计算机研究与发展
Journal of Computer Research and Development
DOI: 10.7544/issn1000-1239.202440576
61(11): 2664−2677, 2024

# 面向大模型时代的网络基础设施研究:挑战、阶段成果与展望

翟恩南　操佳敏　钱　坤　关　宇
(阿里云计算有限公司　杭州　310000)
(ennan.zhai@alibaba-inc.com)

# Towards Network Infrastructure Research for the Era of Large Language Models: Challenges, Practices, and Prospects

Zhai Ennan, Cao Jiamin, Qian Kun, and Guan Yu
(*Alibaba Cloud Computing Ltd.*, *Hangzhou* 310000)

**Abstract**　Large language models (LLMs) with hundreds of billions of parameters have brought significant technological and business transformations to today's AI and cloud services. However, there exists a fundamental difference in network pattern between LLM training and general cloud computing (e.g., Amazon EC2 Elastic compute service), leading to a variety of new challenges. These challenges mainly include load balancing difficulties due to the traffic pattern difference (Challenge 1), the impact of multi-job communication contention on GPU utilization (Challenge 2), and high sensitivity to network failures (Challenge 3). Therefore, data center network technologies designed for general cloud computing (e.g., network architecture, routing, communication scheduling, and reliability) are no longer suitable for LLM training today. This necessitates the development of new data center networks and accompanying technical solutions specifically for LLM training. We introduce Alibaba Cloud's high-performance network (HPN) and the multi-job communication scheduling approach Crux, designed to address the aforementioned challenges. HPN introduces a two-layer, dual-plane network architecture, which not only achieves high-speed interconnectivity for 15 000 GPUs within a Pod but also ensures precise routing suitable for LLM training (addressing Challenge 1). Furthermore, HPN proposes a novel dual-top-of-rack (ToR) design, replacing the traditional single ToR switch connection in data center networks and fundamentally avoiding single-point failure reliability risks (partially addressing Challenge 3). To tackle Challenge 2, Crux reduces the NP-complete problem of optimizing GPU utilization by modeling it as a communication scheduling issue related to GPU computational intensity. Crux then proposes an algorithm that prioritizes the flows of job with higher GPU computational intensity, significantly reducing multi-job communication contention and improving GPU utilization. Compared with the state-of-the-art efforts, Crux increases GPU utilization by up to 23%. Both HPN and Crux have been deployed and used in Alibaba Cloud production for over eight months and will continue to evolve and iterate. Building on this, we further envision possible research directions in the field of LLM training and inference, providing guidance for subsequent work.

**Key words**　AI infrastructure; large language model (LLMs); large models; model training; data center networks; collective communication; communication scheduling

**摘　要**　拥有千亿级别参数的大语言模型(large language model, LLM)已为今天的人工智能和云服务带来了巨大的技术和商业变革.然而,大模型训练与传统的通用云计算(例如,亚马逊EC2弹性计算服务)之间存在较多根本性的网络行为差异,从而带来了很多新的挑战,主要包括流量模式差异造成负载难均衡(挑战1)、多训练任务通信竞争影响GPU利用率(挑战2),以及对网络故障的高敏感性(挑战3)等.

收稿日期:2024-06-25;修回日期:2024-09-20

因此，为通用云计算设计的数据中心网络技术（例如，网络架构、选路方法、流量调度，以及可靠性保障方法等）已不适合今天的大模型训练，这要求专门为大模型训练设计新型的数据中心网络以及配套的技术方案. 介绍了阿里云专门为大模型训练设计的数据中心网络 HPN 以及多任务通信调度方法 Crux 解决上述 3 个挑战. HPN 通过引入了一种 2 层、双平面（dual-plane）的网络架构，不但能够在一个 Pod 内高速互联 15 000 个 GPU，还能做到适用大模型训练的精准选路（解决挑战 1）. 此外，HPN 提出了一种新型的去堆叠双 ToR（top-of-rack）设计来替代传统数据中心网络的单 ToR 交换机连接方式，根本性地避免了单点失效可靠性风险（部分解决挑战 3）. 针对挑战 2，Crux 通过对 GPU 利用率优化问题的建模与证明，将该 NP 完全问题近似成 GPU 强度相关的流量调度问题. 随后，Crux 提出了一个方法优先处理具有高 GPU 计算强度的任务流，从而极大降低了多任务的通信竞争，优化了 GPU 利用率. 与相关工作对比，Crux 可以将 GPU 利用率提高多达 23 个百分点. HPN 和 Crux 均已在阿里云生产环境规模化部署超过 8 个月，后续会持续演进迭代. 在此基础上，进一步展望了大模型训练与推理领域可能的研究方向，为后续工作提供指导性建议.

**关键词** AI 基础设施；大语言模型；大模型；模型训练；数据中心网络；集合通信；通信调度

**中图法分类号** TP393

大语言模型（large language model, LLM, 以下简称“大模型”）[1-3] 已为当今的人工智能和云服务带来了巨大变革，且该趋势在迅速蔓延. 而拥有数千亿参数的大模型的训练则强依赖配备有万卡 GPU 的大规模分布式训练集群提供算力. 主流训练框架（例如，Megatron-LM[4] 和 DeepSpeed[5]）采用不同的并行策略高效协调所有 GPU 的工作.

1）数据并行（data parallelism, DP）. 数据并行策略通过将数据集分割成小批量（minibatches），然后在多个 GPU 上同时处理这些小批量来工作. 在大模型训练中，相同的模型副本在不同 GPU 上运行，每个 GPU 处理输入数据的不同部分，但使用相同的模型参数. 之后，这些计算单元会在每一轮训练的末尾同步更新模型参数.

2）流水线并行（pipeline parallelism, PP）. 流水线并行将模型分成多个阶段，然后将这些阶段分布到不同的 GPU 上. 输入数据会按顺序通过这些阶段，就像在流水线上一样. 当一个阶段处理某个数据批次时，其他阶段可以同时处理不同批次数据. PP 可减少 GPU 之间的等待时间，从而提高效率.

3）张量并行（tensor parallelism, TP）. 张量并行将模型内部的单个计算操作（如矩阵乘法）分布到多个 GPU 上执行. 这意味着模型的不同部分可同时在不同 GPU 上执行更小规模的计算. TP 通常用于处理模型的单个组件非常大、单个 GPU 无法有效处理的情况.

以上 3 种并行策略各有优劣，根据具体任务和可用硬件等特点灵活选择，以达到提高计算效率和加快模型训练的目的. 在大模型训练中，为保持模型参数的一致性，需要在每轮迭代后同步所有 GPU 上的梯度（gradient）或参数. 这种同步机制确保了模型在每次更新后都能保持一致的状态，是实现有效并行训练的关键.

不难发现，大模型训练的“计算→同步通信→计算→同步通信”分布式协同迭代通信特征与传统的弹性、通用云计算的计算特征存在显著差异. 因此，想要为今天的大模型训练提供高效且稳定的网络基础设施服务势必会面临许多全新的挑战.

## 1 大模型训练面临的网络挑战

我们所在的阿里云基础设施网络团队在为大模型训练提供底层网络的长期支持与实践期间总结了如下 3 个关键挑战. 注意，这些挑战是大模型训练原理本身产生的问题，而非特定网络结构或场景引入的问题，并且在当前国内外各大互联网厂商进行大模型训练中均有提及[6-7]. 本文通过实际规模化的生产数据将这些挑战进行展示.

### 1.1 挑战 1：流量模式差异造成负载难均衡

大模型训练的网络流量模式与通用云计算的模式存在显著不同. 大模型训练的流量表现为：低熵性（low entropy）和流量突发性. 具体地，通用云计算会生成数百万条数据流，形成网络数据流的高熵（high entropy）. 根据我们已发表文献 [8] 的内容所示图 1 展现了通用计算中数据流是连续的，并且带宽较低，通常不会超过网卡容量的 20%. 而大模型训练则产生少量的（一个 GPU 只有几十个连接）、周期性的突发性数据流，这导致网络流量的熵值降低，同时瞬间网络利用率飙升，这是由于大模型训练的协同迭代机制造成的. 图 2 展示了在大模型训练实际场景中，最高网络利用率可达 400 Gbps，呈周期式进行.

