【第六课】AI-Infra 总览：构建支撑大规模训练与推理基础设施平台

目录

1. 导读

在前面的度学堂直播课程《AI体系基础全覆盖课程》中，我们从 **AI 发展史与 Transformer 架构** 出发，逐步深入讲解了 **芯片、算子与 AI 编译器**，并在推理与训练章节中，帮助大家建立了对 **大模型训推流程与工具体系** 的整体认知。

然而，**当模型真正部署于生产环境，面临规模持续扩大、时长不断延伸、成本日益攀升的现实挑战时，仅靠上述这些知识和技术是否足够？**答案往往是**否定**的。

因此，在今天的课程中，我们将视角再提升一层——从 **AI基础设施与系统工程** 的角度出发，系统梳理支撑大规模训练与推理的 **AI-Infra 基础设施平台是如何构建的。**

本节课程内容将围绕以下四个层次展开：

- **为什么需要 AI-Infra**：从模型能力的竞争，走向算力、系统与平台的综合竞争；
- **AI-Infra 架构全景**：解析支撑大规模训推的关键技术栈与分层架构；
- **平台实践案例**：结合工程与平台视角，探讨 AI-Infra 如何真正落地并持续演进；
- **AI-Infra 发展趋势和总结**：探索 AI Infra 的技术趋势，洞察 AI Infra 未来演进方向

2. 为什么需要 AI-Infra：从模型竞争到基础设施竞争

在深度学习发展的早期，AI 能力的提升主要来自三点：**更好的模型结构**、**更大的数据规模**、**更高效的训练方法**。

演进维度	代表性案例	带来的关键变化
模型结构	CNN、ResNet、Transformer	突破**模型表达能力**边界，持续抬升模型能力上限
数据规模	ImageNet、Web 级文本数据	从**小样本走向大规模预训练**，模型泛化能力实现跃迁
训练方法	Adam、BatchNorm、分布式训练	显著**提升训练稳定性与效率**，使**模型规模化**成为可能

进入大模型时代后，AI发展的核心驱动已从早期的**模型、数据与训练方法**“三驾马车”，逐渐演变为**模型、数据、训练方法、算力、工程与组织协同**的“多元驱动”。模型、数据、训练方法、算力、工程与协同能力的深度融合，既是突破AI能力上限的**新门槛**，也是释放其潜能的**关键赋能体系**。

2.1. 模型规模带来的系统性挑战

当模型从亿级参数演进到百亿、千亿规模后，**系统层面面临更大的挑战**：

- 单机显存难以容纳完整模型权重与推理阶段不断膨胀的 KV Cache
- 多卡多机成为常态，通信与同步开销逐步主导训练与推理性能
- 训练任务运行周期显著拉长，对系统稳定性、容错与恢复能力提出更高要求
- 推理场景需要在高并发、低时延与成本控制之间进行复杂权衡

在这一阶段，真正的挑战在于如何将异构、分散且高度复杂的底层算力资源，转化为对模型友好的、可持续输出的生产能力 —— 即通过 AI Infra，实现算力的高效利用、稳定运行与规模化扩展，真正将“算力”转化为“模型生产力”。

2.2. AI-Infra 的核心定位

正是在上述背景下，AI-Infra 成为大模型时代不可或缺的关键基础设施。

AI-Infra 并不是简单的 GPU 机器堆叠，也不是某一个独立的软件组件，而是一整套 **面向 AI 工作负载深度定制的系统工程体系**。其**核心目标**包括：

- **屏蔽异构硬件与复杂系统差异**，为模型提供统一、稳定的运行环境
- **协同算力、网络、存储与调度资源**，释放系统整体性能
- **支撑训练与推理的规模化运行**，从百卡走向千卡、万卡
- **通过平台化与产品化能力降低使用门槛**，提升交付与使用效率

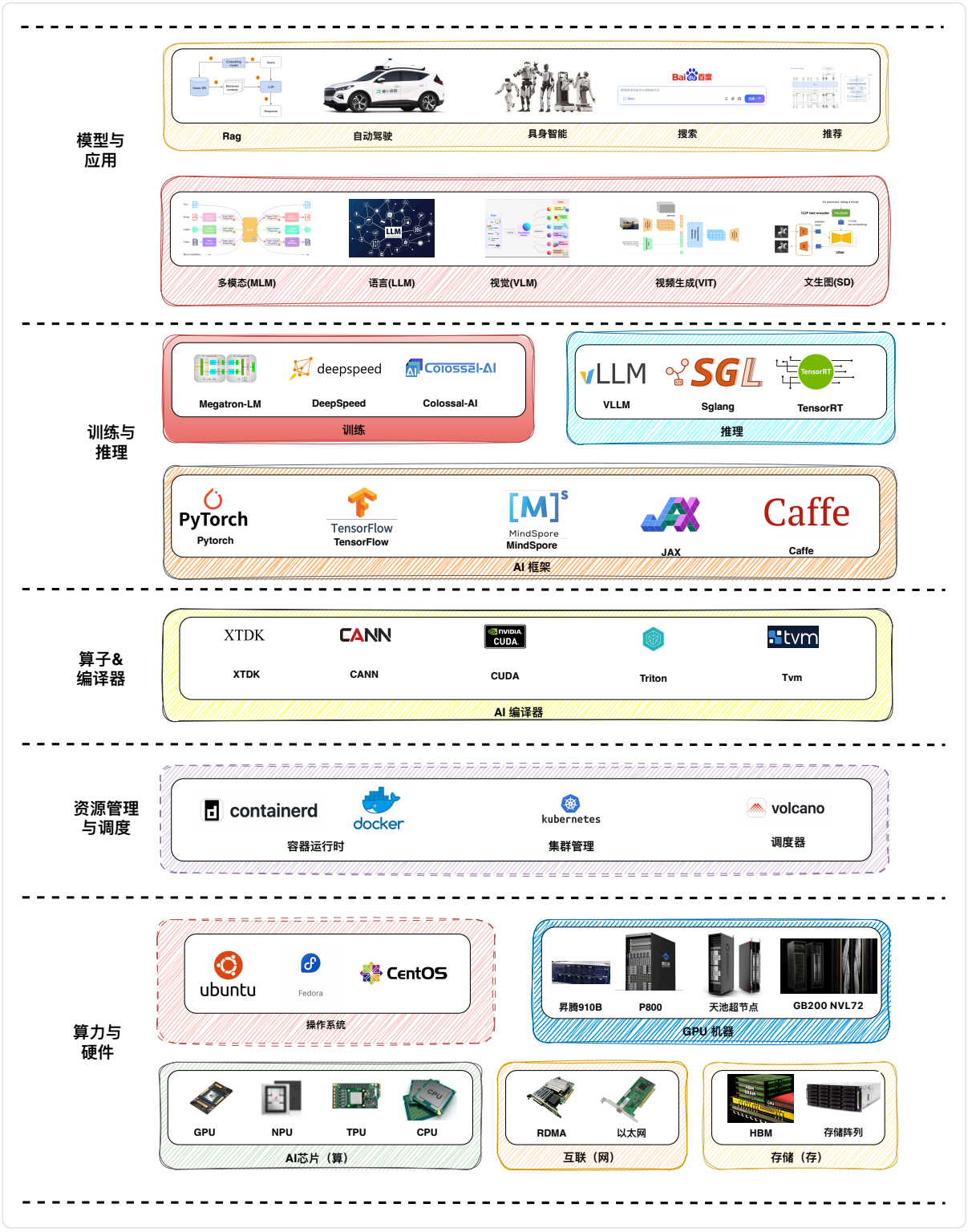
可以说，**AI-Infra 正是连接模型能力与业务价值的关键枢纽**。接下来，我们将围绕这一定位，重点介绍 **AI-Infra 的整体架构设计与核心能力拆解**，并探讨其如何在真实场景中解决大模型规模化落地的问题。

