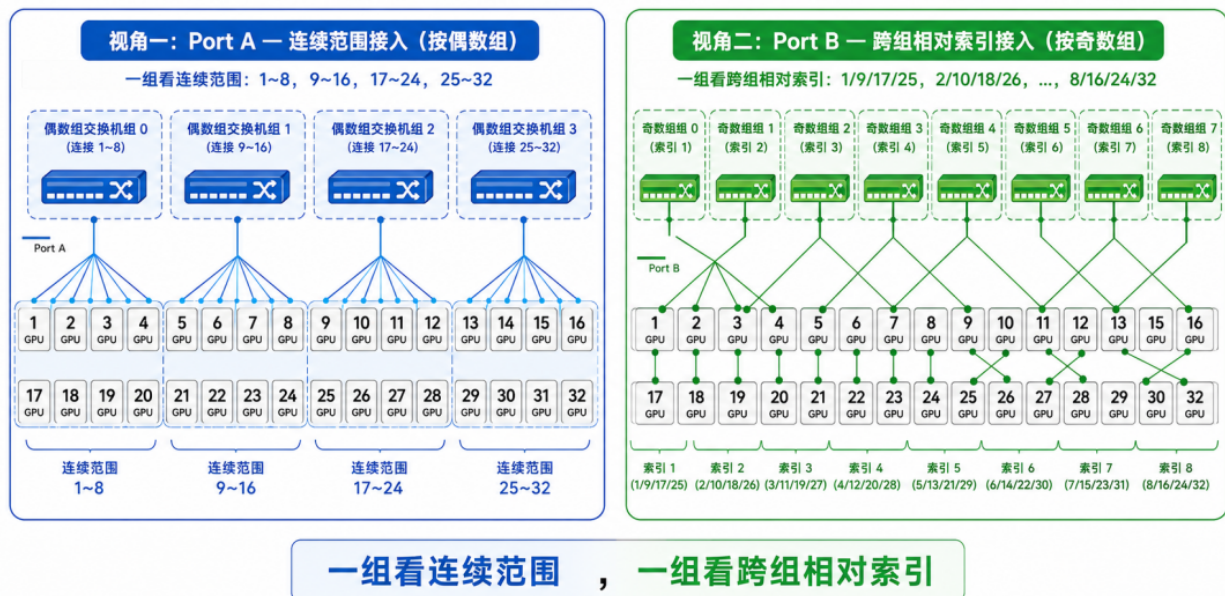


图 5-4 (图源: 微信公众号「驭驯网络」系列文章三, L2) : 双端口 NIC 的两种互补接入视角——一个端口按连续 GPU ID 范围分组 (Single-Rail Access), 另一个端口按跨组相同相对索引交叉分组 (Multi-Rail Access), 32 GPU 编号实例。

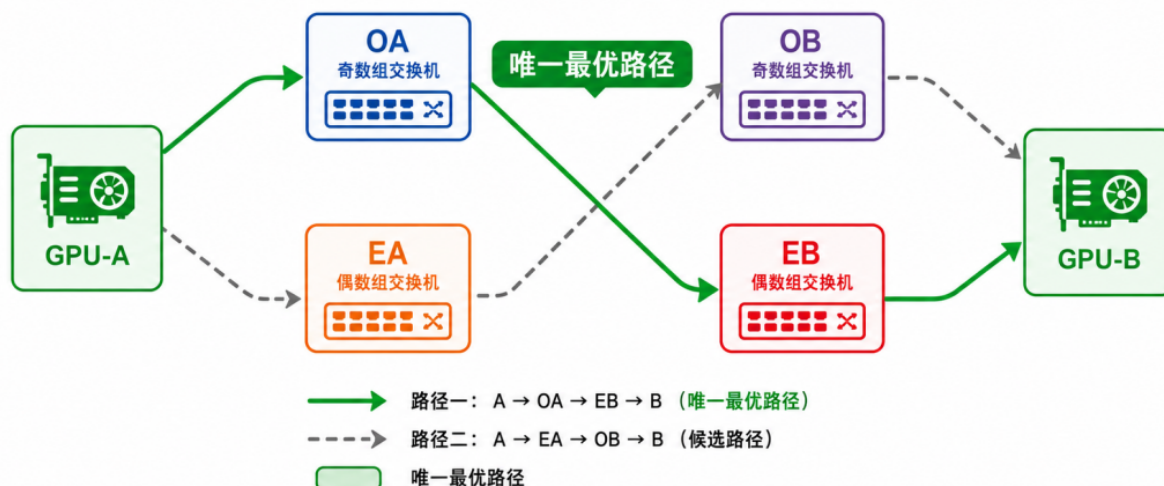
验证两个正交性:

- 同一台服务器的 8 张 GPU (如 1-8) : 端口 1 被打散到奇数交换机 1,3,5,7,1,3,5,7 (多轨), 端口 2 全部接偶数交换机 2 (单轨)——两种分组**正交**;
- 论文 (L1) 的等价描述: 奇数组  $O \square \dots O_m$  按模位置、偶数组  $E \square \dots E_m$  按连续块, 两个划分正交, 交集构成  $m \times m$  网格, 每格恰好  $k/m$  张 GPU。

为什么需要两种而不是一种？一组适配"连续的局部通信"（PD 分离的 Prefill/Decode 池按连续编号分配、同机 KV cache 块），一组适配"跨服务器的同位通信"（NCCL ring 的同位 rank）——两种互补视角叠加，再配合组间完全二部图，任意 GPU 对都有 2 跳短路径。规则训练流量与随机推理流量各取所需。

**路由规则：**GPU  $i$  到 GPU  $j$  的候选路径有两条—— $i \rightarrow O_{\{o(i)\}} (i \text{ 的奇数交换机}) \rightarrow E_{\{e(j)\}} (j \text{ 的偶数交换机}) \rightarrow j$ ，以及对偶的  $i \rightarrow E_{\{e(i)\}} \rightarrow O_{\{o(j)\}} \rightarrow j$ ，都是 2 跳。ZCube 控制器为每对 GPU **事先确定性地指定其中唯一一条**并下发转发表：稳态下没有 ECMP、没有逐包选路自由度，这就是 4.4 节说的 PDI=1。

## 一对 GPU 最多经过两台交换机



两端接入提供候选路径，ZCube 路由负责确定唯一选择

图 5-5（图源：微信公众号「驭驯网络」系列文章二，L2）：GPU-A 到 GPU-B 物理上有两条 2 跳候选路径（ $A \rightarrow OA \rightarrow EB \rightarrow B$  与  $A \rightarrow EA \rightarrow OB \rightarrow B$ ），ZCube 的路由策略为每对 GPU 提前确定唯一最优路径，而不是把选择权交给交换机的逐包哈希。

### 5.3 端口预算： $k + n/k \leq \text{端口数}$ ，交换机总数 $2n/k$

"没有 Spine 之后端口从哪来？"这是理解 ZCube 规模与成本的第一把钥匙（微信系列文章一的核心推导，L2 工程口径，与官方博客口径一致）。答案：**ZCube 不是凭空减少交换机，而是重新分配端口预算**——把原来留给 Leaf→Spine 上联的端口，改造成"GPU 接入口 + 组间二部互联口"。

记号： $n$  = GPU 数（每 GPU 一张双端口 NIC，两个端口分进两组）， $k$  = 每台交换机接入的 GPU 数， $R$  = 交换机端口数（radix）。设每组  $m$  台交换机：

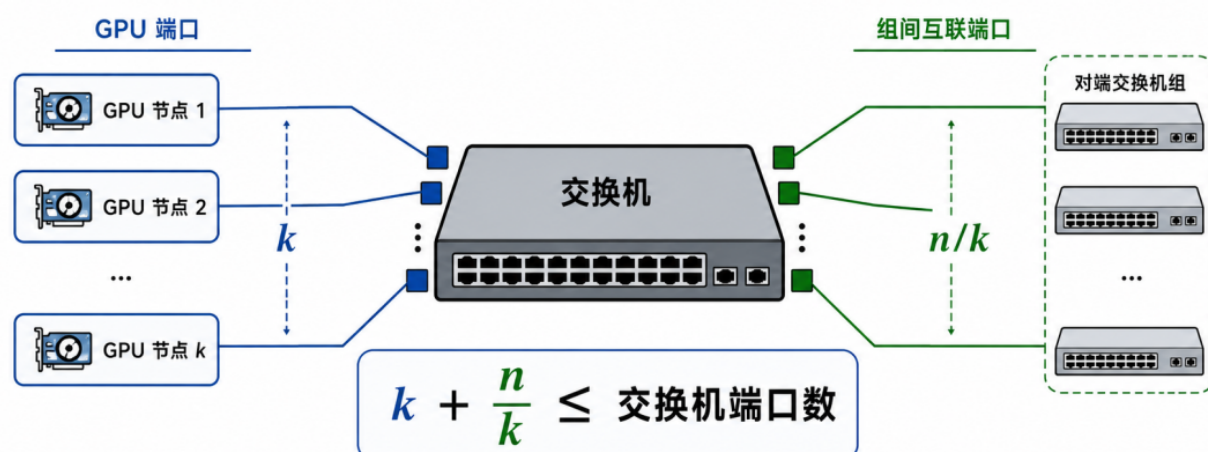
- 每组要接全部  $n$  张 GPU（每 GPU 有一个端口进该组） $\rightarrow m \cdot k = n$ ，即  $m = n/k$ ，**两组交换机总数 =  $2n/k$** ；
- 每台交换机的端口分两类： **$k$  口向下接 GPU +  $m$  口组间互联**（它要直连对侧全部  $m$  台） $\rightarrow$  端口预算约束（同速率链路下）：

$$k + m = k + n/k \leq R$$

- 给定  $R$ ，能支撑的 GPU 数  $n = k \cdot m \leq k \cdot (R - k)$ ，由均值不等式在  $k = m = R/2$  时取最大： **$n_{\max} = R^2/4$** 。



# 没有 Spine，不等于没有上联端口



## 端口从 Leaf-Spine 上联，转化为两组交换机直接互联

图 5-6（图源：微信公众号「驭驯网络」系列文章一，L2）：端口预算公式  $k + n/k \leq \text{交换机端口数}$  的图解——GPU 接入口与组间二部互联口分享同一个 radix。

代数字（ $R = 128, 51.2\text{T}, 128 \times 400\text{G}$ ）：取  $k = m = 64$ ，单层支撑  $n = 64 \times 64 = 4,096 \text{ GPU}$ （**8,192 个 400G NIC 端口**），交换机  $2n/k = 128$  台。请与 2.3 节对比：同样  $R=128$ ，两层 Clos 的无阻塞上限是  $R^2/2 = 8,192$  端点——**ZCube 单层的规模恰是两层 Clos 的一半**，这就是“砍一层”在端口算术上的真实代价；ZCube 换回的是 2 跳直径、确定性路径、以及省掉的整层 Spine 与其光模块（5.5 节算账）。

$k$  的选取是一门权衡（微信原文）： $k$  太大 → 组间互联口不够、二部图变稀； $k$  太小 → 交换机台数  $2n/k$  变多、组间互联需求爆炸。 $k = R/2$  是“接入与互联对半分”的对称甜点；实际部署还要结合链路速率、端口拆分（breakout）、上下联带宽比例微调。

**官方口径对账（重要，避免以讹传讹）**：官方博客的“一层支撑 16,384 NIC”（L1）相当于  $n = 8,192 \text{ GPU} \times 2$  端口，**超出了  $k + n/k \leq 128$  的简单整数预算**（ $n = 8,192$  要求  $k \cdot m = 8,192$  且  $k + m \leq 128$ ，无整数解，约需 182 个 400G 等效口）。实现路径是端口速率拆分与更高速率代际（如 800G 口 breakout 成  $2 \times 400\text{G}$ 、或 102.4T 芯片），微信原文同样注明“实际规模受每端口速率、Breakout 方式、上下联带宽比例影响”。至于“102.4T + 四端口 CX-8 支持 65,536 GPU”（L3 媒体口径），也请用同一个公式代一遍再决定信不信——本文只要求你掌握公式本身：拿到任何拓扑宣传数字，先算端口预算。