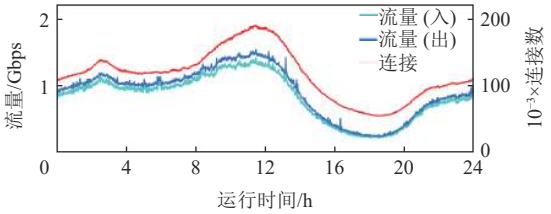



Fig. 1 Traditional cloud computing traffic pattern[8]

图 1 传统云计算流量模式[8]

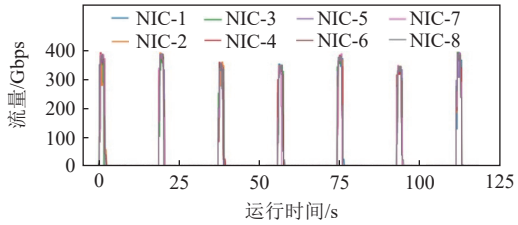


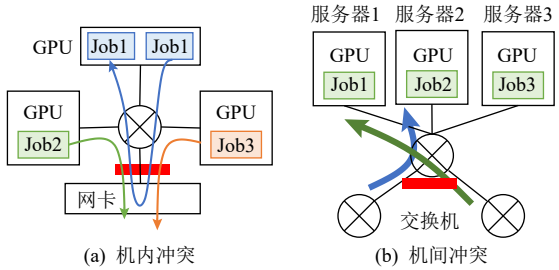

Fig. 2 NIC egress traffic pattern during production model training[8]

图 2 模型在生产环境下训练期间的网卡出方向流量[8]

大模型训练这种突发流量模式破坏了传统数据中心网络中广泛使用的等成本多路径(equal-cost multi-path, ECMP)负载均衡策略. ECMP 依赖哈希算法将流量均匀分配到等效路径上, 在通用云计算的高熵性和低利用率的网络流量模式下表现良好. 然而, 面对大模型训练中出现的少量“大象流”模式, ECMP 不再适用, 在实际应用中我们确实也遇到了多起负载不均的问题.

### 1.2 挑战 2: 多训练任务通信竞争影响 GPU 利用率

在多租场景的多个大模型训练任务场景中, 当这些任务共享相同服务器间网络转发路径或服务器内链路(如 PCIe 和 NVLINK)时, 会发生通信竞争(communication contention), 显著降低训练任务的平均完成时和 GPU 利用率. 图 3 展示了多训练任务在机内和机间产生通信竞争的情况. 例如, 在实际环境中观察到 2 个大模型训练任务在同一集群并行执行时, GPU 利用率下降了 12.8 个百分点, 这意味着一个 1 000 卡 GPU 的集群中有 128 个 GPU 被浪费, 相当于数百万美元的开支.




(a) 机内冲突 (b) 机间冲突

Fig. 3 Communication contention between multi-job

图 3 多任务间的通信竞争

传统云计算中的计算任务也存在通信竞争问题, 但是这些任务的流量特征与大模型训练不同, 更不像大模型训练这样具有多轮迭代的特点, 因此, 我们在大模型训练中无法复用传统云计算中的通信调度方法优化 GPU 利用率. 而该问题严重困扰我们的集群训练. 在阿里云生产环境的部分集群中, 我们观察到集群运行的所有任务中有 36.3% 的任务(占用 51% 的 GPU)在其生命周期中曾遭受通信竞争, 其中大部分是机间通信竞争, 即网络转发路径上的竞争.

进一步地, 在文献 [9] 中, 我们针对阿里云生产环境中的一个 GPU 集群选取了一对代表性任务并复现它们的训练过程. 这 2 个任务分别是一个 64-GPU 的 GPT-3 变体大模型和一个中等大小的 BERT 模型. GPT-3 变体分布在 2 个机架顶(top-of-rack, ToR)交换机下的 8 个主机上, 即每个交换机下有 4 个主机. BERT 模型被分配到相同的 2 个 ToR 交换机下的另外 4 个不同主机上, 即每个主机使用 4 个 GPU. 由于这 2 组主机共享相同的 ToR 交换机, 因此, 在训练过程中可能会遇到通信竞争. 如图 4 所示, 与单独执行 GPT-3 变体相比, 当同时启动 2 个模型时, GPT-3 变体的训练迭代时间增加了 11.0%, 从 1.53 s 增加到 1.70 s. 2 个模型训练的吞吐量分别下降了 9.9% 和 7.7%, 最终导致每个 GPU 的利用率下降了 9.5 个百分点.

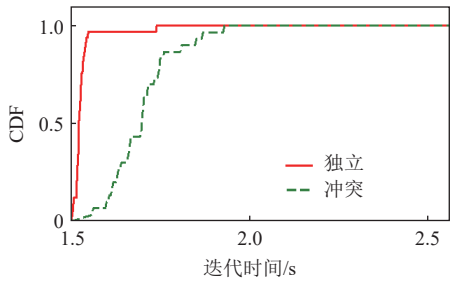



Fig. 4 Impact of communication contention on the iteration time of GPT-3 training[9]

图 4 通信竞争对 GPT-3 变体大模型训练迭代时间的影响[9]

### 1.3 挑战 3: 对网络故障的高敏感性

大模型训练本质上是一个 GPU 计算与网络通信的协同过程, 所有 GPU 协同完成一系列同步迭代. 因此, 任何 GPU 或其网络连接的异常都可能导致训练过程的减速甚至中断. 这表明, 大模型训练对故障的敏感性远高于通用云计算. 据我们的观察, ToR 交换机的单点故障对大模型训练的影响尤为严重, 因为其他层(例如, 汇聚层和核心层)的冗余设计相对充足, 而 ToR 交换机仅通过 1 条链路连接整个机架. 因

此，ToR 交换机的故障可能直接影响整个机架的 GPU. 更重要的是，大模型训练中的故障代价非常高昂. 根据我们的统计，大模型训练中的故障成本大约是通用云计算中故障成本的 20 倍.

其次，即便大模型训练通常会利用检查点（checkpoint）从故障中恢复[10]. 但生成这些检查点不仅需要大量存储空间而且开销很高. 因此，租户会选择每隔几小时才生成 1 次检查点. 根据我们的实际经验，具有代表性的客户检查点生成间隔通常为 2~4 h. 即便如此，检查点引入的开销仍然可能占到大模型训练总成本的 5% 左右. 这意味着一旦发生故障，整个训练必须回滚到几小时前重新训练. 考虑到利用 3 000 个 GPU 的训练任务的成本是每小时 2 万美元，一次故障可能导致 3 万美元的经济损失.

### 1.4 其他挑战

1.1~1.3 节描述的挑战 1~3 只是揭示了大模型训练中比较根本性区别于通用云计算的网络特征和挑战性问题. 实际中，除上述这些挑战外，大模型训练还有很多其他挑战，包括但不限于训练框架和集合通信库的 bug 等造成的网络故障和性能问题，以及大模型训练多租容器化带来的训练和推理网络性能瓶颈等问题. 本文主要关注解决上述 3 项关键挑战，因为我们认为这些挑战是大模型相比通用云计算的根本性问题.

## 2 阿里云的解决方案

上述挑战 1~3 的解决方案有很多种方式. 例如，设计新的选路或负载均衡方法解决挑战 1[11-20]，设计新的集合通信方法解决挑战 2[6,21-22]，以及设计新的监控故障定位工具解决挑战 3. 然而，我们认为这样的解决方案是烟囱式的，并非整体性的解决思路，也不方便后续针对大模型训练推理的更多新挑战进行叠加式的机制设计与增强.

我们的解决思路是首先从数据中心网络架构入手，设计一款专门适用大模型训练的数据中心网络集群架构，根本性地使得网络基础设施的整体直接适用于大模型的训练网络特征，也从根本上杜绝部分大模型训练带来的问题（例如，挑战 1 提到的 ECMP 问题以及挑战 3 的单点风险问题）. 随后，在该网络架构基础上，进行服务器-网络相结合的思路设计新技术解决架构层面无法解决的具体问题（例如，挑战 2 中提到的多任务通信竞争问题）. 这样的好处是可以确保每一层解决方案的设计都可以为后续的新能力研发提供基础，同时尽可能利用网络和服务器的信息互通协同来根本性地解决各类挑战.

根据上述思考，阿里云网络基础设施设计研发了 2 个解决方案解决第 1 节中提到的 3 个挑战：

1）我们设计建立了专门针对大模型训练的新一代数据中心网络架构 HPN（high-performance network）. HPN 通过双平面（dual-plane）设计与去堆叠双 ToR 设计分别解决了挑战 1 以及挑战 3 中的单点可靠性风险. 此外，与经典的 Clos 架构不同，HPN 仅通过接入层和汇聚层 2 层可容纳 15 000 个 GPU 的高速互联[9].

2）在 HPN 基础上，我们研制了 Crux，一款面向大模型训练多任务的通信调度方法. Crux 的目标是优化训练集群的 GPU 利用率，而在多任务大模型训练环境优化 GPU 利用率被证明是 NP 完全问题. 因此，Crux 首先将该目标规约为在给定链路上传输的所有任务的 GPU 计算强度（GPU computational intensity）之和问题，从而再结合工业界真实观察到的现象设计了根据 GPU 强度调整通信优先级的集合通信方法，达到在多任务场景下的 GPU 利用率优化[8].

上述 2 项成果主要内容来自我们已发表的文献 [8–9]，且都已经在阿里云基础设施网络中规模化部署和使用超过 8 个月. 此外，前面所提到的基础挑战均是大模型训练本身产生的普遍不存在的问题，而非特定公司或环境引入的问题. 本文在第 3 节和第 4 节分别进行文献 [8] 和文献 [9] 工作的凝练，并额外对其中的核心设计理念、思考及部署指标进行介绍. 此外，在第 5 节，我们针对更多的大模型可靠性问题进行论述，介绍我们的思路和实际规划. 最后，在第 6 节和第 7 节分享大模型训练的经验与后续可能的发展及研究方向.