3. AI-Infra 架构全景：支撑训推的关键技术栈

从系统视角来看，成熟的 AI-Infra 并非单点能力的叠加，而是一套分层解耦、协同演进的整体系统。AI-Infra 通常可以拆解为五个相互依赖的核心层次，各层关注的问题不同，但共同决定了模型能力能否被高效、稳定、规模化地释放，如下所示：

- **算力与硬件层**：提供基础计算与带宽能力，决定模型规模与性能的物理上限

- **资源管理与调度层**：对算力进行统一抽象与调配，使其从“存在”走向“可用”、“好用”
- **算子与编译层**：将模型计算语义映射为高效可执行代码，决定算力是否能被充分释放
- **训练与推理系统层**：承载大规模训练与高并发推理，保障模型稳定、高效运行
- **平台与应用层**：通过平台化能力降低使用门槛，使 AI 能力真正转化为业务价值

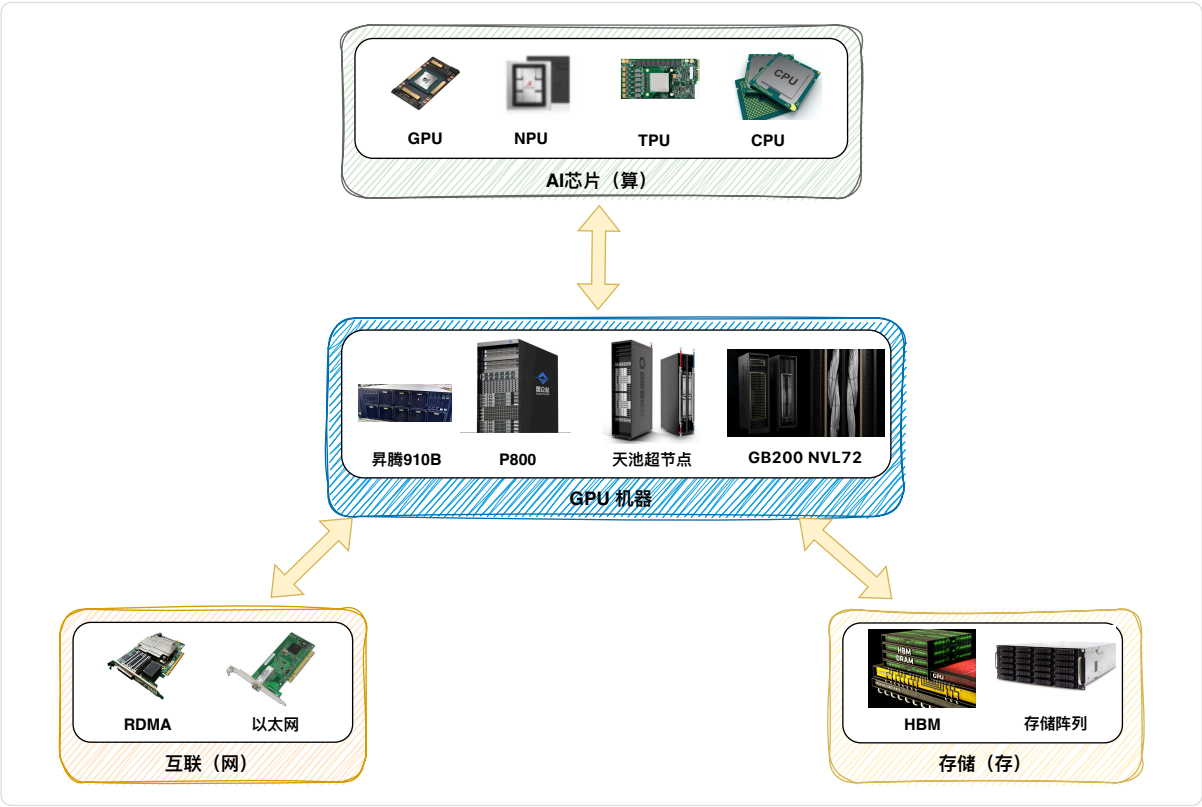


3.1. 算力与硬件层：决定模型能力上限

算力与硬件层是 AI-Infra 的 **物理基础**，是整个体系中唯一涉及到硬件的一层，其处于AI Infra体系的最下层，**决定了模型能力的上限**。主要关注：

- GPU/NPU 等异构算力并存
- 单机算力密度与多机横向扩展
- 高带宽、低时延的高速互联网络
- 面向大规模数据的高性能存储

这一层决定的是 **“模型能跑多大”**。他们的主要组织形式如下：



3.1.1. 算力概念

算力是衡量计算机处理信息能力的核心单位。算力水平用单位 FLOPS(Floating point Operations Per Second) 每秒浮点计算次数来衡量，以下是更多不同数量级的浮点运算单位，让我们对算力有一个直观的认识。

名称	Unit（单位）	Value（值）	中文描述	算力水平
Kilo FLOPS	KFLOPS	10^3	每秒一千次	0.000000
Mega FLOPS	MFLOPS	10^6	每秒一百万次	0.000001
Giga FLOPS	GFLOPS	10^9	每秒十亿次	0.001部
Tera FLOPS	**TFLOPS**	10^{12}	每秒一万亿次	一部 Mat
Peta FLOPS	**PFLOPS**	10^{15}	每秒一千万亿次	一千部（
Exa FLOPS	EFLOPS	10^{18}	每秒一百京次	一百万部
Zetta FLOPS	ZFLOPS	10^{21}	每秒十万京次	十亿部（
Yotta FLOPS	YFLOPS	10^{24}	每秒十亿京次	一万亿部