## 5.4 网络直径：2 跳 vs 3 跳

数一下路径上的交换机台数：ZCube 任意两 GPU 经过  $O \rightarrow E$  **两台交换机（2 跳）**；两层 Clos 是  $\text{Leaf} \rightarrow \text{Spine} \rightarrow \text{Leaf}$  **3 跳**；单层大交换网络 1 跳但规模受限。ZCube 恰好卡在中间：**比 Clos 少一跳（时延更低、buffer 更少），比单层多一跳但规模大了两个数量级**。规模口径：按 5.3 节端口预算，一层 128 台 51.2T（ $128 \times 400\text{G}$ ）交换机支撑 4,096 GPU / 8,192 NIC 端口；官方“16,384 NIC”口径依赖端口拆分/速率假设（对账见 5.3 节）；更大规模的官方路径是划分更多平面（multi-plane partitioning），扩展到数万~数十万 GPU。（另有“102.4T 交换机 + 四端口 CX-8 支持 65,536 GPU”的说法——L3 媒体转载口径，未见于英文博客正文。）

少一跳的工程意义（L2，微信文章一）：路径上每少一台交换机，就少一次交换芯片转发、一段光链路、一组队列、**一个潜在拥塞点**。拥塞点数量的下降与 TTFT 尾时延直接相关。



## 5.5 省 1/3 成本：自己算一遍验证

官方口径（L1）：同规模下 ZCube 比 Clos/ROFT 减少约 1/3 的交换机与光模块 CapEx，万卡集群省约 2.1-6.4 亿元人民币。来源就是一句话：**砍掉整个 Spine 层**。我们用第三部分的口径亲手验证。

以  $n = 4,096$  GPU（ZCube 一层 51.2T 的端口预算上限）、每 GPU 双 400G 端口（800G 总出口）为例。公平的对比基准是**双平面两层 Clos**——推理生产集群标配双平面（高可用 + 负载均衡，见第六部分），同样给每 GPU  $2 \times 400G$ ：

- **双平面两层 Clos @4,096 GPU**：每平面 4,096 NIC  $\rightarrow$  Leaf  $4,096/64 = 64$  台 + Spine 32 台 = 96 台，两平面共 **192 台**；光模块  $2 \times 4N = 32,768$  只（ $N = 4,096$ ）。
- **ZCube @4,096 GPU**：**128 台**交换机；光模块：接入链路 8,192 条（每 GPU 2 条） $\times 2$  端 = 16,384 端 + 二部图互联  $K_{\{64,64\}} = 4,096$  条  $\times 2$  端 = 8,192 端，共 **24,576 只**。

对比：

- 交换机：192  $\rightarrow$  128，**恰好省 33.3%**；
- 光模块：32,768  $\rightarrow$  24,576，省 **25%**；
- 合并 CapEx（交换机 \$4 万、模块 \$200，L2 锚点）：双平面 Clos  $\approx 192 \times 4 + 32,768 \times 0.02 = \$768$  万 + \$655 万 = **\$1,423 万**；ZCube  $\approx \$512$  万 + \$492 万 = **\$1,004 万**。省约 **29%**。

官方"33%"与交换机一项精确吻合；计入光模块后总节省约 29%（光模块单价相对交换机越高，总节省越接近 25-30% 区间）。结论："省  $\geq 1/3$ "在交换机台数上精确成立，在合并 CapEx 上大致成立（约三成），且大头确实来自砍掉的整层 Spine 及其配套光链路。（单价均为 L2 经验锚点，绝对数字仅供量级参考；相对比例是稳健的。）

## 成本下降来自角色合并，而不是链路消失

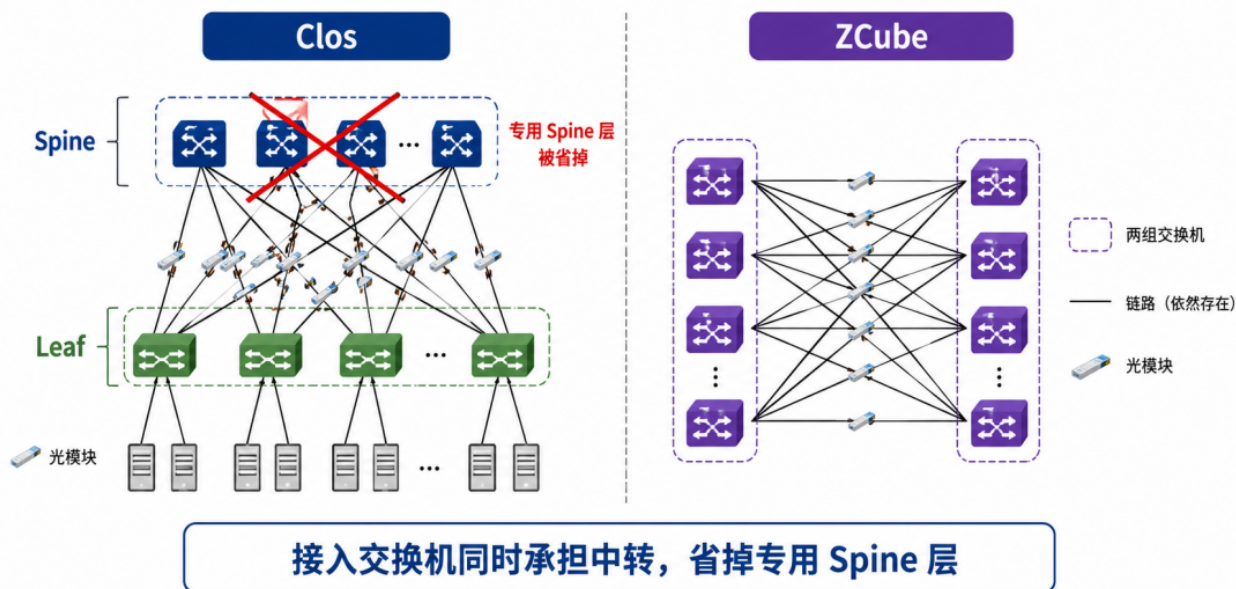


图 5-7（图源：微信公众号「驭驭网络」系列文章一，L2）：Clos vs ZCube 成本对比示意——传统两层 Clos 需要接入 Leaf、专用 Spine 与大量 Leaf-Spine 光链路；ZCube 中所有交换机同时承担 GPU 接入与跨组互联，不再需要"只中转"的一层。

再给一个超大规模口径的压力测试：对比**单平面三层 Clos @16,384 GPU**（3.4 节：640 台 + 98,304 只  $\approx \$4,526$  万），ZCube 需要 4 个 4,096 GPU 平面 = 512 台 + 98,304 只  $\approx \$2,048$  万 + \$1,966 万 = **\$4,014 万**，省约 **11%**——规模越大，ZCube 的成本优势越收敛（多平面也要花交换机和光模块），省

钱最甜的区间恰是"单平面一层装得下"的千卡~数千卡规模，这也正是官方已公开部署的区间。

万卡省 2.1~6.4 亿元人民币（L1 官方口径）：按上面 \$1,000 万级 / 4,096 GPU 的口径外推万卡（约 2.5 个平面），节省量级与官方区间一致——官方区间覆盖了不同速率代际（400G vs 800G）与不同对比基准。

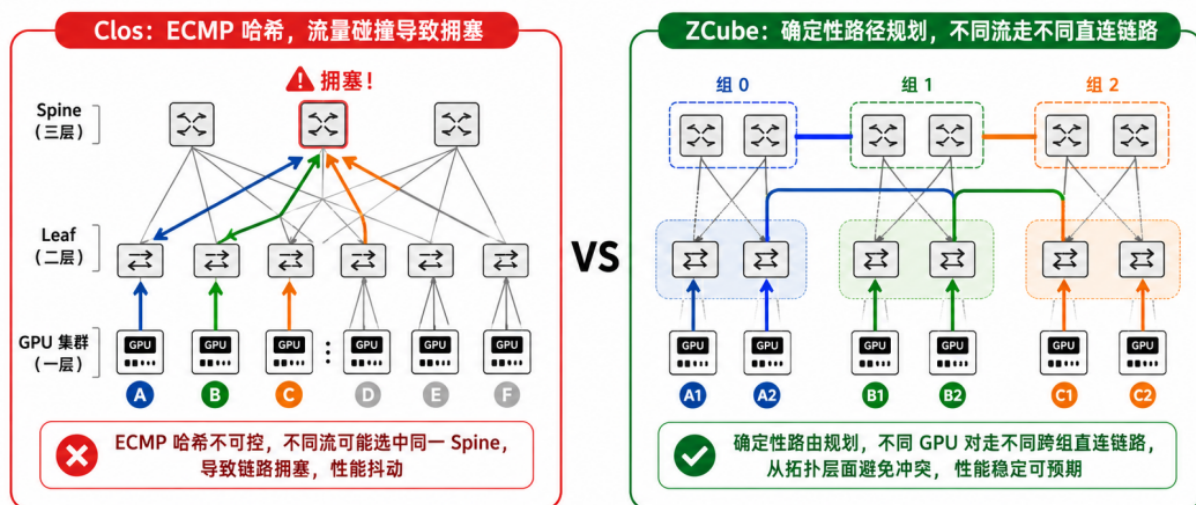
## 5.6 负载扩散定理：为什么任意流量矩阵都打不垮它

这是 ZCube 最硬核的理论结果（L1，arXiv 2607.21893），直觉版如下：

每张 GPU 同时属于两个**正交**分组：多轨组（模位置）和单轨组（连续块）。任意流量从  $i$  到  $j$ ，必走路径  $O_{\{o(i)\}} \rightarrow E_{\{e(j)\}}$ 。固定一条核心链路  $O_a \rightarrow E_b$ ，经过它的流量 = "所有多轨组为  $a$  的源"  $\times$  "所有单轨组为  $b$  的目的" 的流量之和。因为两组正交、每组 GPU 数相同（每格  $k/m$  张），**任何流量矩阵——无论怎么倾斜、怎么逐 step 变化——都会被均匀地"扩散"到  $m^2$  条核心链路上**，每条链路的负载由行/列边际约束决定，数学上不存在"结构性热点链路"。论文称之为**负载扩散定理（load-diffusion theorem）**，并给出对"连续编号热点块"（如 Prefill 池）的显式上界：每链路至多  $\text{ceil}(h/m) \cdot R_{\text{max}}$ ，比 ROFT 最坏情况好  $m$  倍。

两个推论（论文原话级别的概括）：**训练 AllReduce 与推理 KV cache 传输是同一个定理的两个推论**；以及——对 EP 的意义——MoE 逐层逐 step 变化的 all-to-all 流量矩阵，同样被这个定理覆盖，**不会产生结构性热点**（论文 §8 的明确预测；注意：属论文预测，尚待生产验证，第六部分会再强调）。

## ZCube 不靠哈希碰运气，而靠拓扑和路由规划路径



先避免可避免的冲突，再处理真正不可避免的拥塞

图 5-8（图源：微信公众号「驭驯网络」系列文章二，L2）：两种均衡哲学的对比——Clos 的 ECMP 是"发生碰撞后希望概率上均匀"（左）；ZCube 是"在拓扑和路径设计阶段避免本不该发生的碰撞"（右）。