## 3 面向大模型训练的数据中心架构 HPN

HPN 是阿里云为大模型训练量身定制的新一代数据中心网络架构. HPN 由前端网络和后端网络两大部分构成. 后端网络专注于承载训练任务；而前端网络则负责处理其他流量，如模型推理和存储等. 在大模型训练方面，我们重点关注 HPN 的后端网络，如图 5 所示. 每台服务器均配备了 8 个 GPU，并通过专有的高带宽机内网络（例如，NVLink[23]）实现 GPU 间的直接通信，通信速度可达双向 400~900 GBps. 为确保网络容量最大化，HPN 为每台服务器配置了 9 个网卡，每个网卡提供 2×200 Gbps 的带宽. 其中，一个网卡连接至前端网络，其余 8 个网卡则连接至后端

网络, 负责训练期间的数据传输. 这 8 个网卡中的每一个都直接连接至对应的 GPU, 形成所谓的“轨”(rail), 确保每个 GPU 拥有专用的 400 Gbps RDMA 网络吞吐能力, 总体带宽高达 3.2 Tbps. 这样的设计旨在充分利用 GPU 的 PCIe 能力(PCIe Gen5×16), 从而将网络的发送/接收容量推至极限.

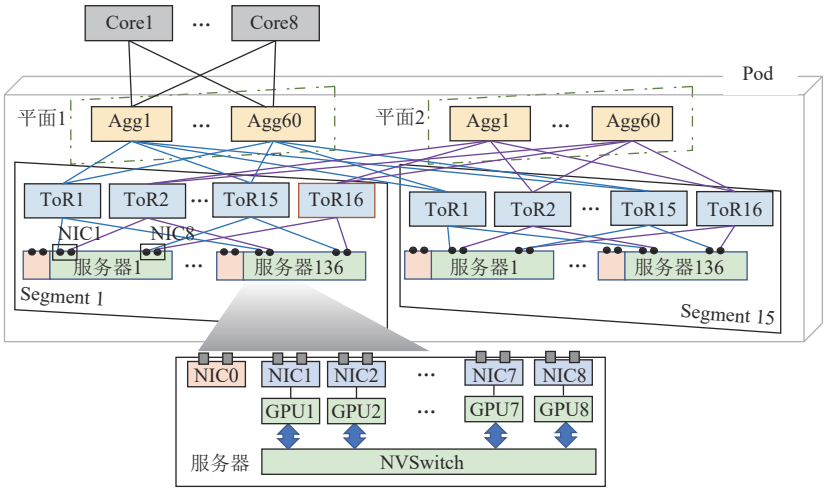



Fig. 5 HPN backend network architecture overview

图 5 HPN 后端网络架构概览

HPN 解决了哪些挑战? HPN 意在解决挑战 1 及挑战 3 中提到的单点 ToR 风险. 与传统数据中心[24–27]相比, HPN 提出了一种称为“去堆叠双 ToR”的设计, 有效规避了 ToR 相关的单点故障风险, 即根除挑战 3 中的单点风险(详见 3.1 节). 其次, 在接入层设计中, HPN 引入多轨优化网络(rail-optimized network), 使其能够在接入层容纳 1 000 个 GPU, 使大模型训练任务能够享受到卓越的直连网络性能(详见 3.2 节). 注意, Clos 架构由于其面向通用计算的设计初衷是无法做到的. HPN 在汇聚层引入双平面设计, 不仅在汇聚层避免了哈希极化(Hash polarization), 还显著减少了选路搜索空间, 为承载不同大流量的数据包选择了理想路径, 有效解决了挑战 1. 此外, 该设计允许 HPN 通过 2 层架构就能够容纳高达 15 000 个 GPU, 相比传统数据中心 Clos 架构, 减少了 1 层的需求(详见 3.3 节).

### 3.1 去堆叠双 ToR 设计

传统数据中心网络普遍采用的是单 ToR 设计, 即每个网卡的 2 个端口通过一根电缆或光纤连接到 ToR[25-26,28-29]. 这种设计主要被今天国内外各大云厂商广泛使用. 然而, 该设计容易受到交换机或链路故障的影响, 可能导致大模型训练性能的显著下降.

为解决这一问题, 双 ToR 设计应运而生. 在该设计中, 每个网卡的 2 个端口连接到不同 ToR, 这 2 个 ToR 配置相同的 IP 和 MAC 地址, 即使一个 ToR 出现故障, 另一个仍能正常工作. 关键是 2 个 ToR 使用 1 条链路连接以同步状态. 2 个 ToR 的控制平面以主从方式运作, 通过带外网络同步控制平面状态以确保正确的主 ToR 选择. 在服务器侧, 双 ToR 接入采用绑定(bond)模式[30](一个内置的 Linux 模块)实现. 特别地, 绑定模式 4(动态链路汇聚)[31] 能在 2 个端口间自动均衡负载, 并在链路或 ToR 故障时自动重新路由数据流. 这种设计的目标是最小化对服务器端的修改, 为大规模部署和升级提供便利.

双 ToR 之间的直连链路一旦出现问题, 或 1 个 ToR 发生数据面故障, 可能会在大模型训练中引发严重的网络异常, 导致该双 ToR 下所有网卡离线. 例如, ToR1 作为主 ToR, ToR2 作为从 ToR. 如果 ToR1 因 MMU 溢出导致数据面无法正常工作, 而控制面未能识别这一根因, ToR1 和 ToR2 将无法再通过直连链路同步 ARP 或 MAC 信息. 即便带外网络仍然正常工作, ToR1 和 ToR2 的控制面仍可正常协商, 但从 ToR1 角度来看, ToR2 数据面不可达, ToR1 应继续以主 ToR 状态工作; 从 ToR2 角度来看, ToR1 的数据面不可达, 这意味着转发信息无法再同步, 但 ToR1 仍处于主状态. 为防止数据面中的不一致转发, ToR2 选择主动停机. 然而, 由于剩下的 ToR1 的数据面已经无法工作, 所有在此双 ToR 下的网卡均不可用. 这种整机架级别的故障在生产中会造成严重的不可用问题. 根据阿里云内部故障总结, 在过去 3 年中, 超过 40% 的重大故障是由堆叠双 ToR 引入的问题所导致, 这凸显了需要改进现有设计以提高数据中心网络的

可靠性和稳定性.

在堆叠双 ToR 设计中，故障的根本原因在于 2 个 ToR 之间通过直连链路进行同步. 这种设计使得一旦链路或数据面出现问题，就可能导致整个网络的不稳定. 为了解决这个问题，HPN 设计了一种去堆叠的双 ToR 方案：这一方案去除了 2 个 ToR 之间的直连链路，从根本上消除了直连链路带来的风险. 然而，这一创新想法也带来了新的挑战：如何在没有直连链路的情况下同步 ToR 信息？传统的解决方案中，2 个通过链路直连的 ToR 可以通过直连链路协商一个共享的 sysID，使得服务器能够通过 LACP[32] 将 2 个 ToR 识别为同一个逻辑设备进行通信. 但在去除直连链路的情况下，ToR 之间无法再使用 LACP 进行协商，这就要求设计一种新技术.

HPN 的解决思路是将 2 个独立的 ToR“伪装”成为一个虚拟设备，从而“欺骗”服务器，使其能够像单个设备通信一样与 2 个 ToR 通信. 这需要保证在 LACP 协商期间 2 个 ToR 使用相同的 MAC 地址和不同的 portIDs. 首先，我们通过自研交换机重新研发了 ToR 控制面模块（同时也定制了 LACP 模块）. 当 ToR 的 LACP 模块接收到 LACPDU 时，它会根据预配置的 MAC 地址生成 sysID. 如何挑选出这个预配置的 MAC 地址？这个 MAC 地址应该在相同的双 ToR 中的 2 个交换机之间保持一致. 此外，该 MAC 地址不能被任何其他服务器使用；否则，可能发生冲突. 因此，我们选择了一个 RFC 保留的虚拟路由器 MAC 地址 00:00:5E:00:01:01 作为预配置的 MAC 地址. 原则上，在同一个二层子网中，不同的双 ToR 集合中的 ToR 交换机不得使用相同的预配置 MAC 地址，以避免 MAC 地址冲突. 在阿里云中，我们在不同双 ToR 交换机之间完全采用三层转发，即通过 BGP 协议. 因此，不同的双 ToR 组天然属于不同的二层子网，消除了 MAC 地址冲突的可能性. 其次，为 2 个 ToR 生成不同的 portID 则非常容易，我们针对原始的 portID 进行相对的移位运算后通过 LACP 模块返回给服务器即使得服务器看到不同 portID. 综上，HPN 的设计完成了对服务器的“欺骗”，使其认为上面连接的是一个 ToR.

### 3.2 接入层设计：多轨优化 Segment 承载千级 GPU

接下来，我们描述 HPN 每层的创新设计，本节描述接入层设计.