未来算力增长情况

在了解了算力的概念后，接下来我们按照“算—网—存”的顺序依次介绍算力与硬件层中的内容。

3.1.2. 异构芯片

AI 芯片是 AI Infra 体系的核心，**主要用于提供算力**。回顾第三节课程 【第三课】AI芯片&算子&编译器大观（上），介绍了芯片的发展历史，了解到 AI 计算早已不再是单一 GPU 形式，而是明显的异构算力并存的局面。常见的 AI 异构芯片主要有 CPU、GPU、NPU、DSP 和 FPGA，如下所示：

芯片类型	核心定位/比喻	主要优势	主要劣势/瓶颈	典型应用场景
CPU (中央处理器)	负责**复杂逻辑控制**和**任务调度**。	**1. 通用性最强：几乎能运行所有代码。** 2. 软件生态极度成熟。 3. 擅长处理复杂分支和控制流任务。	1. **并行度低**：不擅长海量重复计算。 2. 功耗墙/频率墙限制了主频提升。 3. 在AI重负载下效率低、发热严重。	操作系统运行、 务调度、数据预
GPU (图形处理器)	通过**海量线程**进行**大规模并行计算**。	**1. 极强的并行处理能力：适合矩阵运算和卷积。** 2. 算力远高于CPU。 3. 软件生态相对成熟（CUDA）。	1. 控制流程不如CPU灵活。 2. 功耗较大。 3. 存在“内存墙”问题（算力增长快于带宽增长）。	深度学习的**训练** 是云端/数据中心
NPU (神经网络处理器)	**专为AI算子设计**的ASIC（专用集成电路）。	**1. 极致的能效比（性能/功耗高）。** 2. 专为神经网络设计（如低精度运算、张量运算）。 3. 功耗低、速度快（尤其在边缘侧）。	**专用性强，灵活性差**：如果算法或模型架构发生巨大变化，可能无法适应。	手机、摄像头、 **推理**；云端专用 TPU）。
DSP (数字信号处理器)	数字信号处理专用硬件。	1. 针对音频、图像、通信等信号处理优化。 2. 低延迟，实时处理能力强。 3. 功耗/面积比优于通用核。	并行能力弱于GPU，专用性弱于NPU（处于中间态）。	移动SoC中的音 调制解调、嵌入
FPGA (现场可编程门阵列)	硬件电路在制造后可重新配置。	1. 极高的定制灵活性，硬件可重构。 2. 可针对特定算法设计专用电路。 3. 低延迟。	1. 开发难度大且复杂。 2. 功耗通常大于专用ASIC。 3. 性能不一定优于专用加速器。	小批量定制、边 研究验证、需要 任务。

从 AI-Infra 视角看，异构并存带来了算力的提升，但同时也带来了挑战：

- **算力不可替代性**：不同芯片对算子、精度、内存模型的支持差异巨大；
- **编程模型割裂**：CUDA、ROCm、各类 NPU SDK 并存，软件适配成本高；
- **调度复杂度上升**：同一个模型在不同硬件上的最优执行路径完全不同；

因此，异构芯片的普及，需要 AI-Infra 向更强的抽象层演进，以屏蔽底层差异，提供通用算力，异构计算资源的管理会在后续 3.3.1.章节 异构计算资源管理 中详细介绍。

在真实的大模型训练与推理场景中，**单卡再强，也无法独立支撑万亿参数训练或高并发在线服务**。所以模型规模**走向多卡、多机**，那算力之间如何高效协同？梯度、激活、KV Cache 如何在设备间高速传输？模型参数、Checkpoints 和中间状态如何被稳定、快速地读写？解决这些问题的关键，已不仅仅局限在芯片本身，更需要依托 **高性能网络与高性能存储的协同支撑**。接下来，我们将从系统视角介绍 **高性能网络与高性能存储**这两大核心组件。

3.1.3. 高性能网络

传统集群网络往往难以支撑大规模AI训练和高性能推理，下图是构建不通规模集群的通信量以及传统k8s网络的表现情况：

训练规模	每轮通信量（GB）	通信量增长倍数	通信瓶颈表现
100卡集群	约120	-	无明显瓶颈
1000卡集群	约**1200**	约**10倍**	**All-Reduce延迟增加5倍**
10000卡集群	约**12000**	约**100倍**	**训练停滞**

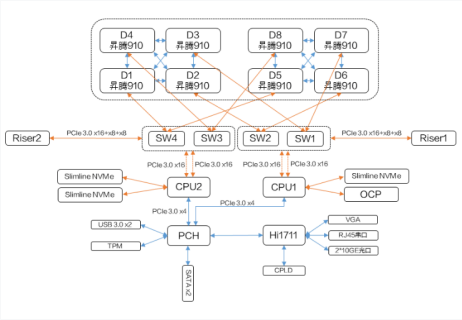
可以发现：当我们构建大规模AI训练（千卡/万卡）时，通信量及 All-Reduce 延迟会急剧增加甚至**近乎停滞**，导致训练效率**大幅降低**；因此，我们需要高性能网络来支撑我们大规模AI训练和高性能推理。

高性能网络的目标主要有三个：**大规模、高带宽、高稳定性**，需要解决的场景有两个：**单机机内通信和多机跨节点通信。**

3.1.3.1. 单机机内通信

高性能网络单机机内通信主要涉及的设备和技术如下：

GPU机器	单机机内通信

英伟达	**NVLink**	1. NVLink 是一种高速互连技术，旨在加快**CPU 与 GPU**，**GPU 与 GPU** 之间的数据传输速度，提高系统性能。 2. NVLink 通过 GPU 之间的直接互联，相较于传统 PCIe 总线可提供更高效，**是低延迟的互联解决方案**。 3. NVLink 首个版本于2014年发布，首次引入高速GPU互连。 4. **NVLink4.0**（Hopper）单通道带宽25 GB/s，双向总带宽速度 **900GB/s**，**NVLink5.0**（Blackwell）单通道带宽50 GB/s，达到 **1800GB/s**
昆仑芯	**XPULink** 	1. 昆仑芯服务器内部通过 **XPULink** 进行连接，卡和卡之间使用**XCCL** 通信库进行相互通信。 2. 百度-昆仑-Communication-Library（BKCL）通信库实现了常见的集合通信的接口，比如 bkcl_broadcast、bkcl_all_reduce等
昇腾	**HCCS** 	1. **HCCS 是华为自主研发的高速互联技术**，采用对等拓扑结构实现无交换芯片直连，允许芯片之间进行高速数据传输。该技术使得华为的AI芯片可以更加**高效地进行并行计算**，从而提升了整体计算性能。 2. **昇腾 910B** 的 HCCS 总带宽为**392GB/s**，和英伟达 A800 NVLINK的400GB/s 相当。 3. 支撑超大规模 AI 训练集群，通过 HCCS协议栈确保内存访问一致性，最多支持256物理核 NUMA 架构。