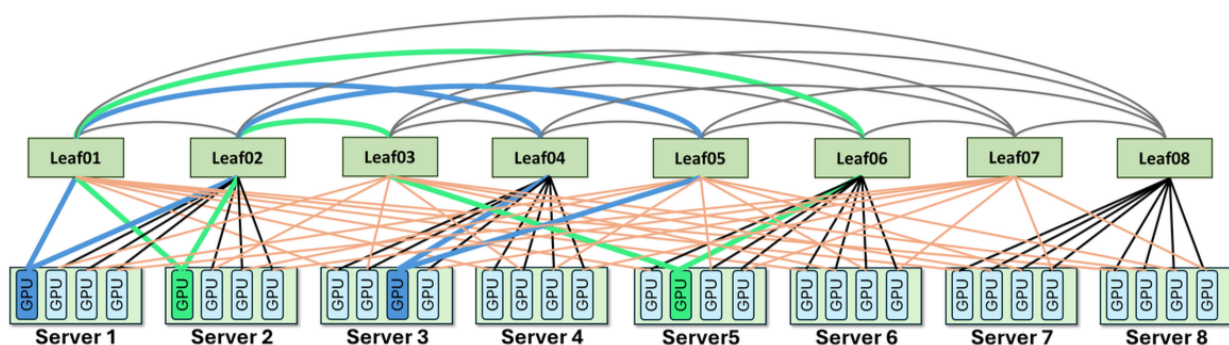


图 9（官方博客 Figure 8）：蓝色流（GPU1→某目的）与绿色流（不同源→不同目的）在 ZCube 中被分散到不同的奇偶交换机对上，没有两条流被迫共享同一条核心链路——对比图 1/图 4 中 ROFT 的"flows collide on link"。

## 5.7 对照组图：ROFT 的痛点与 ZCube 的解法

把官方博客的几张关键配图串起来看，整个论证链非常清晰。

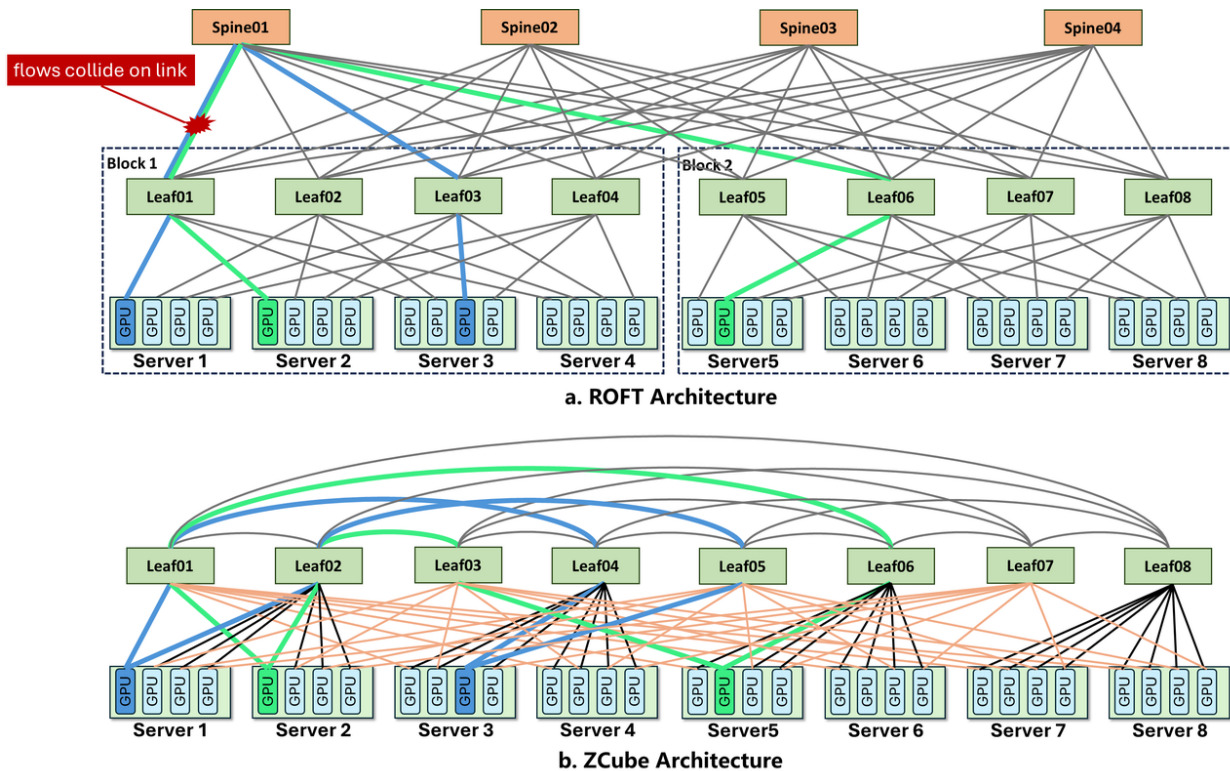


图 1（官方博客 Figure 1）：上半是 ROFT——Spine 层 + 两组 Leaf Block，蓝色与绿色两条大象流被 ECMP 哈希钉到同一条 Leaf→Spine 链路（红点 "flows collide on link"）；下半是 ZCube——没有 Spine，同样的流量被奇偶二部图结构天然分散。

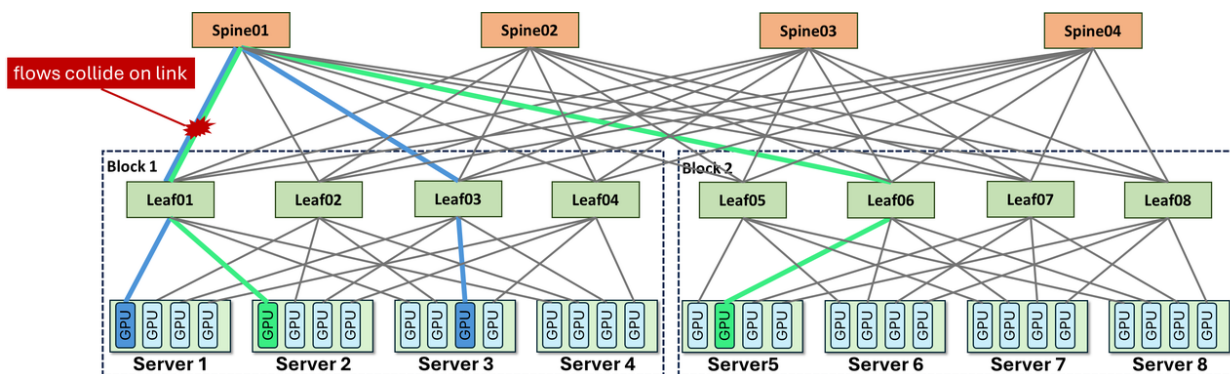


图 4（官方博客 Figure 3）：ROFT 中 Leaf 间负载不均的机理特写——流量跨 Block 时只能挤过少数 Leaf-Spine 链路。



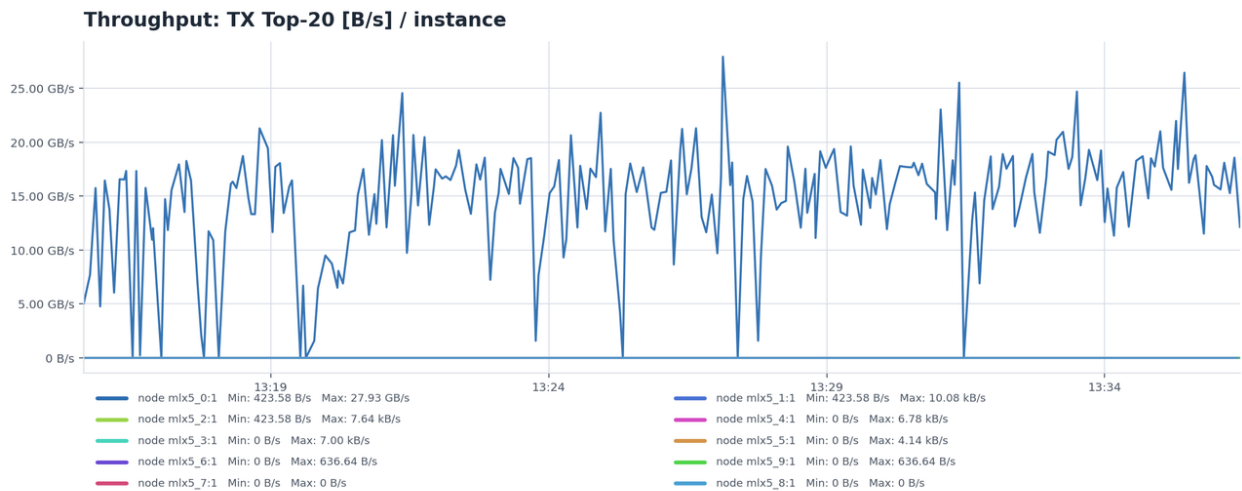


图 5（官方博客 Figure 4）：生产实测——同一台服务器的 10 条网卡实例（mlx5\_0:1 ~ mlx5\_9:1）中，只有一条在跑 10–28 GB/s 的 KV cache 流量，其余几乎为零。这就是 PD 分离流量“源-目的动态不对称”在网卡计数器上的样子：静态 rail 映射面对这种流量完全失效。

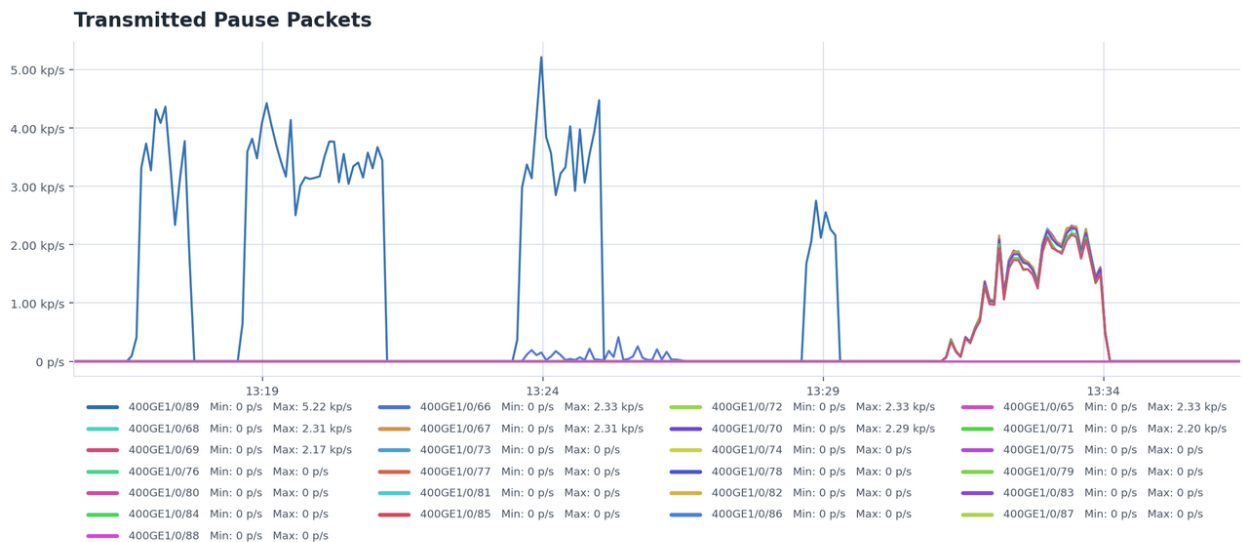


图 6（官方博客 Figure 5）：热点 Leaf 的端口频繁发出 PFC Pause 帧（峰值 >5 kp/s）。前一篇文档讲过：PFC 是无损网络的反压机制，Pause 帧频发 = 队列持续积压 = 拥塞正在向上游传导、殃及无辜流量。