为充分将不同服务器之间的 GPU 进行直连，HPN 采用了 NVIDIA 提出的多轨优化方案[25]. 在 HPN 多轨优化中，同一轨道内的网卡通过相同的双 ToR 交换机组进行连接. 不同轨道内的网卡可通过服务器内和服务器间转发的组合进行通信. 如图 6 所示，如果服务器 1 中的 GPU1 想要与服务器 3 中的 GPU2 通信，转发路径是服务器 1 中的 GPU1→服务器 1 中的 GPU2→ToR3→服务器 3 中的 GPU2. HPN 允许 3.2 Tbps（8×400 Gbps）流量通过多达 16 个 ToR 交换机送达单个服务器. 与经典 Clos 相比，HPN 中的一个 Segment 可包含的 GPU 数量增加了 8 倍. 每组双 ToR 交换机可服务 128 个 GPU，16 个 ToR 交换机共同连接 1 024 个 GPU，大大减少了转发延迟，并提供了最高的性能. 更重要的是，该设计显著减少了跨越汇聚层的流量，尽可能做到 GPU 直连.

### 3.3 汇聚层设计：一个 Pod 承载 15 000 个 GPU

在接入层网络中使用双 ToR 架构时，如果简单地在 ToR 和汇聚层之间按照典型的 Clos 拓扑连接，哈希极化问题仍会频繁发生. 这是因为在下行方向，由于双 ToR 的存在，来自 60 个汇聚交换机到 2 个 ToR 交换机的流量高度汇聚，导致流量极易产生极化. 如图 7(a) 所示，在实际训练 GPT-3175B 变体时，某个双 ToR 集合中 2 个下行端口向同一网卡的出口

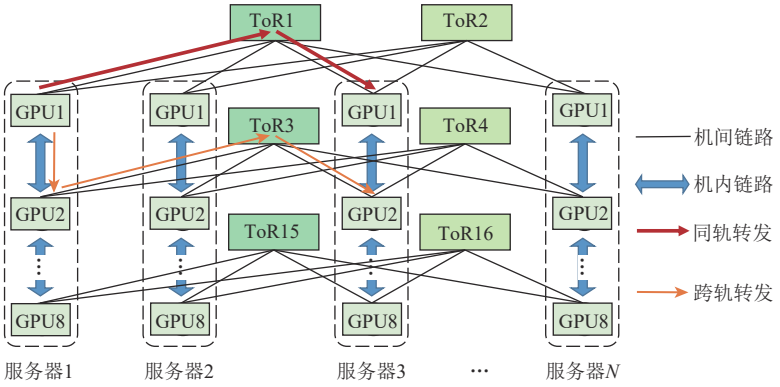



Fig. 6 Rail-optimized network of dual-ToR[8]

图 6 双 ToR 的多轨优化组网[8]

流量负载显著不同(相差约 3 倍),这种不均衡负载严重影响了训练性能.

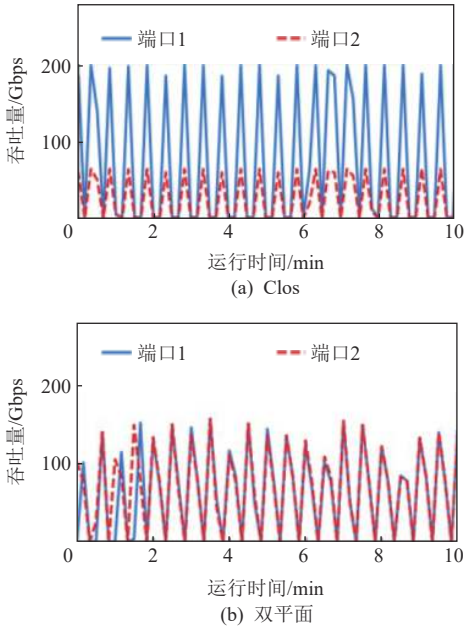



Fig. 7 Traffic on two ToRs' ports towards the same NIC[8]

图 7 ToR 交换机同一网卡的 2 个端口流量[8]

为了在 Pod 中消除哈希极化,HPN 采用了双平面设计(见图 5 中平面 1 和平面 2),在 Pod 中将每个双 ToR 集合的 ToR 交换机分为 2 个独立的组. 这种设计确保了一旦网络数据流进入 ToR 中的任一上行链路,其在 Pod 内的转发路径就完全确定,从而消除了哈希极化. 例如,图 5 中,Segment1 中的 ToR1 和 ToR2 的上行流量一定被分配到 2 个独立的平面,不会发生同时收敛到 Segment1 中的 ToR15,而如果是 Clos 架构则会引发极化问题. 在双平面设计中,这 2 个流最后会转发到 Segment1 的 ToR15 和 ToR16. 如图 7(b),根据我们的实际显示,部署双平面设计后,不同端口的入流量变得均匀,ToR 下行端口的队列长度减少了 91.8%. 深入观察发现,双平面设计显著提升了跨 Segment 流量的性能,高达 71.6%. 此外,在实际生产环境中训练 GPT-3175B 变体时,双 ToR 集合中 2 个下行端口到同一网卡的队列长度从持续拥堵的 267 KB 和 3 KB 变为平均 20 KB,显著提高了网络效率.

数据中心的选路方法众多,主要集中在解决由 ECMP 引起的负载不均[11-20]. 其中许多基于交换机的解决方案[11, 13, 15, 17, 18–19]需要交换机执行负载感知的有状态数据包转发. 然而,由于过于依赖精准的动态流感知能力或跨交换机的负载协商,这些工作在实际应用中难以实现;另外其他方法在大规模部署中未经验证. 另一方面,基于服务器侧的解决方案[12,14,16,20]选择多条路径并根据拥塞信号(例如,ECN、RTT 和 INT)在它们之间进行选路. 但这些方案不仅在选路过程中得到的是启发式结果,而且大多数需要在传输层进行修改,这在大规模 RDMA 固化硬件的网络中难以部署实现.

HPN 通过精确获取不相交等价路径(disjoint equal paths)并在集合通信层面均分它们的负载来克服现有解决方案的缺陷. 首先,对于每个新的连接请求,HPN 生成一组通过不相交等价路径的连接,这是通过大规模部署的 RePaC[33]实现的. 服务器能够直接在每个交换机中获得确切的哈希结果,基于该结果,HPN 能找到所有不相交等价路径和对应的五元组,并据此建立 RDMA 连接. HPN 部署了一个服务器-交换机端网协作系统,确保所有服务器维护最新的链路状态并计算出正确的且不相交的路径. 其次,HPN 实施了一个简单而有效的应用层负载均衡方案,通过计数器记录当前活动的工作队列元素(WQEs)中的总字节数,以揭示当前连接的拥塞状况,并通过计数器最小的连接发送消息.

上述设计得益于双平面设计. 具体地,在寻找不相交路径时,我们只需要搜索每个 ToR 交换机中的链路(最多搜索 60 个链路),显著减少了路径搜索空间的时间消耗. HPN 的选路复杂性 $O(60)$ 与 FatTree 复杂度 $O(48\times48)=O(2\,304)$ 以及 Jupiter 复杂度 $O(2\,048)$ 相比,可将计算复杂性减少 1~2 个数量级.

单 Pod 承载 15 000 个 GPU. 双平面设计不仅优化了路径选择,还显著减少了接入层和汇聚层之间的连接数量. 这种设计允许汇聚交换机在同一个 Pod 中支持更多的 Segment,从而使汇聚层网络的规模得以加倍. 此外,通过将汇聚-核心的带宽收敛比设为 15∶1,我们为汇聚交换机上互连额外的 Segment 释放了 87.5% 甚至更多的端口. 这些设计决策最终帮助我们实现了在同一个 Pod 中容纳 15 000 个 GPU 的目标,并为每个 GPU 提供了 400 Gbps 的网络接入能力.

### 3.4 HPN 网络规模扩展的关键机制总结

表 1 展示了 HPN 的设计中有助于扩展网络规模的关键机制. 首先,双 ToR 设计允许单个网卡的 2×200 Gbps 带宽通过 2 个 ToR 交换机提供服务,这一设计直接使得网络规模加倍. 结合最新的 51.2 Tbps 单芯片交换机和多轨优化网络,单个 Segment 能够容纳 1 000 个 GPU. 汇聚层的双平面设计进一步释放了每个汇聚交换机中一半数量的端口,极大地提升了网络的扩展性. 最后,通过汇聚-核心的 15∶1 带宽收敛比,HPN 实现了在一个 Pod 中支持 15 000 个 GPU

的设计目标. 目前根据实际观察, 不会有单任务超过万卡 GPU 需求, 因此一个 Pod 已经能够覆盖我们当前最大的训练任务. 通过单个 Pod 支持 15 000 个 GPU, 我们进一步减少了连接多个 Pod 所需的不必要交换机数量. 根据阿里云内部统计, HPN 的设计节省了大约 30% 的总体网络建设成本.