3.1.3.2. 多机跨节点通信

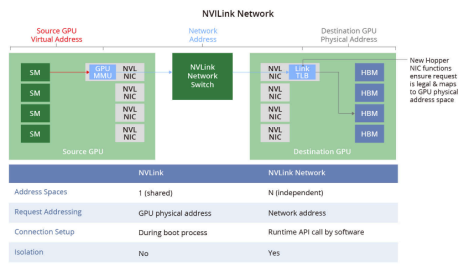
高性能网络多机跨节点通信主要涉及的设备和技术如下：

GPU机器	多机跨节点通信

英伟达

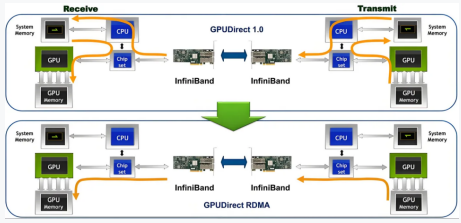
NVIDIA

NVLink交换机



NVLink 交换机是英伟达开发的 **ASIC 芯片级机架交换机**，通过高速互连技术实现多 **GPU 全对多通信**，支持AI训练和高性能计算（HPC）场景下的超大规模并行计算。

GPUDirect RDMA

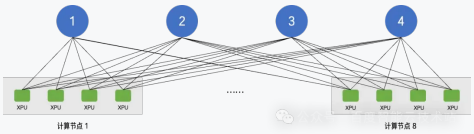


GPUDirect RDMA 结合 GPU 加速计算和 RDMA 技术，实现 GPU 和 RDMA 网络设备之间的直接数据传输和通信。该技术允许 GPU 直接访问 RDMA 网络设备中的数据，无需通过 CPU 或主机内存。

昆仑芯

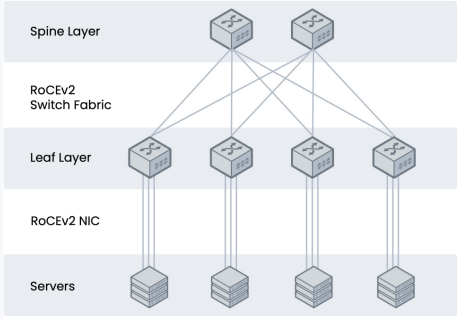
昆仑芯 KUNLUNXIN

超节点



昆仑芯超节点在设计上突破了传统单机 **8 卡互联**的架构限制，创新性地引入**多 Switch** 通信结构。以 32 卡为例，可以通过 4 台 **Switch Tray 模块**实现算力全互联，构建出一个 Scale-Up 域规模为 32 卡的统一算力池。

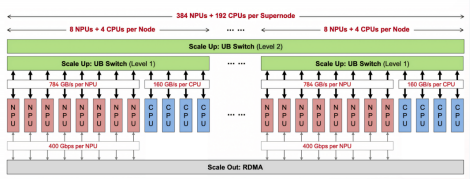
RDMA（IB Or RoCEV2）



昇腾

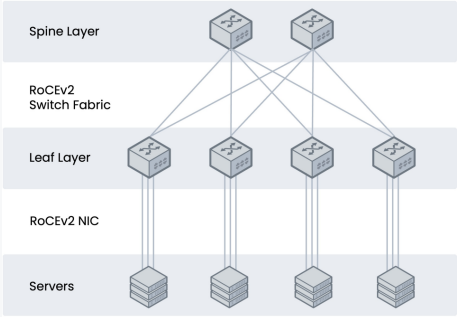
Ascend 华为昇腾

超节点

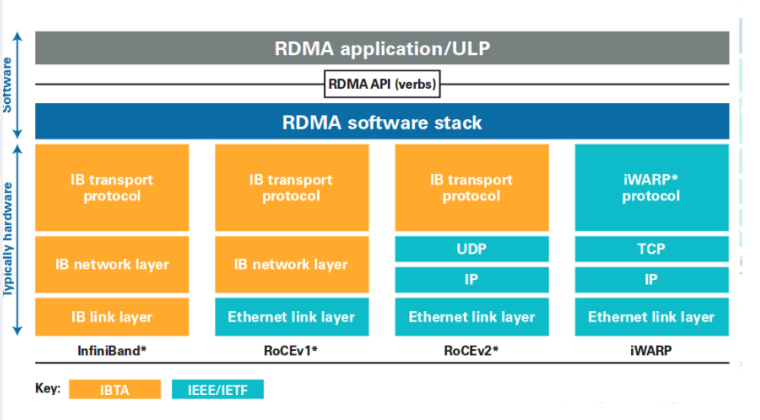


Atlas950 SuperPoD 支持 8192 张昇腾卡，采用华为自研的“灵衢（UnifiedBus）”互联协议，实现了超节点内部的**全对等互联**。其互联带宽高达 16.3 PB/s，将卡间时延降至纳秒级别，并通过液冷方案解决了单机柜散热难题。

RDMA（IB Or RoCEV2）



这里深入介绍下 RDMA。**RDMA（Remote Direct Memory Access）是远程直接地址访问**，通过 RDMA，本端节点能直接访问远端节点的内存。RDMA本身指的是一种技术，具体实现包含 **IB（Infiniband）**，**RoCE（RDMA over Converged Ethernet）**和 **iWARP（internet Wide Area RDMA Protocol）**。