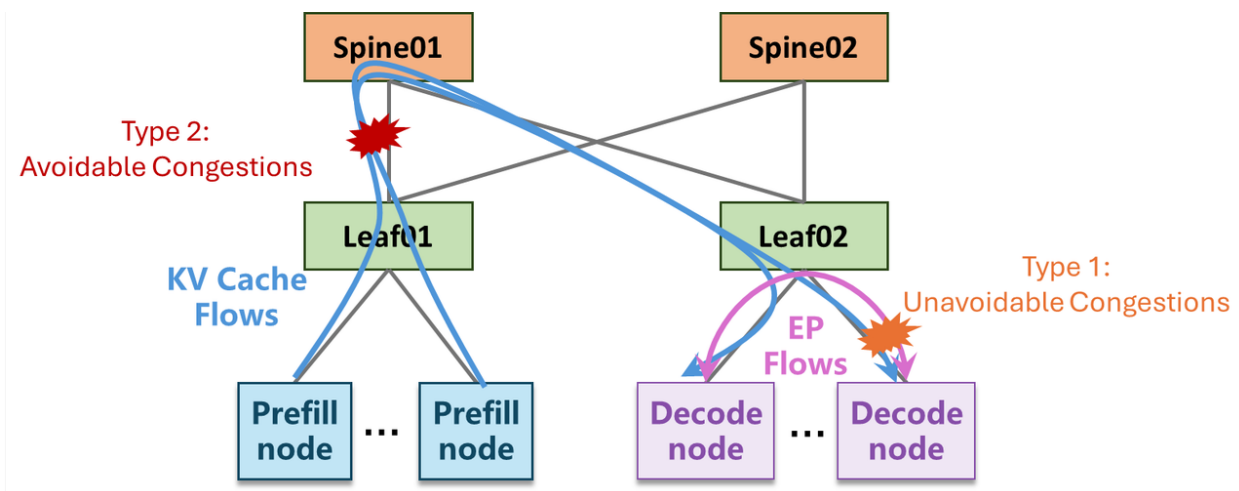




图 7（官方博客 Figure 6）：ZCube 的拥塞分类学——**Type 1 不可避免拥塞**（多条 EP flow 同时打到同一 Decode 节点，最后一跳竞争，只能靠拥塞控制/流量整形缓解，任何拓扑都有）；**Type 2 可避免拥塞**（KV cache flow 因拓扑/映射/多路径利用不均在中链路相撞）。ZCube 的声明很精确：**从架构层消除 Type 2，不声称解决 Type 1**。

## ZCube 能消除路径冲突，不能消灭最后一跳 Incast

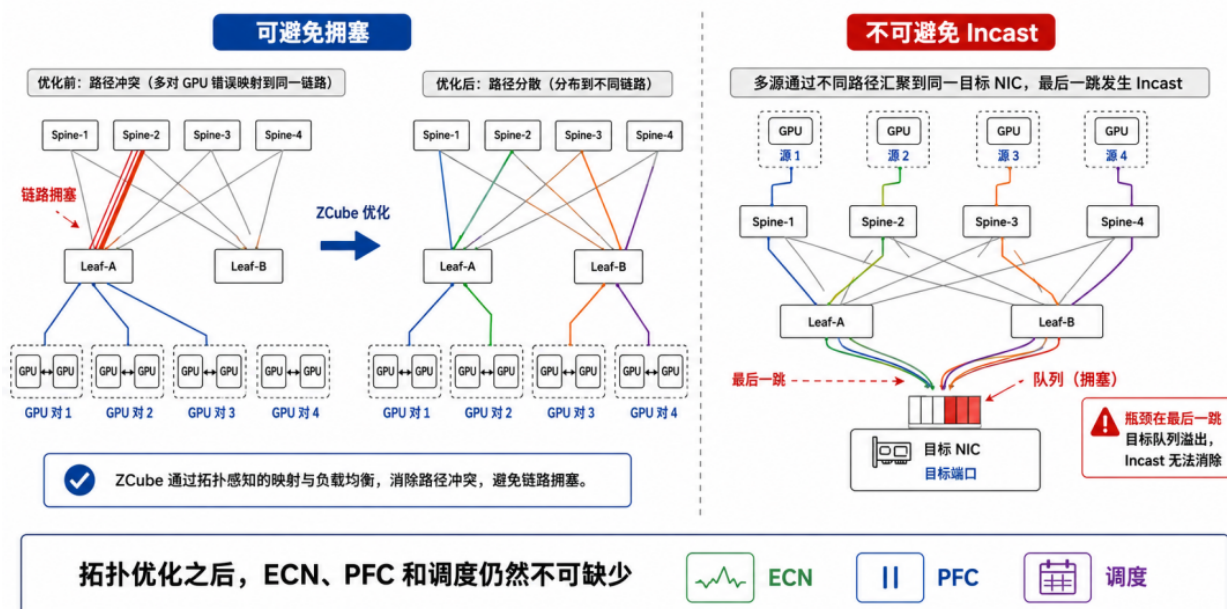


图 5-9（图源：微信公众号「驭驯网络」系列文章二，L2）：可避免拥塞（路径冲突，发生在中间链路）vs 不可避免 Incast（最后一跳汇聚）的示意——20 台 Prefill 节点同时向同一 Decode 节点发 KV cache 时，中间路径可以分散，**最后一跳的带宽无法变宽**。

必须再强调一遍 Type 1 的应对（这点常被 ZCube 的宣传淹没）：**部署 ZCube 之后，RoCE 拥塞控制全家桶一样都不能少**——ECN 标记、CNP 拥塞通知、发送端速率控制（DCQCN 类算法）、PFC 阈值与队列水线、接收端处理能力、请求调度与 KV cache 热点均衡，全部仍是日常运维的主角（机制细节见前一篇文档 stor-2 的 ECN/PFC 章节）。一句话分工：**拓扑解决"流与流在哪里相遇"，端侧拥塞控制解决"相遇之后怎么办"**。

## 5.8 生产实测数字

32 卡控制变量实验（L1）：只改网络带宽这一个变量——

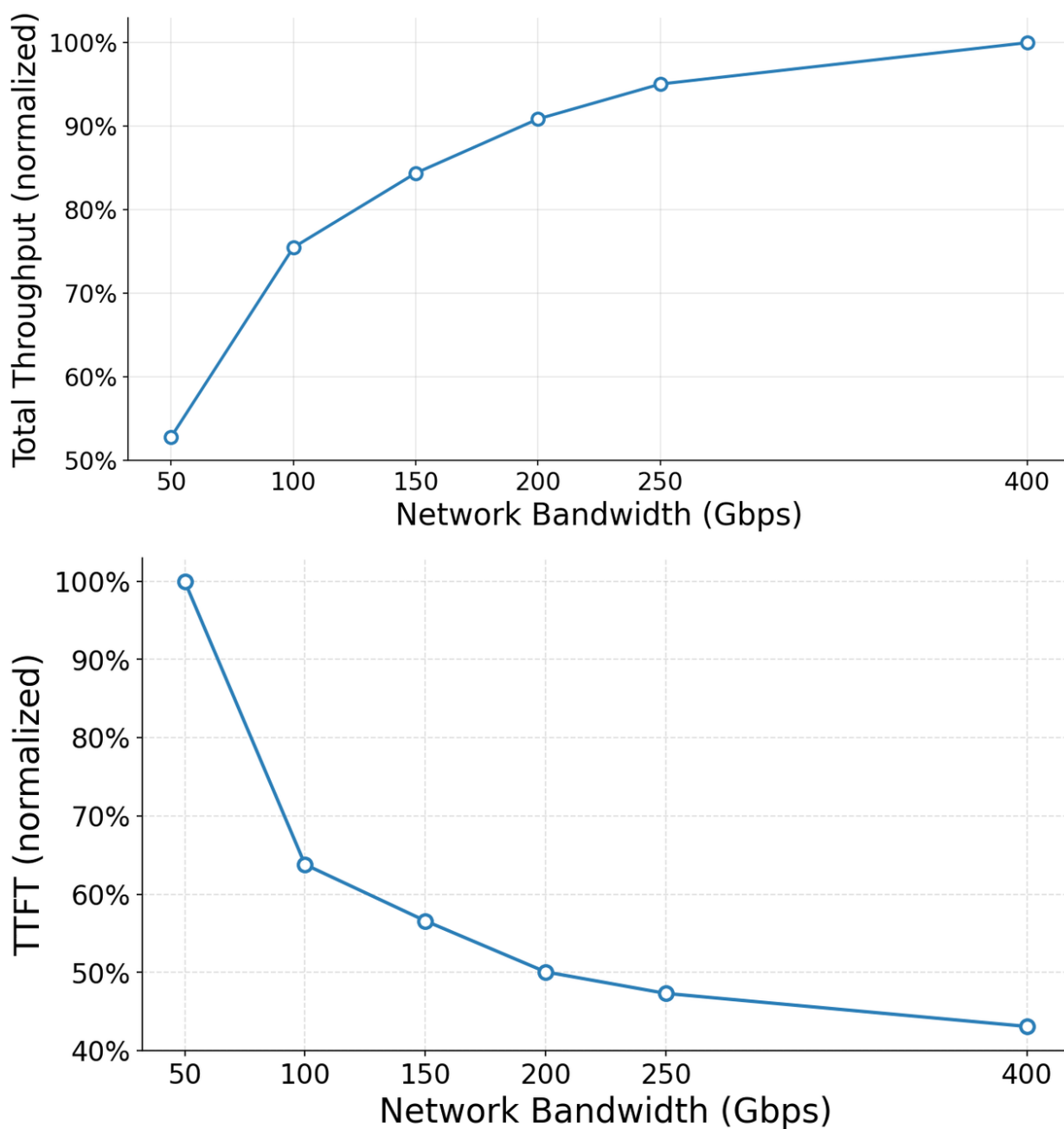


图 2/图 3（官方博客 Figure 2 两幅）：网络带宽 100Gbps → 200Gbps，推理总吞吐 +约 19%、TTFT - 约 22%；带宽从 50G 提到 400G，归一化吞吐从 ~53% 升到 100%，TTFT 降到 ~43%。这组实验同时证明了"推理性能对网络带宽高度敏感"这一前提。