**Table 1 Key Mechanisms for Maximal Scale[8]**

**表 1 扩展规模的关键机制[8]**

| 关键机制 | 一层规模（扩展倍数） | 二层规模（扩展倍数） |
|---|---|---|
| 双 ToR | 128（×2） | 4 000（×2） |
| 多轨优化 | 1 000（×8） | |
| 双平面 | | 8 000（×2） |
| 15∶1 收敛比 | | 15 000（×1.875） |

### 3.5 HPN 对大模型训练的端到端评估

HPN 已作为阿里云大模型训练的集群架构被广泛部署, 为上百个客户的数千个模型训练任务提供服务. 为了突出 HPN 的效果, 我们将 HPN 与我们上一代训练网络架构 DCN+进行比较. DCN+的后端网络是一个传统的 3 层 Clos 数据中心和堆叠双 ToR 设计. 在 DCN+中, 每个 Segment 承载 128 个 GPU, 每个 Pod 包含 4 个 Segment.

我们采用阿里云自主研发的大模型在 2300+ GPU (288+服务器)的训练环境作为评估环境. 该大模型最初在 DCN+上训练, 然后迁移到 HPN. 在 DCN+中, 训练任务跨越了 19 个 Segment, 而在 HPN 中, 训练任务只需 3 个 Segment. 我们观察到大模型迁移后性能显著提升. 如图 8(a) 所示, 端到端训练性能提高了超过 14.9%. 这种端到端的性能提升在实际生产环境中具有很大价值. 考虑到整个训练集群的构建可能会花费数十亿美元, 14.9% 的性能提升可达到显著的成本节省. 我们进一步统计了训练期间汇聚交换机的数据. 汇聚交换机承载跨 Segment 流量, 其统计数据直接反映网络状态. 如图 8(b) 所示, 跨 Segment 流量平均减少了 37%. 较少的跨 Segment 流量导致了网络中拥塞骤降. 图 8(c) 展示了汇聚交换机下行链路队列长度分布. 大流量和哈希冲突在 DCN+中不断积攒队列长度; 而在 HPN 中, 该问题得到了极大的缓解.

## 4 多任务通信调度 Crux 优化 GPU 利用率

在 1.2 节中提到的挑战 2 主要来自于多任务的大模型训练中任务间存在严重的通信竞争问题, 这导致整体 GPU 利用率严重受限.

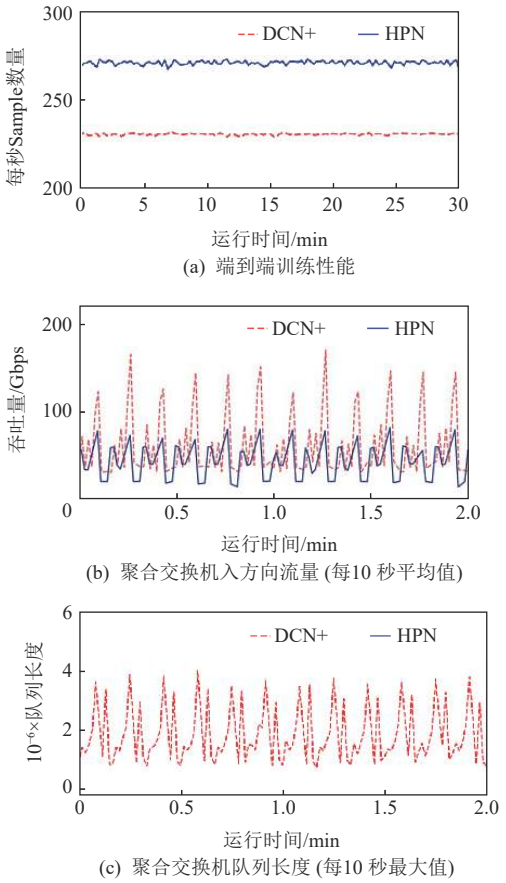



Fig. 8 Model training performance on 2300+GPU under different network architectures[8]

图 8 2300+GPU 在不同网络架构下的模型训练性能[8]

我们的目标是优化 GPU 利用率. 解决通信竞争问题前我们要思考得到什么样的优化目标. 虽然已有相关工作旨在优化任务完成时间[6,21-22], 而针对 GPU 利用率进行优化的工作并没有太多涉及. 但是, 我们从工业界得到的经验认为 GPU 利用率是更好的量化指标, 其能够衡量大模型训练的整体吞吐量以及模型的训练成本. 此外, GPU 利用率和任务完成时间原则上并不冲突, 优化整体 GPU 利用率将带来最佳的任务完成时间. 因此, 本文研究工作意在优化 GPU 利用率. 此外, 从技术的原理上来说, 已有相关工作主要是通过调度任务完成优化的, 本文研究工作是通过通信调度进行 GPU 利用率优化的, 这两者其实是属于正交的关系, 可以互为补充而不矛盾. 任务调度是把任务分配好, 通信调度是任务放好固定之后, 决定谁先得到调度.

由于多任务大模型训练的复杂性, 最大化或者极端优化训练集群的 GPU 利用率已被证明是 NP 完全问题[9]. 因此, 如何对其建模并将其转化为一个近似可求解问题是我们面临的第 1 个技术挑战. 在实际大模型训练中能做的通信调度手段非常有限. 例如,

当前 HPN 的选路设计(3.3 节所述)只能基于五元组进行选路,而交换机的端口队列仅支持有限数量的优先级,这大大约束了实际可用的通信调度方法. 换言之,在支持多任务大模型训练的集合通信场景,普通的基于 ECMP 的调度方法显得过于简单. 因此,如何在实际多任务大模型训练中设计有效的通信调度方法,从而最大化 GPU 利用率,是第 2 个挑战.

### 4.1 问题定义和建模

我们首先对多任务大模型训练定义和建模. 假设一个 GPU 数据中心内部通过网络相连,网络拓扑为 $G=\langle V^{\mathrm{G}},E^{\mathrm{G}}\rangle$,其中 $V^{\mathrm{G}}$ 是 GPU 集合,$E^{\mathrm{G}}$ 是所有链路集合(包括网络链路和主机内链路). 每个链路 $e\in E^{\mathrm{G}}$ 的带宽为 $B_e$. 定义 $J$ 表示此集群中的一组训练任务. 每个任务 $j\in J$ 占用 $V^{\mathrm{G}}$ 中的一些 GPU. 在任务 $j$ 的每次迭代中,它会在其占用的 GPU 上产生 $W_j$ 计算工作负载(以浮点运算表示),并向网络生成一定的数据流量. 定义 $M_{j,e}$ 为单次迭代中由任务 $j$ 在链路 $e$ 上产生的数据量. 无论哪种并行策略,一个任务的计算和数据通信可能会重叠(尤其是大模型训练任务).

**定义 1.** GPU 利用率. 在给定的时间段 $T$ 中,假设 GPU $v$ 完成了计算工作负载 $L_v$. 这个 GPU 集群的总体 GPU 利用率 $U_T$ 定义为

$$U_T=\sum_{v\in V^{\mathrm{G}}}L_v.$$

我们的目标为最大化 $U_T$. 最大化 $U_T$ 并不比 NP 完全问题简单. 当考虑将所有 GPU 的时间段看做背包,将训练任务的计算工作量作为物品时,最大化 $U_T$ 其实是背包问题的一个复杂版本. 与原始背包问题相比,最大化 $U_T$ 问题有更复杂的约束:任务的计算工作量不能被自由地分配给一个 GPU 的时间段,因为它的计算可能会被通信阻塞,而每次通信的持续时间取决于与网络中其他任务的通信竞争.

因此,我们首先在一个简单的案例中探讨解决方案:单链路中的流量调度. 具体地,假设一个具有恒定带宽 $B_{e_0}$ 的链路 $e_0$. 对于其他链路 $e\in E^{\mathrm{G}}\backslash e_0$,有 $B_{e_0}=\infty$. 在这种设置下,GPU 利用率完全受 $e_0$ 影响. 现在问题简化为:在该单链路情况下如何最大化 GPU 利用率 $U_T$?

**定义 2.** 任务 $j\in J$ 的 GPU 强度(intensity) $I_j$:

$$I_j=\frac{W_j}{t_j},$$

其中 $t_j=\max\limits_{\{e\in E^{\mathrm{G}}\}}\dfrac{M_{j,e}}{B_e}$. 由于只有 1 个链路 $e_0$,有 $t_j=M_{j,e_0}/B_{e_0}$. GPU 强度 $I_j$ 的含义是,任务 $j$ 在每次通信工作时长 $t_j$ 中完成 $W_j$ 的计算工作. 对于网络来说,传输较高 $I_j$ 的任务的数据流更重要,因为它完成了更多 GPU 的计算量,从而更好地提高了 GPU 利用率.

最大化 $U_T$ 的目标通过证明转化为在给定链路上传输的所有任务的 GPU 强度之和[9].

### 4.2 Crux 解决方案

我们设计并研发了 Crux,一款以优化 GPU 利用率为目标的集合通信调度方法. 该方法已经在 HPN 网络部署的自研集合通信库中实现.