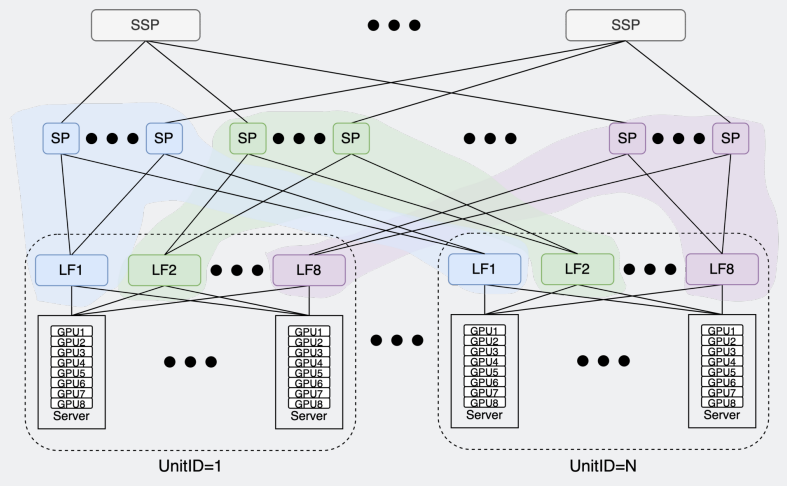
主要说明：

- RDMA比传统的网络通信有更低的延迟、更高的带宽和更低的CPU利用率，可以显著提高网络通信的性能和效率
- **IB(InfiniBand)**：基于InfiniBand架构的RDMA技术，由IBTA(InfiniBand Trade Association)提出。搭建基于IB技术的RDMA网络需要专用的IB网卡和IB交换机。
- **RoCE(RDMA over Converged Ethernet)**：基于以太网的RDMA技术，也是由IBTA提出。RoCE支持在标准以太网基础设施上使用RDMA技术，但是需要交换机支持无损以太网传输，需要服务器使用RoCE网卡，目前有V1和V2两个版本。
- **iWARP(Internet Wide Area RDMA Protocal)**：基于TCP/IP协议的RDMA技术，由IETF标准定义。iWARP 支持在标准以太网基础设施上使用RDMA技术，但服务器需要使用支持iWARP的网卡。

	特性维度	IB（Infiniband）	RoCE (RDMA over Converged Ethernet)	iWARP（interr
2	**协议本质**	专为HPC设计的独立网络（**不是以太网**），原生支持RDMA	在**以太网**上承载RDMA，需要特殊网卡和交换机支持	在TCP/IP协议栈
3	**网络要求**	专用InfiniBand交换机和线缆，构成独立、无丢包的网络	需要无损以太网，依赖PFC、ECN等流控技术避免丢包，对交换机配置要求高	运行在标准TCP过TCP重传）
4	**性能特点**	**延迟最低**（亚微秒级），吞吐最高，稳定性极佳	**延迟较低，延迟和吞吐接近InfiniBand**（微秒级），但在拥塞时可能波动	延迟最高（受T
5	**CPU开销**	**极低**，完全旁路内核和CPU	**极低**，同样旁路内核	**相对较高**，因需
6	**部署与成本**	总拥有**成本最高**，需专用设备，但性能无妥协。生态由NVIDIA主导	**成本低**，可利用现有以太网架构升级，是“性能与成本”的平衡之选	成本适中，兼容**者取代**
7	**主要优势**	极致性能与稳定性，是大规模AI训练和HPC的黄金标准	在追求高性能的同时，复用以太网生态，部署灵活性高	无需改造网络，
8	**主要劣势**	**封闭生态**，成本高昂，与现有IP网络管理工具不统一	配置复杂，需精心设计无损网络，大规模下有拥塞管理挑战	性能无法满足前
9	**典型应用场景**	超大规模AI训练集群、顶级HPC超算、金融极速交易	企业级AI训练/推理集群、高性能存储、虚拟化数据中心	对延迟不敏感但

3.1.3.3. HPN网络架构

当集群规模扩展到**数以万计的节点**时，一个更根本的挑战随之出现：如何设计一个高效的“网络架构”，将这些高速链路组织起来，从而在整个数据中心尺度上**释放其最大价值**？这是**面向高性能计算的HPN网络架构**所要回答的核心问题。



感知GPU节点的HPN网络架构

概括说明：

- HPN（High Performance Network）网络是专门为大规模 AI 训练 / 推理场景设计，核心是通过多级**CLOS架构**（一种多级交换机网络结构）和**轨道优化**（Rail-Optimized）实现网络的水平扩展、资源高效利用、业务隔离与低延迟高可用的服务能力。
- SSP（SuperSpine）：**骨干连接层**，核心作用是**打通不同 Unit 之间的通信**，使得 Unit 1 的 GPU 可以以全带宽访问 Unit N 的 GPU。
- SP（Spine）：**集群转发层**，按照 GPU 索引 进行颜色分组（如蓝色轨道专供所有节点的 GPU1 通信）。这种设计确保了集合通信（Collective Communication）流量在物理轨道内隔离，彻底消除跨轨道干扰。
- LF（Leaf）：**单元接入层，**Unit 内的接入交换机。`LF1` 至 `LF8` 对应服务器的 8 个网口，实现流量的首次汇聚。
- Server（GPU 节点）：**计算资源层，**每个 Server 带 8 张 GPU（GPU1~GPU8），每张 GPU 通过独立的 HCA 网卡接入网络，是实际执行训练 / 推理任务的硬件载体。
- 采用**轨道优化**（Rail-Optimized），有效降低哈希选路冲突风险。

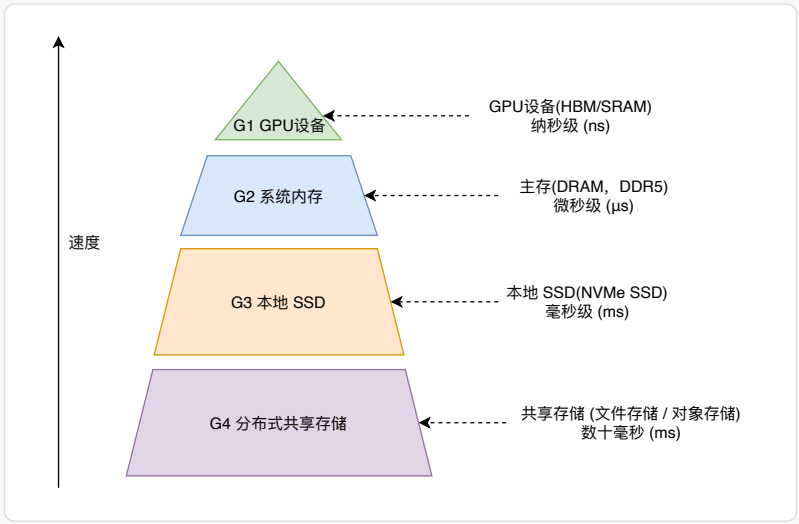
3.1.4. 高性能存储

AI 任务对存储的要求如下：

概括总结

1. 模型规模大，需要支持海量存储，**要求存储容量大**
2. 训练时间长，要求基础设施提供**高性能和长时间的稳定**
3. 大模型要结合具体应用才能发挥巨大的威力，需要存储具备大规模的**敏捷部署**能力
4. 大模型离不开持续更新的海量数据，需要让**数据能在各个环节便捷地流动**
5. 训练和推理需要支持**高并发，高吞吐、低延迟和高效率**