千卡生产集群（GLM-5.1 coding，ROFT→ZCube 迁移，GPU/软件/应用零改动，L1）：

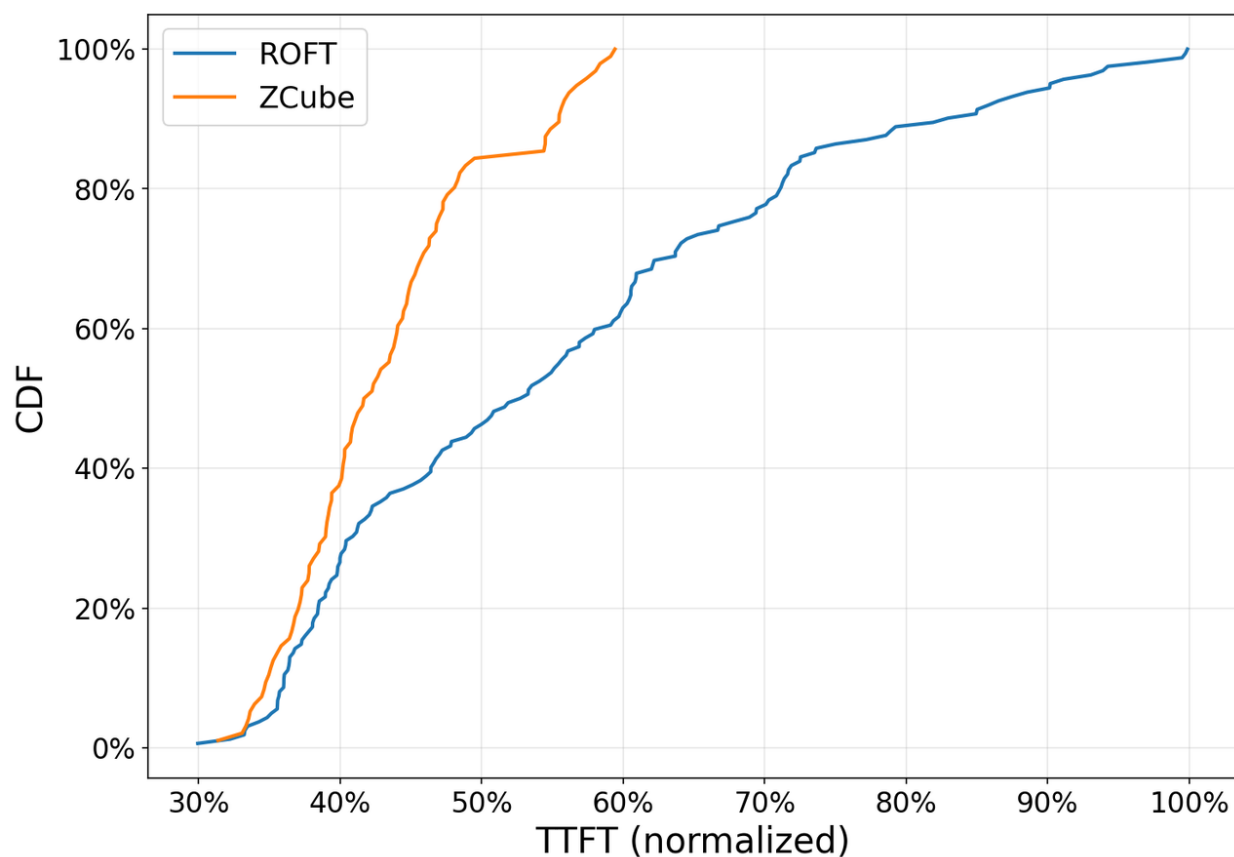
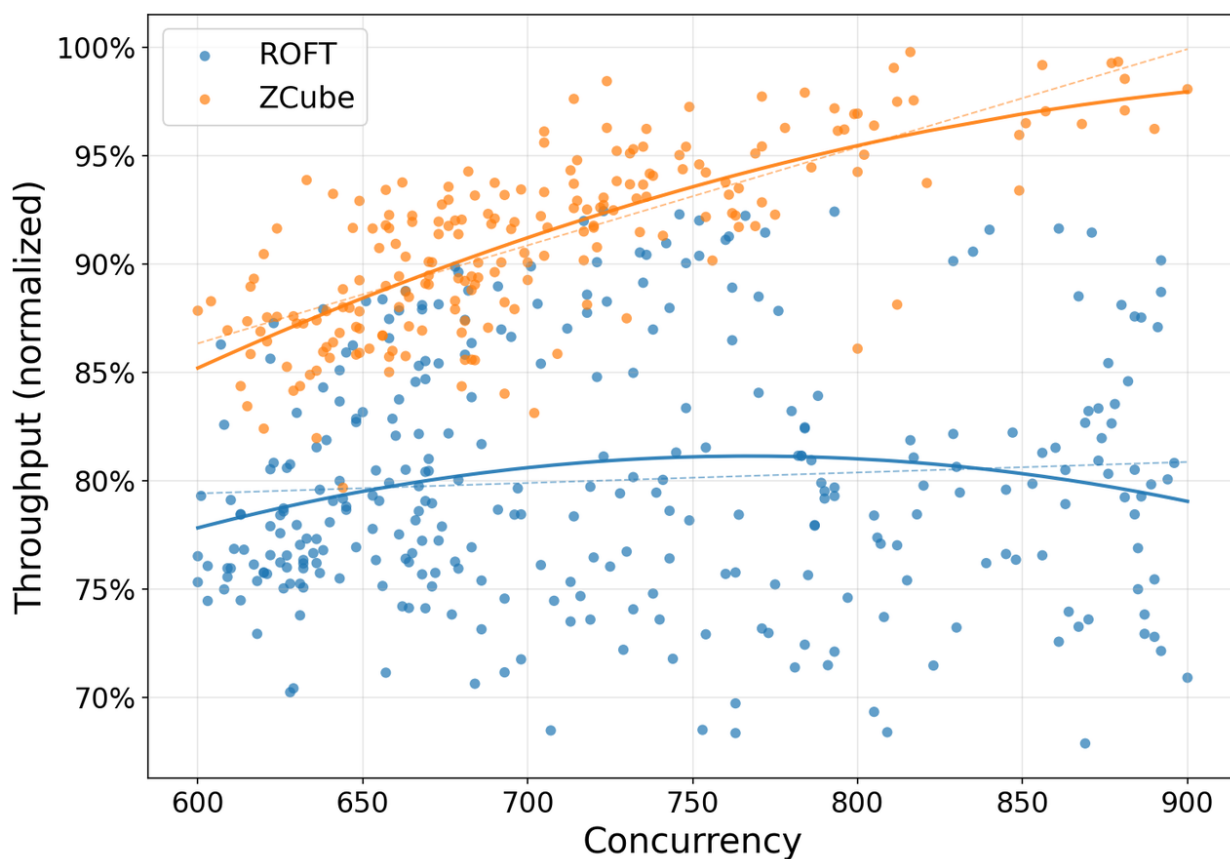


图 10/图 11（官方博客 Figure 9 两幅）：迁移后 GPU 平均推理吞吐 +15% 以上（散点图：ZCube 在高并发下维持 95%+，ROFT 跌至 ~80%）；TTFT P99 -40.6%（CDF 曲线：ZCube 的 P99 约在归一化 60% 处，ROFT 拖到 100%——尾时延几乎砍半）。注意 TTFT 的改善幅度远大于平均吞吐——这与 4.3

节的分析吻合：ZCube 消灭的是结构性热点造成的"尾巴"。

## 少一层交换机只是表面，减少结构性热点才是关键

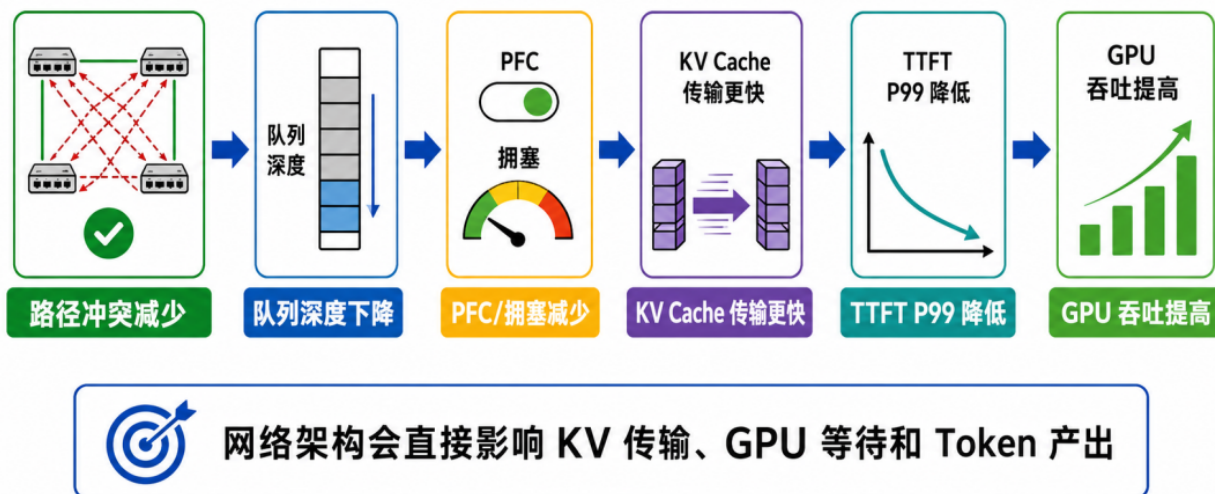


图 5-10（图源：微信公众号「驭驯网络」系列文章一，L2）：ZCube 性能收益的因果链——路径冲突减少 → 队列深度与 PFC Pause 下降 → KV cache 传输更快 → TTFT P99 改善 → 有效吞吐上升。注意链条的起点是"消除可避免拥塞"，终点指标全部来自特定生产环境的测量，不可直接外推。

### 5.9 故障域分析： $1/m^2$ 的精确计算

来自 arXiv 2607.21893 § 6.2 (L1)，这是"减少  $1/3$  故障域"传闻能找到的最硬的原始出处——虽然论文实际给出的比" $1/3$ "好得多：

- **接入链路故障**：只影响 1 张 GPU；网卡第二端口（另一组交换机）接管，应用无感知；
- **核心链路  $O_a \rightarrow E_b$  故障**：恰影响  $k^2$  对 GPU 通信，占全部  $n^2$  对的  $1/m^2$ ； $n=16,384$ 、 $k=128$ 、 $m=128$  时仅 **0.006%**；且绕行路径存在（变长为 4 跳），控制器预计算绕行；
- **交换机故障**：奇数交换机故障 → 其下  $k$  张 GPU 切到偶数侧端口；偶数交换机故障 → 控制器对受影响目的流量重路由；Yan et al. 报告单交换机故障下平均路径长度退化有界。

**代价是什么？**（好的工程分析必须问这句）Fat-Tree 的故障恢复是 ECMP 哈希的本地行为，**微秒级**；ZCube 没有等价多路径，恢复依赖集中式控制器，**毫秒级**。论文的辩护链（L1 § 6.3）：① 网络硬件年故障率  $<1\%$ ，ZCube  $>99\%$  的时间运行在最优状态，而 Fat-Tree 每天都在付"Braess 税"（静态哈希失衡）；② LLM 上层机制（请求重试、KV cache 复制、checkpoint）天然吸收毫秒级中断；③ 双平面 Clos 把冗余做在拓扑内，因此每天承担 Braess 风险；ZCube 把冗余挪到拓扑外（第二端口、预计算绕行、上层重试），日常零成本。

另有媒体口径（L3，冷月清谈/腾讯新闻转载）："相同链路故障率下，ZCube 的 GPU 对不可达概率比传统双平面 Clos 低 50% 以上"——此表述未见于英文博客正文与论文，引用时必须标注为媒体演绎。

### 5.10 部署与运维：复杂度没有消失，只是搬了家

ZCube 不是改个配置就能上的。微信系列文章一的一句总结非常到位（L2）：**ZCube 把运维复杂度从"层级数量"转移到"拓扑准确性和自动化控制"**。具体是三件事：



**1. 拓扑准确性。**传统 Clos 有成熟的工程模板（机柜摆放、布线、IP 分配、BGP、ECMP 验证、扩容），插错一根 Leaf-Spine 线只是损失一条等价路径。ZCube 不一样：它要求每张 GPU 的端口与交换机端口严格满足 5.2 节的数学映射， $K_{\{m,m\}}$  完全二部图的物理布线必须逐条正确——任意一根线插错（GPU 双口接入线或组间互联线）都会导致 GPU 对路径不符设计、负载均衡失效、链路异常变热、**控制器视图与物理网络不一致**。IP 规划、路由策略、交换机配置都不能照搬 Clos 模板。驭驯网络（Harnets.AI）为此配套了 **ZCube Controller**、**数据中心布局设计工具**、**布线正确性校验程序**，支持规划、布线验证、配置自动生成与批量部署（L1）。

## 物理层更扁平，工程实施却更依赖自动化

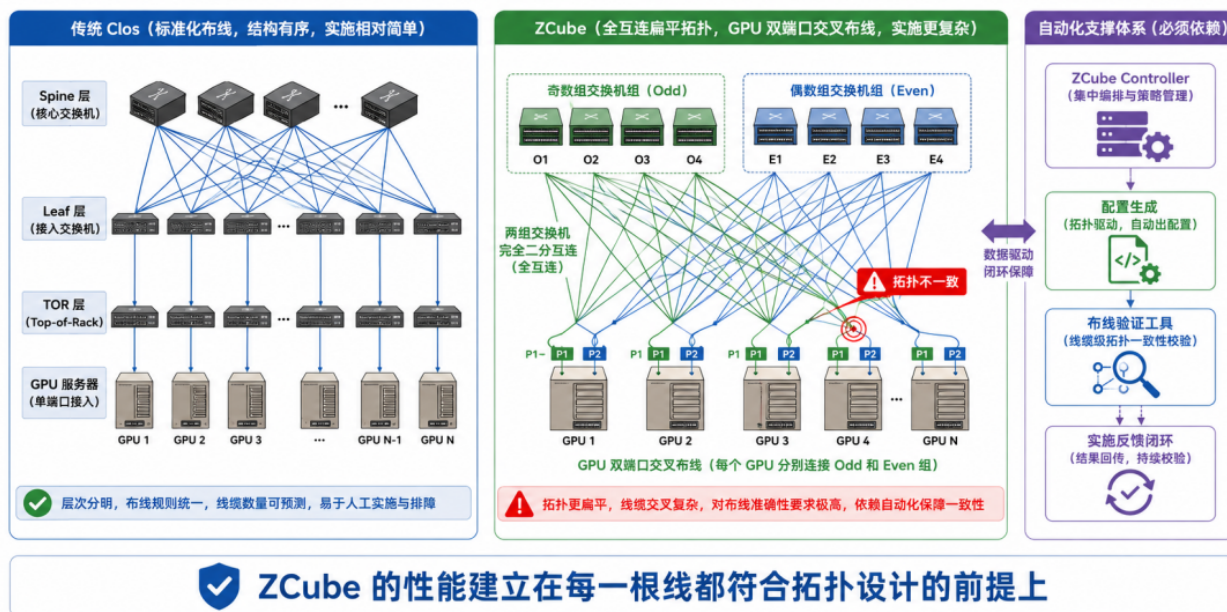


图 5-11（图源：微信公众号「驭驯网络」系列文章一，L2）：传统 Clos 的成熟布线/运维模板 vs ZCube 依赖 Controller + 布局设计工具 + 布线校验程序的自动化体系。