为了最大化 GPU 利用率,我们要通过通信调度得到目标链路上所有 GPU 强度之和. 理想的调度是让网络在 100% 的时间内传输最 GPU 密集型任务(即,拥有最高 GPU 强度的任务)的数据流,例如,在图 9 中为任务 1 花费整个时间段 $T$. 然而,这个简单案例与现实之间存在巨大差距. 首先,每个训练任务只在一部分时间里以及在网络路径的一部分上生成数据流,遵循其自己的流量模式. 一个合适的调度器应当在充分利用 GPU 集群网络的同时,优先考虑 GPU 密集型任务. 然而,GPU 集群的流量调度是受限的. 例如,ECMP 支持基于五元组的有限路径选择,而 DSCP 仅支持整个集群网络中几种统一的有限优先级. 这些约束导致无法实现理想的通信调度.

针对上述约束,我们提出了 2 个机制进行通信调度,以提升集群的 GPU 利用率.

1)Crux 路径选择方法:基于 GPU 强度的路径选择. 鉴于 GPU 密集型任务之间的竞争更加降低整体 GPU 利用率,Crux 首先提出了一种基于 GPU 强度的路径选择,为 GPU 密集型任务选择不同的路径以避免竞争,并且一定程度地允许其他任务之间的竞争.

我们的核心思路是:尽管通信竞争是不可避免的,但通过仔细的路径选择可以避免 GPU 密集型任务之间的通信竞争,因为 GPU 密集型任务对集群的 GPU 利用率的影响更为显著. 假设 2 个高 GPU 强度的任务(任务 1 和任务 2)和 2 个低 GPU 强度的任务(任务 3 和任务 4)正在一个包含 2 个汇聚交换机的集群网络中运行. 如果不考虑 GPU 强度,则默认的选

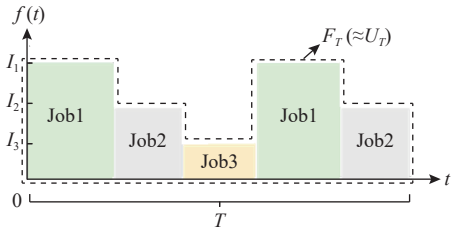



Fig. 9 Deriving GPU utilization problem to a flow scheduling problem

图 9 将 GPU 利用率问题转化为流量调度问题

路方法可能会将任务 1 和任务 2 映射到一个交换机，将任务 3 和任务 4 映射到另一个交换机. 尽管这样在任务级别上平衡了网络负载，但它在 2 个 GPU 密集型任务之间产生了通信竞争. 相比之下，将任务 1 和任务 3 映射到一个交换机以及将任务 2 和任务 4 映射到另一个是更好的路径选择，因为它提供了一个机会来保持 2 个 GPU 密集型任务的性能不受影响（如果优先考虑这 2 个任务而不是另外 2 个）.

因此 Crux 提出了基于 GPU 强度的路径选择. 对于集群中的多个任务，从最 GPU 密集型的任务开始，依次进行路径选择. 对于每个任务，选择在当前所有可用路径中最不拥堵的路径（信息可以通过 in-network telemetry 获得）. 这为 GPU 密集型任务提供了更多机会在彼此之间选择不同的路径.

2）Crux 的优先级分配：考虑大模型训练任务特性的优先级分配. 由于大模型训练任务的特点，例如，多次迭代以及计算-通信重叠，直接根据它们的 GPU 强度为任务分配优先级可能会在时间维度上产生不平衡的网络负载（例如，有时有多个任务竞争网络资源，有时网络可能暂时空闲），从而导致次优表现. 我们分析了这些新特性如何影响网络负载模式，并提出了一个新的数学模型来微调优先级分配.

如前所述，GPU 密集型任务优先级应该更高. 然而，这种贪婪策略可能会导致时间维度上的不均匀网络负载（例如，有时多个任务竞争网络，有时网络可能暂时空闲）. 这是因为这些训练任务具有不同的迭代时间和计算-通信重叠行为. 基于这些新特性，通过微调优先级分配可以错开不同任务的数据流，并获得更好的性能.

例 1：任务 1 和任务 2 争夺同一个链路. 任务 1 每次迭代需要 2 s 的计算时间，而任务 2 只需要 1 s. 根据 GPU 强度公式计算，它们具有相等的 GPU 强度. 如图 10 所示，如果优先级给了任务 1，链路将有 25% 的时间处于空闲状态；如果优先级给了任务 2，链路

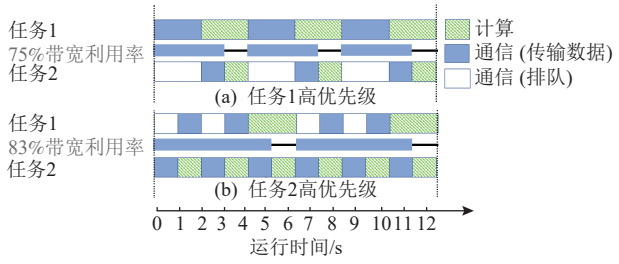



Fig. 10 Example of communication contention between jobs with different iteration time

图 10 不同迭代时间的任务通信冲突示例

只有 17% 的时间空闲. 优先考虑迭代时间短的任务可更好地利用链路带宽传输更多数据，提高整体 GPU 利用率.

例 2：任务 1 和任务 2 竞争同一个链路. 任务 1 每次迭代需要 4 s 的计算时间，而任务 2 只需要 2 s，因为任务 2 使用了更多的 GPU. 为了简化，假设一个任务在完成一次迭代的 50% 计算后开始通信（例如，在完成大约一半计算的前向传播后，数据通信可以与后向传播重叠）. 如图 11 所示，如果任务 1 优先，链路将有 17% 的时间空闲；如果任务 2 优先，链路可以被完全利用. 这是因为任务 1 中的通信可以被其计算完全重叠. 少于 1 s 的通信延迟不会减少其训练吞吐量或 GPU 利用率. 相反，任务 2 对通信延迟非常敏感，因为它的通信不能被重叠.

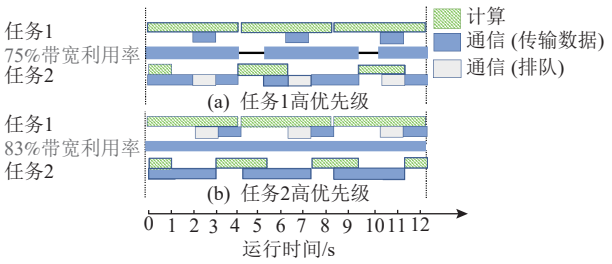



Fig. 11 Example of communication contention between jobs with different computation contention overlapping

图 11 不同计算冲突重叠的任务通信冲突示例

为了衡量这些新特性如何影响优先级分配，Crux 为每个任务 $j$ 提出了一个修正因子 $k_j$. 例如，在图 10 中，2 个任务的 GPU 强度相同. 当任务 1 优先时，网络花费 6 s 传输任务 1 的数据，花费 3 s 传输任务 2 的数据. 当任务 2 优先时，网络传输任务 1 的数据需要 4 s，传输任务 2 的数据需要 6 s. 基于我们的优化目标，如果 2 个任务的优先级相同，优先任务 1（多传输任务 1 的数据 2 s）或优先任务 2（多传输任务 2 的数据 3 s）应该获得相同的 GPU 利用率. 因此，任务 1 应该具有 1.5 倍的 GPU 强度，以便与任务 2 具有相同的优先级. 因此，将任务 1 视为参考任务，任务 2 的修正因子 $k_{j_2}$=1.5.

最终，Crux 利用 GPU 强度和修正因子，任务 $j$ 的优先级 $P_j$ 定义为：$P_j = k_j I_j$.

下面总结 Crux 方法. Crux 首先通过路径选择方法来为要调度的任务选择其适合的链路，然后通过优先级分配方案决定在该链路上优先调度哪个任务，从而达到 GPU 利用率的优化.

### 4.3 Crux 的评估

为了评估 Crux 的性能收益，我们展示在 2 000

多GPU的集群2周生产环境的实验情况. 我们在HPN拓扑进行实验, 其由6个ToR交换机、12个汇聚交换机和32个核心交换机组成. 每台主机通过8条链路连接到2个ToR交换机. 实验以整体GPU利用率作为GPU集群的性能指标.

在图12所示的实验结果中, 我们评估了多种模型, 包括GPT3变体、BERT模型, 以及阿里云自研的通义千问模型等. 我们比较了Crux通信调度机制(包括仅使用优先级分配的Crux-PA、结合选路和优先级分配的Crux-PS-PA, 以及结合选路、优先级分配和优先级压缩的完整版Crux-full)与最新成果(包括通用的CoFlow调度机制Sincronia[21]、任务内通信调度机制TACCL[22], 以及多任务间通信调度方法CASSINI[6]), 发现Crux的利用率提升最为显著.