我们通常使用的存储主要如下：

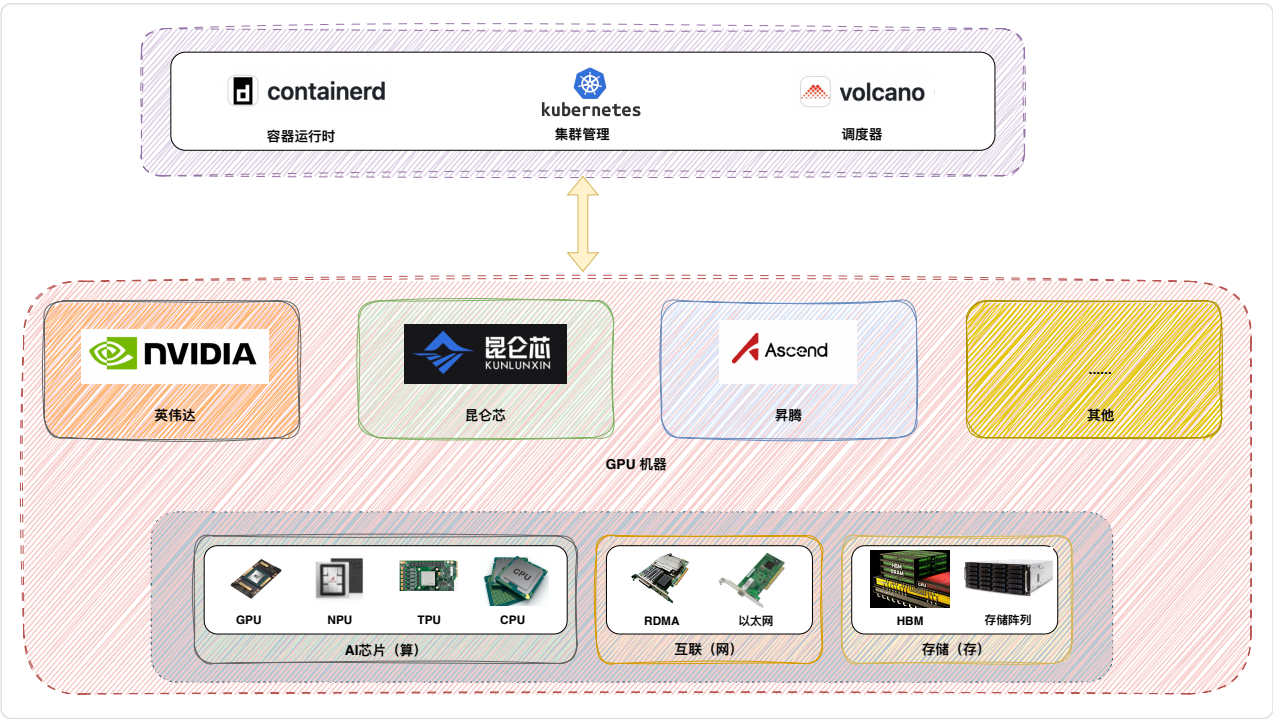


概括总结

1. 面向 **AI 大模型场景** 的存储通常使用多层分级架构，核心围绕**KV 缓存管理、长上下文推理**等核心需求，遵循**速度递减、容量递增、成本递减**的存储金字塔原则，最终实现**性能与成本**的最优平衡。
2. 数据按访问热度从冷到热**向上迁移**（G4→G3→G2→G1），从热到冷**向下下沉**（G1→G2→G3→G4），热度越高，驻留的存储层级越靠前，成本越高，容量越小；
3. 所有需要**实时参与计算**的低延迟敏感数据，优先分配 G1/G2 空间，仅将非实时、长周期数据分配至 G3/G4；

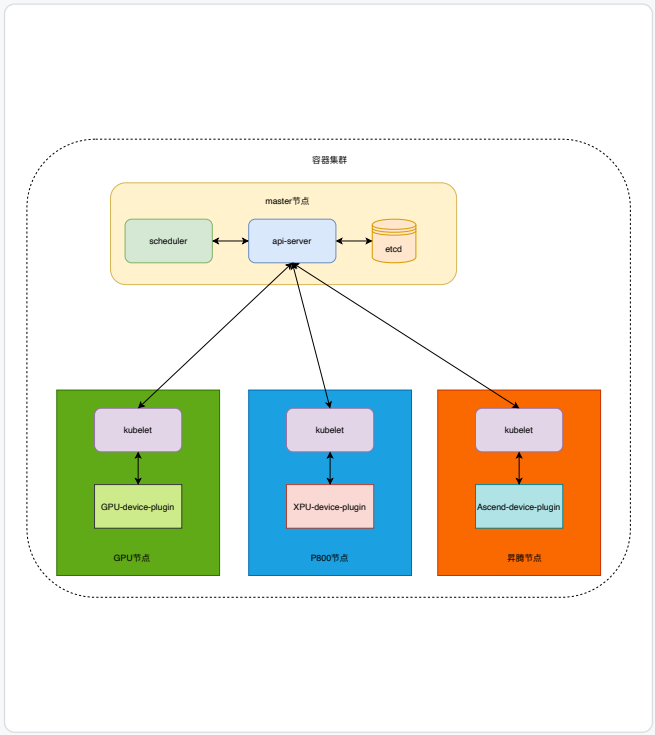
3.2. 资源管理与调度层：让算力真正可用

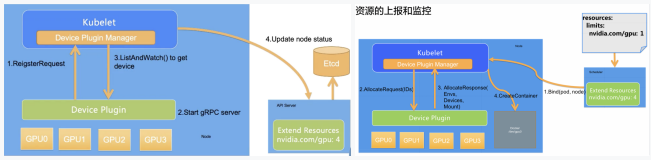
随着训练走向**多机多卡、推理走向多模型和高并发**，我们更加关注**谁在用算力、何时使用、用多少以及如何隔离与抢占。资源管理与调度层**正是通过统一抽象、调度、隔离和弹性机制，将分散的硬件资源转化为稳定、可规模化的算力池，为上层训练与推理系统提供可靠支撑。通常采用**容器化技术**与 **Kubernetes 集群管理**的架构思想，构建高性能容器集群，以提升算力资源利用率。



3.2.1. 异构计算资源管理

我们一般通过在节点上部署硬件专属的 Device Plugin，让 Kubernetes 能感知和调度 GPU、XPU、昇腾等异构资源，进而支撑 AI 训练和推理，主要工作原理如下：





device-plugin工作原理 实际调用过程

```
$ kubectl describe node gpu-node-01

apiVersion: v1
kind: Node
metadata:
    name: node-1
...
Status:
    Capacity:
        cpu:                4
        memory:             15234152Ki
        nvidia.com/gpu:     2
```

```
apiVersion: v1
kind: Pod
metadata:
    name: cuda-vector-add
spec:
    restartPolicy: OnFailure
    containers:
        - name: cuda-vector-add
          image: "nvidia/cuda-vector-add:v0.1"
resources:
          limits:
            nvidia.com/gpu: 1
            memory: "512Mi"
            cpu: "250m"
```