**2. NCCL 的 Rail 认知对齐。**ZCube 是"单 Rail + 多 Rail"混合接入，不再是 ROFT 中每张 NIC 永远对应对称隔离 rail 的世界。NCCL 依据 GPU-NIC 拓扑、可用 HCA、Rank 映射、Channel 与 rail 假设选路（NCCL\_CROSS\_NIC 的行为取决于网络是否 rail-optimized，NCCL 会尽量在节点间使用相同 NIC）。工程上必须确保：NCCL 看到的 NIC 顺序正确、GPU-NIC 亲和关系正确、NIC 选择与 ZCube 路由设计一致——不能照搬 ROFT 的 HCA 假设（L2，微信文章一；原作者亦标注此为工程推断，是否需要改 NCCL 拓扑文件/环境变量/调度策略以实际集成为准）。

## ZCube 改了物理拓扑，NCCL 的 NIC 认知也必须对齐

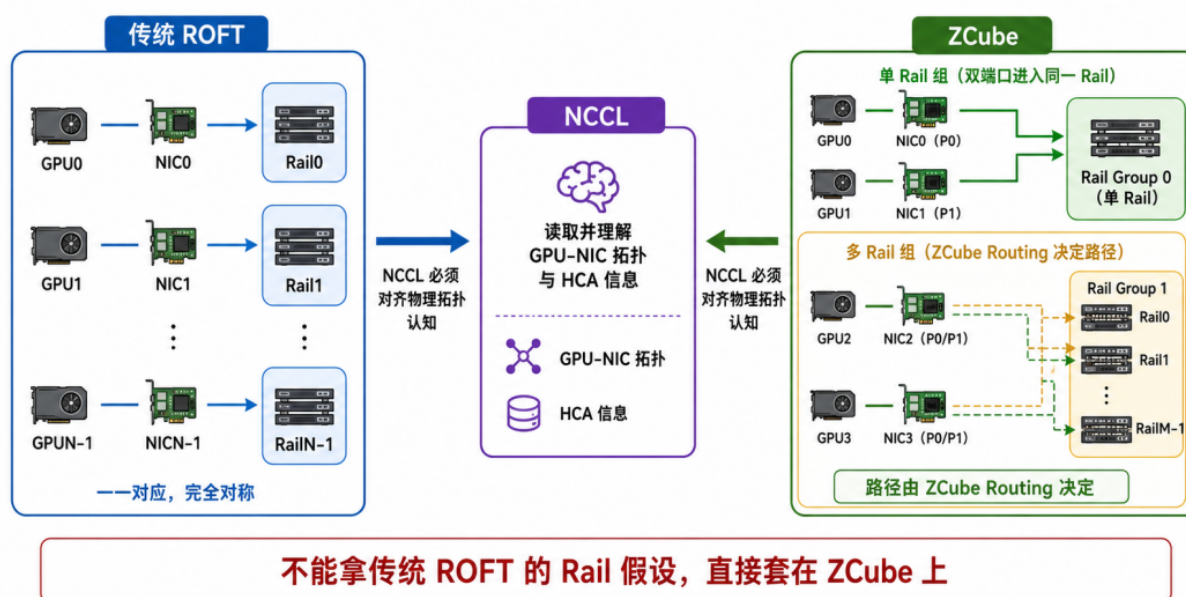
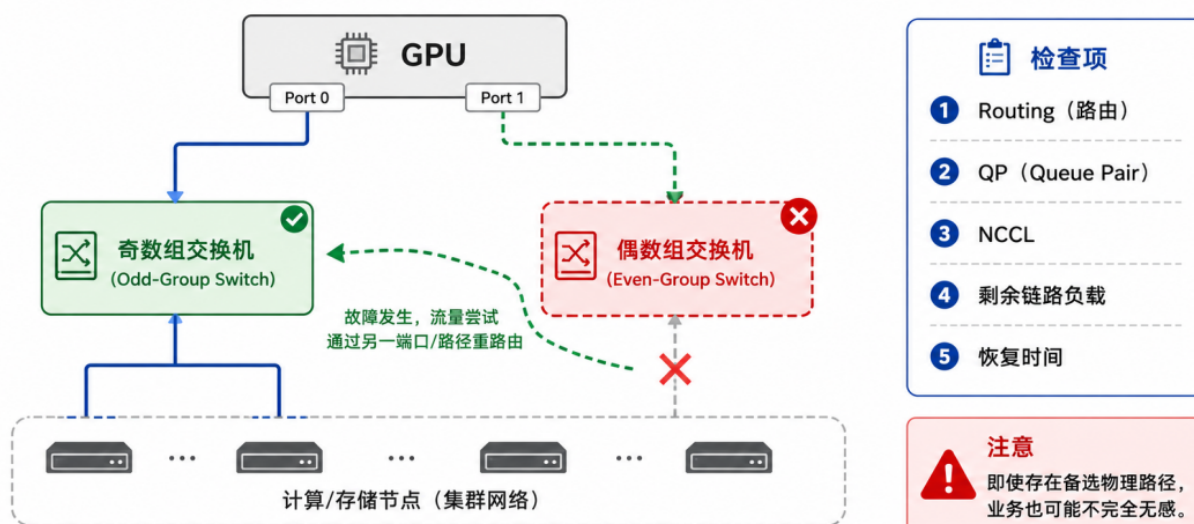


图 5-12 (图源：微信公众号「驭驯网络」系列文章一，L2)：NCCL 对传统 ROFT 与 ZCube 混合 Rail 的认知差异——集合通信库需要被“教会”认识这张新网。

**3. 容错边界。** 双端口 NIC 给每张 GPU 两个接入点，完全二部图提供丰富的备选连接——这是容错的物理基础，但不等于业务无感。故障之后还有一串问题要答：路由重算是否及时、Controller 路径更新、故障链路撤销、RoCE QP (队列对) 连续性、NCCL Communicator 是否受损、剩余链路会不会过载、双端口是否真正构成独立故障域 (同一张网卡的两个端口共享网卡本身这个单点)。

## 有备选物理路径，不等于业务一定无感



## 容错能力必须通过真实故障注入验证

图 5-13 (图源：微信公众号「驭驯网络」系列文章一，L2)：双端口故障切换的容错检查项——路由、QP、NCCL、链路负载、恢复时间，每一项都是工程问题而非拓扑性质。

集中式控制器本身是单点操作风险——论文 § 8 Limitations 明确承认 (恢复时延对比见 5.9 节，应对见 FAQ Q26)。

## 5.11 小结

- ZCube = 取消 Spine + Leaf 奇偶分组  $K_{\{m,m\}}$  完全二部图 + 每 GPU 双端口"一多轨一单轨"正交接入；每对 GPU 由控制器指定唯一最优路径，直径 2 跳。
- 端口预算： $k + n/k \leq R$ ，交换机总数  $2n/k$ ， $n_{\max} = R^2/4$ ； $R=128$  时一层 128 台支撑 4,096 GPU (8,192 NIC 端口)；官方"16,384 NIC"口径依赖端口拆分/速率假设；更大规模走多平面划分。
- "省 1/3"的来源是砍掉整层 Spine 及其光模块：对双平面两层 Clos @4,096 GPU，交换机精确省 33.3%、合并 CapEx 省约 29%；规模上万卡后优势收敛 (~11%)，最甜区间是单平面装得下的规模。
- 负载扩散定理：正交双分组对任意流量矩阵均衡核心链路负载，PD 不对称流量与 EP all-to-all 都被覆盖（EP 部分为论文预测待验证）。
- 实测：吞吐 +15%、TTFT P99 -40.6%、带宽翻倍吞吐 +19%（L1，千卡生产 + 32 卡控制变量）。
- 拥塞分类：ZCube 只消除 Type 2 可避免拥塞；Type 1 最后一跳 Incast 仍需 ECN/CNP/PFC/流量整形与调度配合。
- 故障域：核心链路故障影响  $1/m^2$ （16k 卡 0.006%），代价是毫秒级控制器恢复 vs Clos 的微秒级 ECMP。"不可达概率低 50%+"是 L3 媒体口径。
- 运维复杂度从"层级数量"转移到"拓扑准确性 + NCCL Rail 认知对齐 + 容错边界管理"，依赖驭驯的 Controller 与工具链。

## 5.12 问答 FAQ

### Q22: ZCube 的"双平面"和 NVIDIA/阿里的双平面是一回事吗？

形似神不同。NVIDIA GB300/阿里 HPN 的双平面是"两套完整独立的 leaf-spine 网络"；ZCube 的两个"平面"是同一层交换机的奇偶两组，彼此通过  $K_{\{m,m\}}$  互联、互为对方的转发路径——冗余不在"两套网"里，而在每张网卡的两个端口里。这也是媒体稿"ZCube 无硬隔离平面"说法的含义。

### Q23: 为什么是奇偶分组，而不是前后分组？

奇偶交错只是编号技巧：让"连续块"（单轨）和"模位置"（多轨）两种分组在物理编号上正交。把奇数编号分给一组、偶数给另一组，公式  $((i-1) \bmod m) \times 2 + 1$  和  $\text{ceil}(i/k) \times 2$  就能直接用整数运算表达映射，便于布线与校验。

### Q24: 万一流量恰好全都压在同一条 $O_a \rightarrow E_b$ 链路上呢？

那意味着所有源都在多轨组 a、所有目的都在单轨组 b——但每组只有  $k/m \dots$  按负载扩散定理，这条链路的负载上界是  $\text{ceil}(h/m) \cdot R_{\max}$ ， $m$  越大摊得越薄。真正的"全部打向一张 GPU"属于 Type 1 不可避免拥塞（最后一跳），由端侧机制处理，不归拓扑管。

### Q25: ZCube 能跑训练吗？

官方定位是"面向推理、兼容训练"。负载扩散定理对 AllReduce 同样成立（"同一个定理的两个推论"），但训练对故障恢复时延更敏感（同步式训练一次中断全作业停等），毫秒级控制器恢复是否够用需要按作业规模评估；且 EP 训练的适配目前只是论文预测。

### Q26: 控制器挂了呢？

论文承认这是单点操作风险（§ 8 Limitations）。工程上控制器可做多实例热备；存量转发规则不依赖控制器在线（已下发的路径继续工作），控制器故障影响的是"故障恢复"和"拓扑变更"而非稳态转发。

### Q27: 布线复杂度到底有多夸张？

$K_{\{64,64\}} = 4,096$  条核心互联链路（4,096 GPU 规模）。作为对比，同等规模两层 Clos 的 Leaf→Spine 链路是 4,096 条——ZCube 的核心链路数并不更多；真正的难点不是数量而是"无规律的满配互联"对线缆管理、跨机柜走线、施工校验的要求，这正是驭驯布线校验程序存在的理由。



Q31: k 应该怎么选? 为什么  $k = R/2$  是甜点?

由 5.3 节  $n = k \cdot (R - k)$ : 接入口与互联口分享同一个 radix, 乘积在  $k = R/2$  时最大 (均值不等式)。偏离对称点的代价:  $k$  偏大  $\rightarrow$  组数  $m$  少、二部图稀疏、故障摊薄变粗;  $k$  偏小  $\rightarrow$  交换机台数  $2n/k$  上升、组间互联需求爆炸。实际部署再按链路速率与 breakout 方式微调。

Q32: 完全二部图比全 Mesh 省在哪?