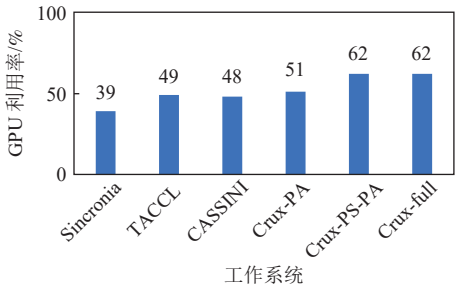



Fig. 12 GPU utilization rates between communication schedulers[9]

图12 不同通信调度方法的GPU利用率[9]

## 5 大模型训练的网络可靠性挑战与思路

1.3节提到的网络可靠性挑战是大模型训练中最关键的问题之一. HPN的设计只是从架构角度解决其中存在的单点风险(即ToR单点隐患), 因为单点风险对可靠性影响最为直接. 但远没有解决所有可靠性挑战.

相比传统的云计算业务, 大模型训练中的可靠性问题有3个鲜明的特征: 1)大模型训练涉及到的软硬件设施极多, 程序行为复杂(例如, 训练框架、容器、集合通信、网卡、GPU等), 技术迭代非常迅速, 因此新的可靠性问题不断涌现, 各个组件在演进过程中新的bug甚至租户训练代码中的bug也不断出现; 2)大模型训练包含非常多的同步操作(这在通用云计算中是比较少的), 因此, 当某个worker节点发生问题时会导致所有worker节点都变慢或停止, 仅通过观察训练任务是无法获得足够的信息用于问题定位的; 3)由于大模型训练通常涉及大量设备, 发生可靠性问题时会导致大量机器停止工作或以较低效率工作, 造成比通用云计算范围更大的业务和成本影响, 因此对问题定位效率要求更高.

以上3个特征为大模型的可靠性问题的快速定位和恢复带来了新挑战. 目前业界普遍采用的监控工具还是以低精度的硬件监控为主, 粒度普遍为秒级, 配合人工为主的问题定位和恢复. 这样的监控及问题定位体系缺乏不同硬件指标间的自动关联, 尤其欠缺与训练任务行为的关联, 因而难以适应大模型训练的可靠性需求. 首先, 由于模型训练时各个软硬件设施的行为很复杂, 有很多问题不是由于单纯硬件故障引起的(例如, 训练框架、容器、集合通信库行为、拥塞控制, 以及硬件网络配置联动造成), 因此仅检查各个硬件或软件指标, 能覆盖到的问题十分有限. 其次, 由于各个worker之间累积的细微行为差异会被高频地同步消除掉, 因此秒级的监控通常并不能精确捕捉到worker之间的不同, 从而难以发现有问题的机器. 最后, 由于涉及到的模块众多, 每次人工定位可靠性问题都需要大量人力和时间, 投入很多设备进行测试实验, 因此会对业务带来较严重的损失.

针对以上挑战, 我们尝试研发高精度的软硬件结合监控技术, 配合自动化的问题定位算法, 来提升可靠性问题的定位效率, 并尝试克服高精度监控带来的性能挑战. 在硬件方面, 我们使用NSYS API对GPU、NVLINK、PCIe、DRAM等设备的各项指标进行每秒万余次的高频监控, 使得粒度达到微秒级. 在软件方面, 我们通过torch profiler监控训练任务的所有Python函数运行记录、CUDA kernel执行记录, 以及通过在集合通信库中加入log来监控通信库工作状况. 通过将以上软硬件信息进行自动匹配关联, 即可通过预先设计好的算法来进行自动诊断分析, 从而自动化定位出常见的计算问题、通信问题等根因. 对于代码层面的问题(例如, 客户代码问题或软件程序应用的bug), 我们的算法无法直接定位到根因, 但也可以通过分析训练各环节性能是否正常自动产出分析报告, 大幅度缩小问题排查范围, 提升定位效率.

综上, 我们认为大模型训练和过去通用云计算的可靠性保障存在很多不同于新的挑战. 这些挑战要么表现在相同方法论但要求的指标(例如定位精准度、反应速度等)更高, 要么表现在通用计算没有的问题根因. 上面描述的只是当前的尝试解决手段, 如何打造更完善和更全面的解决方案系统体系是接下来重点关注的.

## 6 更多背后的思考与部署经验讨论

大模型训练作为不同于通用云计算的一种新兴且迅速成为主流智算的计算通信模式，我们一路走来思考和设计实践了很多. 目前还有很多值得深入思考的技术问题、方案取舍的思考，以及很多设计背后的原因经验. 本节选择一些关键的思考进行讨论，意在为大模型训练领域提供更多的来自工业界的经验.

### 6.1 HPN 为什么选择以太网而不是 InfiniBand

主流云厂商在构建后端训练网络时主要有 2 套方案备选：以太网和 InfiniBand[34]. 总体来说，InfiniBand 发源于高性能计算（high performance computing, HPC）场景，在集群数据传输性能上做了很多针对性的优化，因此性能指标本身比以太网更优. 但是，以太网快速演进的单芯片转发能力是以太网的一个潜在优势. 近几年来，以太网单芯片转发能力相比 InfiniBand 都是领先 1 倍的，并且看上去会一直持续下去. 这对于网络架构的选择也非常重要，尤其是当前我们看到这样的单芯片快速演进可以让我们逐渐从三层组网走向二层组网. 另一方面，由于 InfiniBand 专精于 HPC 领域，在当前大模型训练全面上云的环境下，InfiniBand 可能天生缺乏很多云化服务所需的重要特性. 与之相反，我们认为以太网在诞生之初就被设计为支持泛在连接，具有良好的通用性. 此外，近年来云数据中心的迅猛发展也依托于以太网，在此过程中，很多以太网积累的高可扩展性/弹性、网络虚拟化、多租支持、开放生态等特性是 AI 基础设施网络长期发展的重要基础. 当然，这里的讨论仅代表阿里云基础设施网络的思考和判断，仅代表以太网更适合阿里云的情况，对于不同云厂商、网络架构、人员组成等需要具体问题具体分析.

### 6.2 HPN 为什么不在汇聚层也采用多轨优化

为什么不在汇聚层采用优化轨道的思路来为 HPN 设计一个汇聚层的完全互连网络，从而实现汇聚层的多轨优化？虽然看起来这样的设计会使得 HPN 设计中一个 Pod 可以支持超过 120 000 个 GPU，然而，实际中这样的设计会造成相当大的扩展性负担. 同时，汇聚层多轨优化是基于当前大模型的类型基本上是轨道内通信这一假设的，而未来会出现更多轨道外（或轨道间）的通信模式. 因此，牺牲巨大的维护成本和开销，换来的模型通信效率收益是相对较小的. 所以，HPN 在汇聚层没有采用多轨直连的优化思想，而是利用核心层支持更大规模的扩展.

### 6.3 训推一体架构的扩展

目前阿里云线上用于训练和用于推理的机型规格是完全不同的. 但考虑到：1）模型规模越来越大，需要更高性能的 GPU 来做推理服务，推理所需要的 GPU 性能规格将会越来越接近训练用的 GPU（或者前一代的训练 GPU）；2）很多租用云上训练集群的客户在模型训练完毕或完成第 1 版训练后都会有后续的推理需求. 由于推理需求通常有非常大的潮汐特性，我们遇到很多客户倾向于将训练任务和推理任务部署在租用的相同集群上，这样能更好地实现租用资源复用.

综上，在设计 HPN 前端网络规格时我们充分考虑了上述需求. 目前 HPN 的 2×200 Gbps 前端接入网络即便在专用推理机型中也已经是当前高规格设计. 因此，这样的设计可以保证部署的机型可以同时用于支持训练和推理，帮助实现训推一体，支持用户的弹性需求.

## 7 总结与展望

本文从大模型训练相比于通用云计算的特有挑战开始，描述了当前大模型训练面临的 3 个关键的网络挑战：1）通信模式差异造成负载难均衡；2）多训练任务通信竞争影响 GPU 利用率；3）对故障的高敏感性. 随后，详细介绍了阿里云的解决方案：专门为大模型训练设计研发的新型数据中心网络架构 HPN 以及面向多任务的大模型训练集合通信调度 Crux. 然而，这 2 个解决方案只是从架构以及集合通信调度角度试图解决上述三大挑战，并不能根本性解决. 例如，挑战 3 描述的可靠性挑战，目前仍然存在很多开放性问题值得我们去研究. 此外，不局限于这 3 个挑战，随着大模型的训练和推理进一步发展，还会有更多挑战出现. 下面，我们讨论和展望一些未来可能出现的研究方向.

展望 1. 在大模型训练和推理过程中，分布式节点之间通过集合通信库进行高效同步，因此，提升集合通信的性能是提高训练效率和降低训练成本的关键. 随着新模型和框架技术的不断发展，新的集合通信模式将会不断涌现，例如在 MOE（mixture of experts）模型中将存在不均匀的 Alltoall 通信. 此外，网络架构的持续升级也会改变可用于调度的通信资源. 现有的集合通信库（如 NCCL, RCCL 以及 Gloo）通常采用固定的通信算法，难以适应不断变化的模型、框架和

网络架构. 针对每个具体场景设计最优的集合通信算法需要耗费大量的人力和时间. 因此, 如何自动分析模型和框架的通信行为, 并通过感知底层网络资源自动生成高效的通信算法, 将成为一个重要的方向.