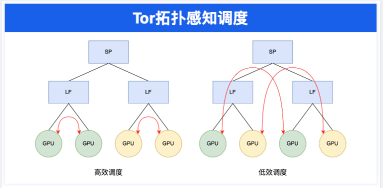
node节点信息 pod使用gpu资源

主要过程：

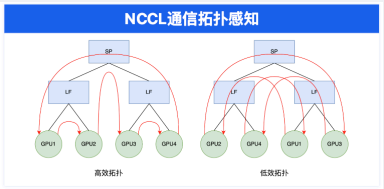
插件注册 -> 启动服务 -> 设备发现 -> 资源上报 -> Pod 请求资源 -> 调度与分配

3.2.2. 高性能网络增强

在本文 3.1.3. 高性能网络 章节中介绍了可以通过高性能网络来支撑大规模AI训练和高性能推理。在实际落地中，我们通常会结合**Tor 拓扑和 NCCL 通信拓扑持续优化调度策略**，以此进一步提升网络通信性能。



Tor拓扑感知调度



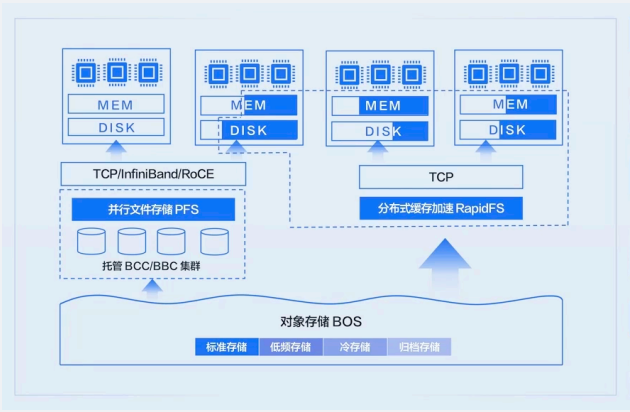
NCCL通信拓扑感知

概括总结

1. 主机内使用NVlink、XPUlink和Hccs等高速互联技术，主机间跨机通信使用RDMA
2. Tor 拓扑感知调度考虑GPU拓扑信息，包括连接类型（如 NVLink、PCIe）、带宽、NUMA亲和性等，提升网络性能
3. NCCL 通信拓扑感知通过 NCCL 内部的系统拓扑结构（CPU、GPU、PCIe、NIC、NVLink等）为后续的通信算法（如 Ring、Tree、CollNet 等）初始化搜索辅助结构，提升网络性能

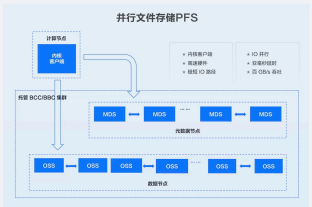
3.2.3. 存储资源管理

常见的高性能存储使用和管理架构如下：

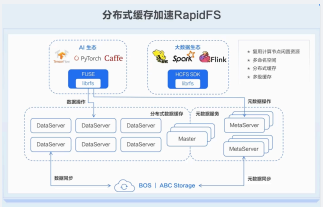


概括总结

1. 存储底座是成本较低的bos，具备大容量、高吞吐、低成本
2. 使用PFS、RapidFS提供最好的I/O性能
3. PFS是一个典型的并行文件系统，其主要适用于AI训练、高性能计算、视频渲染等需要**高吞吐、低延迟、稳定IO性能**的场景
4. RapidFS 是一个缓存加速系统，主要适用于为大数据计算、AI训练等访问**对象存储数据提供加速**的场景