省掉组内链路。ZCube 的跨交换机流量永远是 "奇 $\leftrightarrow$ 偶", 组内 (奇-奇、偶-偶) 本来就没有直连需求, 全 Mesh 的那些链路是纯浪费。8 台例: 16 vs 28 条;  $m+m$  台的一般情形:  $m^2$  vs  $m(2m-1)$ ,  $m=64$  时 4,096 vs 8,128, 省约一半——且不损失任意跨组对一跳可达。

Q33: NCCL 需要为 ZCube 改动吗?

公开资料的口径是 "对齐而非重写" (L2, 微信文章一): 保证 NCCL 看到的 NIC 顺序、GPU-NIC 亲和关系、NCCL\_CROSS\_NIC 行为与 ZCube 的路由设计一致; 是否需改拓扑文件/环境变量以实际集成为准。官方称迁移 "软件栈零改动" (L1) ——两个口径的差异在于, "对齐" 被官方算作了部署工作而非软件改动。

第六部分 批判性评估与横向对比

6.1 五条路线的取舍一览

| 方案               | 核心理想                  | 成本                   | 直径     | 容错模型            | 最适合                  | 主要风险                |
|------------------|-----------------------|----------------------|--------|-----------------|----------------------|---------------------|
| 三层 Clos / ROFT   | 分层无阻塞 + 多路径 ECMP      | 最高                   | 3-5 跳  | 微秒 ECMP 本地恢复    | 训练 (同步 all-reduce)   | 哈希极化、Braess 税、成本超线性 |
| Dragonfly+       | 分组直连、省整层              | ~3 层 Clos 的 66% (L2) | 3-5 跳  | 自适应路由           | HPC、超大规模规则流量         | 路由复杂、全局链路割接面大       |
| 阿里 HPN 双平面       | 双平面物理隔离 + 端网协同        | 双份网络                 | 两层 3 跳 | 平面级隔离           | 通用云场景                | 双份 CapEx、依赖自研端侧栈    |
| NVIDIA GB300 双平面 | 双平面 + rail 优化 (前一篇文档) | 双份网络                 | 3 跳    | 平面级隔离 + CX-8 双接 | NVL72 生态训练/推理        | 绑定 NVIDIA 全栈        |
| ZCube            | 扁平二部图 + 正交双分组         | 比 Clos 省 1/3 (L1)    | 2 跳    | 毫秒控制器 + 第二端口    | PD 分离推理 (实测)、EP (预测) | 控制器依赖、布线/运维门槛、生态单一  |

6.2 ZCube 的局限与开放问题 (必须诚实列出的部分)

- 1. **控制器恢复速度**: 毫秒级恢复对推理 (请求级重试可吸收) 友好, 对大规模同步训练 (一次中断全员停等 + checkpoint 回滚) 是否够用缺乏生产证据;
- 2. **超大规模布线复杂度**:  $K_{\{m,m\}}$  满配互联对机房布局、线缆管理、施工校验要求极高, 依赖驭驯的专用工具链——这同时意味着**供应链/服务商锁定**;
- 3. **标准性**: ZCube 是私有架构, 没有 OCP 之类的开放规范背书; 对比 OpenAI 同期联合 NVIDIA/AMD/Intel/Microsoft/Broadcom 发布的 MRC 协议 (在现有多平面两层以太网上用多路径并发 + 智能路由实现微秒级绕行, 经 OCP 开放, 已部署于最大规模 GB200 集群), 两者代表 "**拓扑层重构 (重画路网) vs 协议层优化 (优化红绿灯)**" 的路线之争 (L3 社区讨论, 对比框架本身成立);



4. **Type 1 拥塞照单全收**：ZCube 只消除结构性热点（Type 2 可避免拥塞），最后一跳 incast 是物理上限问题，任何拓扑都解决不了——部署后 **ECN 标记、CNP 通知、发送端限速（DCQCN 类）、PFC 阈值、队列水线、请求调度** 一样都不能少（机制见 stor-2 的 ECN/PFC 章节）。"上了 ZCube 就可以放松拥塞控制"是危险的误读；
5. **EP 适配是预测不是实测**：论文 § 8 明确标注为预测，尚无公开生产数据；"ZCube 特别适合 EP"的社区评论目前建立在理论论证上；
6. **规模上限依赖多平面划分**：按端口预算公式，一层 51.2T 交换机支撑 4,096 GPU（8,192 NIC 端口）；官方"16,384 NIC"口径依赖端口拆分假设（5.3 节对账）；十万卡级需要多平面，平面间互联的设计细节公开资料未充分展开。"65,536 GPU"说法为 L3 媒体口径；
7. **单点披露**：目前生产案例只有智谱一家、推理场景一种，第三方复现与跨 GPU 生态（昇腾/寒武纪混合部署虽被社区提及但无公开实测）有待观察。

6.3 "省 1/3 成本、减 1/3 故障域"：来源分级表

| 说法                                  | 出处                         | 层级       | 评估                                                       |
|-------------------------------------|----------------------------|----------|----------------------------------------------------------|
| 交换机+光模块 CapEx ↓ 33%                 | 官方博客（z.ai/blog/zcube 及中文版） | L1       | 算术可复现（5.5 节）：交换机项精确省 33.3%，合并 CapEx 约 29%，成立             |
| 万卡省 2.1–6.4 亿元                      | 官方博客                       | L1       | 区间宽（覆盖速率代际/基准差异），量级合理                                    |
| 推理吞吐 +15%、TTFT P99 –40.6%           | 官方博客（GLM-5.1 coding 千卡集群）  | L1       | 单一客户自报数据，无第三方复现                                          |
| 核心链路故障影响 1/m²（16k 时 0.006%）         | arXiv 2607.21893 § 6.2     | L1       | 有精确推导，最硬的一条                                              |
| 端口预算 $k + n/k \leq R$ 、交换机总数 $2n/k$ | 微信系列文章一（与官方口径一致）           | L2       | 算术可自验（5.3 节）；官方 16,384 NIC 口径超出简单预算，依赖端口拆分假设             |
| 运维复杂度转移到拓扑准确性/NCCL 对齐               | 微信系列文章一                    | L2       | 工程解读，与官方披露的 Controller/工具链一致                             |
| "减少 1/3 故障域"                        | 社区流传的简化概括                  | L3       | 原文无此表述；实际论文数据（0.006%）远好于"1/3"，传闻反而是低估，但 <b>作为说法它查无出处</b> |
| 不可达概率比双平面 Clos 低 50%+               | 冷月清谈/腾讯新闻转载                | L3       | 未见于博客正文/论文                                               |
| 102.4T + 四端口 CX-8 支持 65,536 GPU     | 媒体转载                       | L3       | 未见于英文博客正文                                                |
| MoE all-to-all 无结构性热点               | arXiv 2607.21893 § 8       | L1（但为预测） | 理论成立，待生产验证                                               |

教训："省 1/3 成本"有 L1 支撑，"减 1/3 故障域"是查无出处的二手概括——虽然它碰巧方向正确且保守。评估任何架构宣传时，请复用这个分级表的方法。

6.4 选型决策建议

按场景给结论（综合本文全部计算与对比）：

- **大规模同步训练（万卡+，all-reduce 主导）**：成熟稳妥选 rail-optimized 三层 Clos（或 NVIDIA/腾讯星脉式多轨方案）；接受新架构风险且故障恢复要求可验证时，ZCube/多平面值得试点。
- **PD 分离推理生产集群**：ZCube 是目前唯一有公开生产实测的"推理专用拓扑"，TTFT P99 -40.6% 对服务 SLA 是硬收益；阿里 HPN 双平面 + 端网协同是自研能力强的云厂商的替代路线。
- **Wide-EP / MoE 重度场景**：理论上有三条路——ZCube（拓扑层，预测待验证）、自适应路由/端网协同（百度百舸、腾讯 GOR 路线）、MRC（协议层，训练已验证）。建议小规模 A/B 实测后再下重注。
- **时延极度敏感、规模 ≤ 万芯片**：关注 3D torus（TPU 路线）但接受其生态绑定；跳数 2 vs 3 的差异在 TTFT 上真实可见。
- **预算极端受限的小集群（千卡内）**：rail-only 或两层 Clos 超订即可，不要为拓扑先进性付学费。

### 6.5 小结

- 没有拓扑在所有维度最优：Clos 买通用与容错、Dragonfly+ 买成本、双平面买隔离、ZCube 买"直径 + 成本 + 抗结构性热点"，付出控制器依赖与生态锁定。
- ZCube 的三条硬数据（33% CapEx、吞吐 +15%、TTFT P99 -40.6%）有 L1 支撑但均来自单客户自报；"减 1/3 故障域"是 L3 概括，原文数据其实更好；"适合 EP"是 L1 论文预测、待验证。
- ZCube 的能力边界同样清晰：解决不了最后一跳 incast，ECN/PFC/CNP 拥塞控制全家桶仍是日常运维主角。
- 路线之争的完整图景：Clos+MRC（协议层优化）vs ZCube（拓扑层重构）vs Amazon RNG（拟随机图 + 更聪明的分布式路由）——"多路径教义"正在 AI 流量面前被三条相反路线同时攻击。

### 6.6 问答 FAQ

**Q28：如果明天要我拍板一个 8,192 卡推理集群，选 ZCube 吗？**

满足三个条件就选：① 以 PD 分离推理为主负载；② 能接受驭驯工具链与控制器运维模式；③ 有容量做多平面扩展预案（注意按端口预算，8,192 卡需要 2 个 4,096 平面或端口拆分方案，见 5.3 节）。否则选双平面两层 Clos（HPN 式），贵约 30-40% 但生态成熟。

**Q29：ZCube 和 MRC 能同时用吗？**

理论上 MRC 是协议层技术（多路径并发 + 微秒绕行），ZCube 是拓扑层（无多路径可选）——两者设计前提冲突，更可能作为不同代际/不同场景的替代方案存在，而非叠加。

**Q30："光模块占成本一半以上"这个经验数我怎么自己验证？**

用第三部分的方法：数清链路条数 × 2 × 单只价格，对比交换机台数 × 整机价，除一下即得。对任意具体报价单，这个算法 10 分钟就能给出占比——这正是本文希望你带走的能力。

### 附录 A 术语表（中英对照）

| # | 术语   | 英文          | 一句话解释                           |
|---|------|-------------|---------------------------------|
| 1 | 交换机  | Switch      | 基于 ASIC 硬件查表转发数据包的设备            |
| 2 | 交换芯片 | Switch ASIC | 交换机的核心转发芯片（Tomahawk/Spectrum 等） |
| 3 | 端口基数 | Radix       | 一台交换机的端口数量                      |

| #  | 术语            | 英文                          | 一句话解释                        |
|----|---------------|-----------------------------|------------------------------|
| 4  | 交换容量          | Switching capacity          | 端口数 × 端口速率（如 128×400G=51.2T） |
| 5  | 缓存            | Buffer                      | 端口侧吸收突发的共享缓存                 |
| 6  | 等价多路径         | ECMP                        | 按五元组哈希把流静态分配到等价路径            |
| 7  | 哈希极化          | Hash polarization           | 多条大象流被哈希到同一路径的失衡现象           |
| 8  | 优先级流控         | PFC (Priority Flow Control) | 无损以太网逐跳反压机制                  |
| 9  | 显式拥塞通知        | ECN                         | 交换机标记拥塞让端侧降速的机制              |
| 10 | 基于融合以太网的 RDMA | RoCEv2                      | 在 UDP/IP 以太网上跑 RDMA 的协议      |
| 11 | 无损网络          | Lossless network            | 靠 PFC 等机制保证不丢包的 网络           |
| 12 | 光模块           | Optical transceiver         | 端口上的电-光转换模块                  |
| 13 | 短距多模光模块       | SR (Short Reach)            | 多模光纤 ~100m                   |
| 14 | 数据中心光模块       | DR                          | 单模 500m，数据中心主力               |
| 15 | 线性直驱光模块       | LPO                         | 去掉模块内 DSP，省电 30%             |
| 16 | 共封装光学         | CPO                         | 光引擎与交换 ASIC 同封装              |
| 17 | 无源铜缆          | DAC                         | 无芯片铜缆，~3m 内                  |
| 18 | 有源铜缆          | ACC                         | 带 redriver 的铜缆，~4-5m         |
| 19 | 有源电缆          | AEC                         | 带 retimer 的电缆，~5-7m          |
| 20 | 有源光缆          | AOC                         | 两端电-光转换的成品光缆，30m+            |
| 21 | 轨道            | Rail                        | 所有服务器第 i 号网卡构成的逻辑通信组         |
| 22 | 平面            | Plane                       | 交换网络的一套独立物理冗余拷贝              |
| 23 | 单轨接入          | Single-rail access          | 连续编号块接同一台交换机                 |
| 24 | 多轨接入          | Multi-rail access           | 按模位置分散接入不同交换机                |
| 25 | 无阻塞           | Non-blocking                | 任意端点对可同时满速通信（上下行 1:1）        |
| 26 | 收敛比/超订        | Oversubscription            | 下行:上行带宽比 >1 的省钱设计            |
| 27 | 全对分带宽         | Full bisection bandwidth    | 网络任意对分两侧带宽相等                 |
| 28 | Clos 网络       | Clos network                | 1952 年多级交换结构，leaf-spine 的祖先  |
| 29 | 胖树            | Fat-tree                    | 越往上链路越粗的 Clos 特例             |
| 30 | 叶脊架构          | Leaf-Spine                  | 两层 folded Clos 的工程形态         |