展望 2. 大模型训练和推理是一个复杂的系统, 随着模型规模的不断增大, 系统出现问题的概率也随之增加. 一方面需要快速的故障定位和恢复机制, 能够尽量减少训练或者推理的中断时间. 另一方面, 系统还应具备一定的容错能力. 当故障对系统性能影响较小时, 系统能够自适应调整通信和计算任务. 例如, 通过动态负载均衡、资源分配优化以及通信算法改进, 系统可以在部分模块失效的情况下, 重新调整工作负载, 以最大限度地利用现存资源, 从而改善整体系统性能.

展望 3. 随着大模型训练推理的进一步发展, 对可靠性故障(或性能)根因的定位速度与精确度要求会越来越高. 然而, 与传统的通用云计算故障根因定位不同, 大模型训练和推理不但是多层次(例如, 训练框架、容器、集合通信、网卡、GPU 等), 而且计算模式也大为不同(多 GPU 组同步), 因此如何针对性地设计能够获得多层次信息进行这些信息关联的快速、高效定位工具是大模型训练推理后续可靠性保证的重要方向.

**作者贡献声明:** 翟恩南提出论文总体思路, 负责论文整体撰写和结构设计; 操佳敏、钱坤、关宇参与论文部分章节修改以及对照实验.

## 参 考 文 献

[1] OpenAI, Josh A, Adler S, et al. GPT−4 technical report. [J]. arXiv preprint, arXiv: 2303.08774, 2024

[2] OpenAI. Introducing ChatGPT[EB/OL]. [2022-11-30]. https://openai.com/blog/chatgpt

[3] MLSYS ORG. Vicuna: An open-source chatbot impressing GPT−4 with 90% ChatGPT quality[EB/OL]. [2023-03-30]. https://lmsys.org/blog/2023-03-30-vicuna/

[4] NVIDIA. Megatron-LM. [EB/OL]. [2024-06-19]. https://github.com/NVIDIA/Megatron-LM

[5] Microsoft Research. DeepSpeed[EB/OL]. [2024-06-19]. https://www.microsoft.com/en-us/research/project/deepspeed/

[6] Rajasekaran S, Ghobadi M, Akella A. CASSINI: Network-aware job scheduling in machine learning clusters[C]//Proc of USENIX NSDI. Berkeley, CA: USENIX Association, 2024: 1403−1420

[7] Jiang Ziheng, Lin Haibin, Zhong Yinmin, et al. MegaScale: Scaling large language model training to more than 10, 000 GPUs[C]//Proc of USENIX NSDI. Berkeley, CA: USENIX Association, 2024: 745−760

[8] Qian Kun, Xi Yongqing, Cao Jiamin, et al. Alibaba HPN: A data center network for large language model training[C]//Proc of ACM SIGCOMM. New York: ACM, 2024: 691−706

[9] Cao Jiamin, Guan Yu, Qian Kun, et al. Crux: GPU-efficient communication scheduling for deep learning training[C]//Proc of ACM SIGCOMM. New York: ACM, 2024: 1−15

[10] Microsoft DeepSpeed. Model checkpointing[EB/OL]. [2023-01-31]. https://deepspeed.readthedocs.io/en/latest/model-checkpointing.html

[11] Alizadeh M, Edsall T, Dharmapurikar S, et al. CONGA: Distributed congestion-aware load balancing for datacenters[C]//Proc of ACM SIGCOMM. New York: ACM, 2014: 503–514

[12] Dixit A, Prakash P, Hu Y C, et al. On the impact of packet spraying in data center networks[C]//Proc of IEEE INFOCOM. Piscataway, NJ: IEEE, 2013: 2130–2138

[13] Ghorbani S, Yang Zibin, Godfrey P B, et al. DRILL: Micro load balancing for low-latency data center networks[C]//Proc of ACM SIGCOMM. New York: ACM, 2017: 225–238

[14] Katta N, Hira M, Ghag A, et al. CLOVE: How I learned to stop worrying about the core and love the edge[C]//Proc of ACM Workshop on Hot Topics in Networks. New York: ACM, 2016: 155–161

[15] Katta N, Hira M, Kim C, et al. HULA: Scalable load balancing using programmable data planes[C]//Proc of ACM SOSR. New York: ACM, 2016: 1–12

[16] Qureshi M A, Cheng Yuchung, Yin Qianwen, et al. PLB: Congestion signals are simple and effective for network load balancing[C]//Proc of ACM SIGCOMM. New York: ACM, 2022: 207–218

[17] Sen S, Shue D, Ihm S, et al. Scalable, optimal flow routing in datacenters via local link balancing[C]//Proc of ACM CoNEXT. New York: ACM, 2013: 151–162

[18] Vanini E, Pan Rong, Alizadeh M, et al. Let it flow: Resilient asymmetric load balancing with flowlet switching[C]//Proc of USENIX NSDI. Berkeley, CA: USENIX Association, 2017: 407–420

[19] Zats D, Das T, Mohan P, et al. DeTail: Reducing the flow completion time tail in datacenter networks[C]//Proc of ACM SIGCOMM. New York: ACM, 2012: 139–150

[20] Zhang Hong, Zhang Junxue, Bai Wei, et al. Resilient datacenter load balancing in the wild[C]//Proc of ACM SIGCOMM. New York: ACM, 2017: 253–266

[21] Agarwal S, Rajakrishnan S, Narayan A, et al. Sincronia: Near-optimal network design for coflows[C]//Proc of ACM SIGCOMM. New York: ACM, 2018: 16−29

[22] Shah A, Chidambaram V, Cowan M, et al. TACCL: Guiding collective algorithm synthesis using communication sketches[C]//Proc of USENIX NSDI. Berkeley, CA: USENIX Association, 2023: 593−612

[23] NVIDIA. NVLink and NVSwitch[EB/OL]. [2024-06-19]. https://www.nvidia.com/en-us/data-center/nvlink/

[24] Meta. Meta's evolution of network for AI[EB/OL]. [2023-11-01]. https://www.youtube.com/watch?v=5gOOtFySrqA

[25] NVIDIA. NVIDIA DGX SuperPOD: Next generation scalable infrastructure for AI leadership[EB/OL]. [2023-09-22]. https: /docs. nvidia. com/dgx-superpod-reference-architecture-dgx-h100. pdf

[26] Al-Fares M, Loukissas A, Vahdat A. A scalable, commodity data center network architecture[C]//Proc of ACM SIGCOMM. New York: ACM, 2008: 63–74

[27] Greenberg A, Hamilton J R, Jain N, et al. VL2: A scalable and flexible data center network[C]//Proc of ACM SIGCOMM. New York: ACM, 2009: 51–62

[28] Bai Wei, Abdeen S S, Agrawal A, et al. Empowering Azure storage with RDMA[C]//Proc of USENIX NSDI. Berkeley, CA: USENIX Association, 2023: 49–67

[29] Poutievski L, Mashayekhi O, Ong J, et al. Jupiter evolving: Transforming Google's datacenter network via optical circuit switches and software-defined networking[C]//Proc of ACM SIGCOMM. New York: ACM, 2022: 66–85

[30] Linux GNU. Linux bonding modes[EB/OL]. [2010-01-08]. https://thelinuxcluster.com/2010/01/08/linux-bonding-modes/

[31] IEEE Standard SA. IEEE 802.3ad[EB/OL]. [2000-06-28]. https://standards.ieee.org/ieee/802.3ad/1088/

[32] IEEE Standard SA. IEEE standard for information technology – local and metropolitan area networks – specific requirements – part 3: CSMA/CD access method and physical layer specifications amendment 5: Media access control parameters, physical layers, and management parameters for energy-efficient Ethernet[EB/OL]. [2010-10-27]. https://standards.ieee.org/ieee/802.3az/4270/

[33] Zhang Zhehui, Zheng Haiyang, Hu Jiayao, et al. Hashing linearity enables relative path control in data centers[C]//Proc of USENIX ATC. Berkeley, CA: USENIX Association, 2021: 855–862

[34] NVIDIA. InfiniBand networking solutions[EB/OL]. [2024-06-11]. https://www.nvidia.com/en-us/networking/products/infiniband/

**Zhai Ennan**, born in 1984. PhD. Member of CCF. His main research interests include computer network and distributed systems.

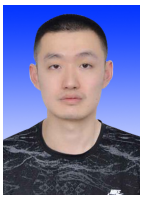
**翟恩南**, 1984 年生. 博士. CCF 会员. 主要研究方向为计算机网络、分布式系统.

**Cao Jiamin**, born in 1997. PhD. Member of CCF. Her main research interests include AI infrastructure and programmable networks.

**操佳敏**, 1997 年生. 博士. CCF 会员. 主要研究方向为 AI 基础实施、可编程网络.

**Qian Kun**, born in 1993. PhD. Member of CCF. His main research interests include AI infrastructure and storage.

**钱　坤**, 1993 年生. 博士. CCF 会员. 主要研究方向为 AI 基础设施、存储.



**Guan Yu**, born in 1993. PhD. His main research interests include AI infrastructure and video streaming.

**关　宇**, 1993 年生. 博士. 主要研究方向为 AI 基础设施、视频传输流.