| #  | 术语         | 英文                             | 一句话解释                           |
|----|------------|--------------------------------|---------------------------------|
| 31 | 轨道优化胖树     | ROFT (Rail-Optimized Fat-Tree) | 按 rail 分组减少跨 Spine 流量的 fat-tree |
| 32 | 蜻蜓拓扑       | Dragonfly / Dragonfly+         | 分组直连 + 少量全局链路的 HPC 拓扑           |
| 33 | 三维环面       | 3D Torus                       | 节点三维度直连回绕的拓扑 (TPU/Fugaku)       |
| 34 | 超立方体       | Hypercube                      | 编号差一位直连、直径 $\log_2 N$ 的拓扑       |
| 35 | 网络直径       | Network diameter               | 任意两端点间最大跳数                      |
| 36 | 故障域        | Fault domain                   | 单一组件失效时受影响的范围                   |
| 37 | 平均无故障时间    | MTTF                           | 作业两次故障间的平均时间                    |
| 38 | 数据并行       | DP (Data Parallelism)          | 复制模型切 batch, 梯度 AllReduce       |
| 39 | 张量并行       | TP (Tensor Parallelism)        | 单层切多卡, 时延敏感, 限 NVLink 域内        |
| 40 | 流水线并行      | PP (Pipeline Parallelism)      | 按层切分, stage 间 P2P               |
| 41 | 专家并行       | EP (Expert Parallelism)        | MoE 专家分置多卡, 每层两次 all-to-all     |
| 42 | 序列并行       | SP/CP                          | 切 sequence 维的并行                 |
| 43 | 混合专家模型     | MoE                            | 每 token 只激活少数专家的稀疏模型            |
| 44 | 规约通信       | AllReduce                      | 全员求和/均值的集合通信                    |
| 45 | 全交换通信      | All-to-All                     | 每对节点互发不同数据的集合通信                 |
| 46 | 多打一网拥      | Incast                         | 多发送端同时打向同一接收端的突发拥塞              |
| 47 | PD 分离      | Prefill/Decode disaggregation  | 推理两阶段分开部署的架构                    |
| 48 | 键值缓存       | KV Cache                       | 推理中缓存的注意力键值, PD 分离下需跨节点传输       |
| 49 | 首 token 时延 | TTFT                           | 请求到首个输出 token 的时延, 尾时延敏感        |
| 50 | Braess 悖论  | Braess's paradox               | 增加路径+盲选路反而整体更差的现象               |
| 51 | 负载扩散定理     | Load-diffusion theorem         | ZCube 正交双分组均衡任意流量矩阵的定理          |
| 52 | 路径多样性指数    | PDI                            | 论文定义的路径自由度指标, ZCube=1           |
| 53 | 结构性拥塞      | Structural congestion          | 拓扑/映射层面可避免的拥塞 (Type 2)          |



| #  | 术语         | 英文                                         | 一句话解释                           |
|----|------------|--------------------------------------------|---------------------------------|
| 54 | 完全二部图      | Complete bipartite graph ( $K_{\{m,m\}}$ ) | 两组节点两两跨组直连的图                    |
| 55 | 端口预算       | Port budget                                | 交换机端口在"接端点"与"互联"之间的分配约束         |
| 56 | 端口拆分       | Breakout                                   | 一个高速口拆成多个低速口 (如 800G→2×400G)    |
| 57 | 拥塞通知报文     | CNP (Congestion Notification Packet)       | 接收端回给发送端的降速通知 (RoCE)            |
| 58 | 确定性路由      | Deterministic routing                      | 每对端点路径事先唯一确定、不逐包哈希              |
| 59 | 双端口网卡      | Dual-port NIC                              | 一张网卡两个网络端口，ZCube/双平面的物理支点       |
| 60 | 可避免/不可避免拥塞 | Avoidable / unavoidable congestion         | ZCube 的拥塞二分法：路径冲突可消除，最后一跳汇聚不可消除 |

## 附录 B FAQ 汇总

全文 FAQ 共 33 问，按章分布如下（Q1–Q6 第一部分，Q7–Q11 第二部分，Q12–Q16 第三部分，Q17–Q21 第四部分，Q22–Q27 与 Q31–Q33 第五部分，Q28–Q30 第六部分；Q31–Q33 为本次更新增补、编号续排）：

1. 交换机和家用路由器有什么区别？（§ 1.9）
2. 51.2T 交换机的"51.2T"指什么？（§ 1.9）
3. 光模块为什么那么贵？（§ 1.9）
4. DAC 和 AEC 差在哪？（§ 1.9）
5. rail 和 plane 怎么区分？（§ 1.9）
6. 为什么训练网不能用超订省钱？（§ 1.9）
7. Clos 的" $m \geq 2n-1$ "在数据中心还适用吗？（§ 2.9）
8. 为什么万卡是两层的极限？（§ 2.9）
9. Spine 为什么不直接接 GPU？（§ 2.9）
10. Dragonfly 那么便宜，GPU 云厂商为什么不用？（§ 2.9）
11. rail 内一跳是怎么做到的？（§ 2.9）
12. 为什么光模块比交换机还烧钱？（§ 3.8）
13. 三层 Clos 比两层贵多少？（§ 3.8）
14. Spine 故障为什么影响小？（§ 3.8）
15. MTTF 1.8 小时是每两小时全挂一次吗？（§ 3.8）
16. Jupiter 的 4×25% 故障域思路通用吗？（§ 3.8）
17. all-to-all 和 all-reduce 差在哪？（§ 4.6）
18. 7KB 小消息为什么可怕？（§ 4.6）

19. 热专家 incast 为什么不可避免？ (§ 4.6)
20. PD 分离不分不行吗？ (§ 4.6)
21. Braess 悖论一句话怎么记？ (§ 4.6)
22. ZCube 的双平面和 NVIDIA/阿里双平面一样吗？ (§ 5.12)
23. 为什么是奇偶分组而不是前后分组？ (§ 5.12)
24. 流量恰好全压一条  $O_a \rightarrow E_b$  链路怎么办？ (§ 5.12)
25. ZCube 能跑训练吗？ (§ 5.12)
26. 控制器挂了呢？ (§ 5.12)
27. 布线复杂度到底有多夸张？ (§ 5.12)
28. 8,192 卡推理集群选 ZCube 吗？ (§ 6.6)
29. ZCube 和 MRC 能同时用吗？ (§ 6.6)
30. "光模块占一半"怎么自己验证？ (§ 6.6)
31. k 应该怎么选？为什么  $k = R/2$  是甜点？ (§ 5.12)
32. 完全二部图比全 Mesh 省在哪？ (§ 5.12)
33. NCCL 需要为 ZCube 改动吗？ (§ 5.12)

## 附录 C 参考链接

### L1 一手资料

- ZCube 官方博客（英文）：<https://z.ai/blog/zcube>
- ZCube 官方中文版：<https://www.zhipuai.cn/zh/research/160>
- ZCube 分析论文：Fewer Paths, Better Performance: Understanding the ZCube Topology through Braess's Paradox, arXiv 2607.21893
- ZCube 原始论文：Z. Yan et al., From ATOP to ZCube, ACM SIGCOMM 2025, pp. 861–881, <https://dl.acm.org/doi/10.1145/3718958.3750503>
- Clos 原始论文：C. Clos, A Study of Non-Blocking Switching Networks, Bell System Technical Journal, 1953, 32(2):406–424
- Fat-tree 数据中心化：Al-Fares, Loukissas, Vahdat, A Scalable, Commodity Data Center Network Architecture, ACM SIGCOMM 2008
- Meta RSC 故障统计：arXiv 2410.21680
- Google Jupiter：Jupiter Evolving, ACM SIGCOMM 2022
- 阿里 HPN：Y. Qian et al., Alibaba HPN: A Data Center Network for Large Language Model Training, ACM SIGCOMM 2024, <https://rmiao.github.io/assets/pdf/sigcomm24-hpn.pdf>
- Rail-optimized / rail-only 对比图出处：Wang et al., HotI 2024, arXiv 2307.12169
- PD 分离代表性工作：Splitwise (ISCA'24)、DistServe (OSDI'24)、Mooncake (FAST'25)
- UCCL-EP / DeepEP / MegaScale-Infer (arXiv 2504.02263)：EP 通信优化相关
- Amazon RNG：arXiv 2604.15261

### L2 行业经验口径 / 工程解读

- 光模块/交换机价格锚点：hyptodevice 等行业行情整理（2026 年口径）

- 光模块市场规模：Gminsights（2026 约 \$154 亿）
- Dragonfly+ 成本测算：cloudswit.ch / Asterfusion（4096 rack、10 pod 案例）
- DAC/ACC/AEC 距离-成本表：topnetsystem.com
- 微信 ZCube 系列三篇（公众号「驭驯网络」转载「算力网络架构手记」，作者：疯狂的兔子tommy；本文按 L2 工程解读引用）：
- 《没有 Spine 的 ZCube，凭什么又省又快？》：<https://mp.weixin.qq.com/s/JJPPhreOEJSwrY17QaWN1w>
- 《ZCube：缓解智算流量选路冲突，实现网络结构化负载均衡》：<https://mp.weixin.qq.com/s/zLlq0NvP78XkuW2g8om72g>
- 《没有 Spine 交换机，ZCube 靠什么撑起 GPU 集群的底层互联？》：  
[https://mp.weixin.qq.com/s/mVlJ7GcmEN\\_tDBlrL9BlTQ](https://mp.weixin.qq.com/s/mVlJ7GcmEN_tDBlrL9BlTQ)

拓扑示意图来源（图 2-x）：Berkeley CS168 教材 Datacenter Topologies（[textbook.cs168.io](http://textbook.cs168.io)）；  
Wikimedia Commons（Three-stage clos network / Fat-tree.svg / Dragonfly-topology.svg / 2x2x2torus.svg）

Dragonfly+ 结构图出处：arXiv 2406.15097（Modeling and Analysis of Application Interference on Dragonfly+）

UB-Mesh 架构图出处：arXiv 2503.20377（学术复现论文，非华为官方）

### L3 媒体转载（引用需标注）

- 冷月清谈、腾讯新闻等 ZCube 转载稿（"不可达概率低 50%+"65,536 GPU"说法出处）
- 格隆汇/富途对 SIGCOMM 2025 评价的报道
- 网界网/观察者网关于 ZCube vs MRC 路线之争的讨论

---

本文是与《NVIDIA GPU 微架构与 NVL72 AI 工厂》（stor-2.md）同系列的第二篇。文中所有"官方口径"数据以智谱/驭驯网络公开披露为准，微信系列文章为 L2 工程解读，所有价格数字为量级估算，不构成采购依据。