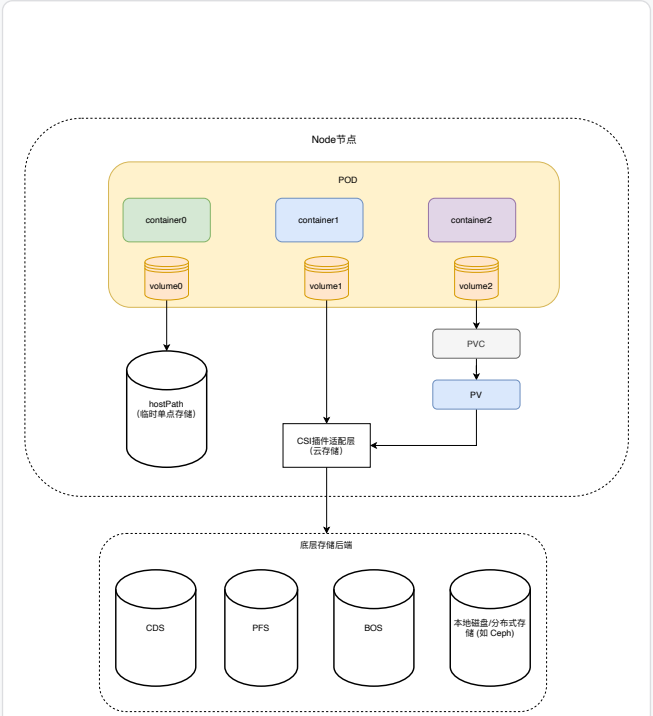


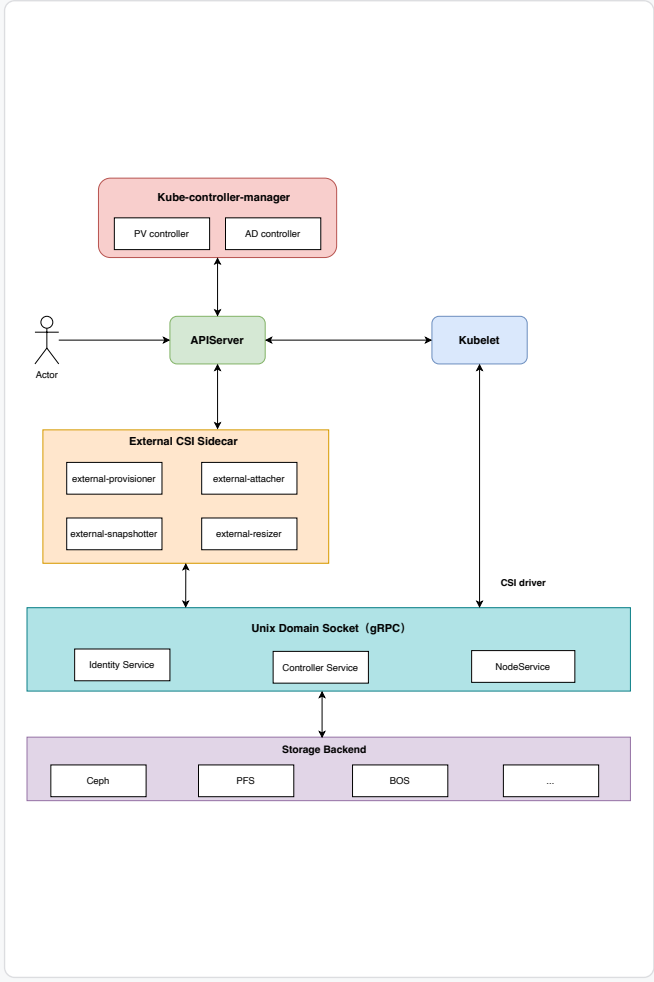
pfs介绍



rapidfs介绍

Kubernetes 通过将存储资源抽象为**存储卷（Volume）**，用于为 Pod 提供临时或持久的存储。由于kubernetes支持的存储系统有很多，为了能够屏蔽底层存储实现的细节，方便用户使用， kubernetes引入 PV 和 PVC 两种资源对象，并且通过CSI（Container Storage Interface）定义行业标准"容器存储接口"，解耦存储逻辑与 K8S 核心，使得 SP（Storage Provider）能够根据标准接口开发一个符合 CSI 标准的存储插件，提供用户使用，CSI主要原理如下：

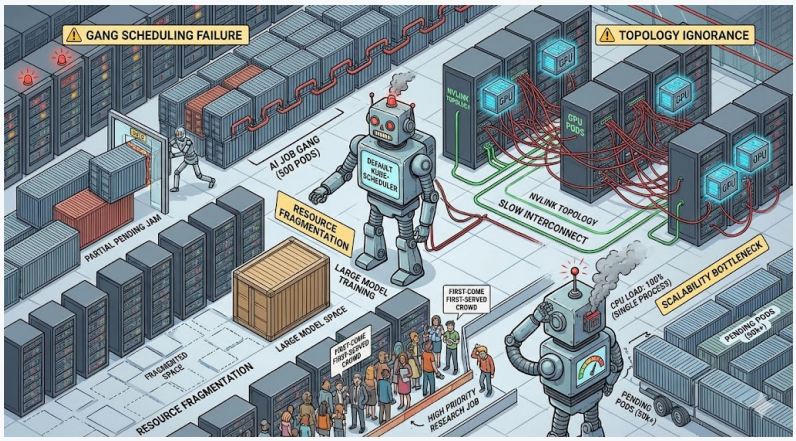




3.2.4. 任务调度

3.2.4.1. 为什么需要专用调度器

传统k8s集群默认调度器（kube-scheduler）是为通用工作负载设计的，在大规模训练和推理场景下确实存在诸多不足，主要原因如下：



传统调度器无法对于AI任务存在不足

主要原因：

1. 资源调度粒度不足
   - 无法感知 Job 层面的需求
   - 缺乏 gang 调度
2. 缺乏拓扑感知和硬件亲和性
   - GPU/NPU 拓扑不感知
   - 无网络拓扑调度
3. 资源碎片化问题严重
   - Bin-packing算法简单
   - 缺乏全局视图
4. 作业排队与公平性缺失
   - 没有队列概念
   - 缺乏抢占策略
5. 弹性与动态调度能力不足
   - 不支持弹性训练
   - checkpoint恢复困难
6. 扩展性和性能瓶颈

3.2.4.2. 专用调度器工作原理

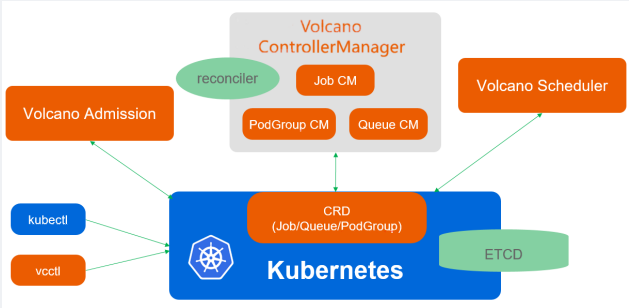
在大规模训练场景下，我们需要调度器能够实现**协同调度**、**精细化资源管理**和**高吞吐量的批处理**及**弹性工作负载**，目前云原生社区能够满足需求的调度框架主要有**Volcano**、**Kueue** 和 **Koordinator** 等。

Volcano	Kueue	Koordinator
VOLCANO	Kueue	Koordinator
主要特征 1. 一站式重型批处理调度系统，CNCF 维护 2. 功能全面，支持 Gang 调度、队列管理、拓扑感知、多框架支持、优先级/抢占等 3. 支持 GPU/NPU 等异构资源管理、支持拓扑亲和调度 4. 主要用于大规模AI训练、HPC、需要复杂调度策略的批处理场景 5. **支持几乎所有的主流计算框架** 6. **AI集群中使用最多**	**主要特征** 1. 轻量级的作业队列控制器 2. 非侵入式，不替代 kube-scheduler，通过准入控制管理作业排队与准入 3. 核心是基于配额的作业排队与准入，确保资源在租户间公平共享，并支持优先级和拓扑感知 4. 主要用于需要为通用批处理/AI作业添加队列和配额管理的云环境 5. AI集群中使用较少	**主要特征** 1. QoS驱动的混部调度增强组件 2. 实现在线服务与离线作业的安全混部，提升资源利用率 3. 通过资源画像、CPU编排、内存隔离等机制，保障AI等批处理作业与在线服务混部时的性能与效率 4. 主要用于追求极致资源利用率、需要进行在线离线业务混部的大型生产集群环境 5. AI集群中使用较少

volcano 因其支持场景丰富，功能全面，**是使用最多的专用调度器。**这里主要介绍下 volcano 调度器的工作原理和调度策略。

Volcano调度器

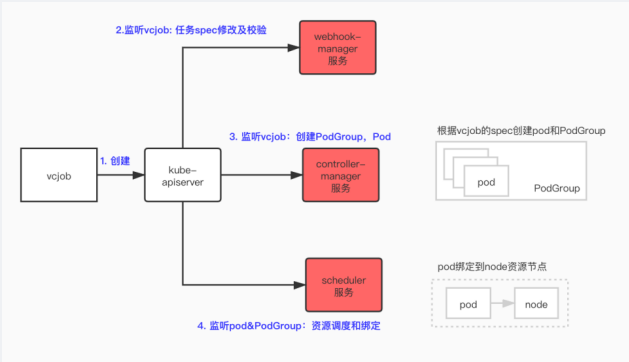
整体架构图



概括总结

1. 由crd、volcano-scheduler、volcano-controller-manager、volcano-admission和vcctl五部分组成
2. Volcano利用声明式的CRD定义我们的API，主要有3个核心的API，分别是 `Job（Volcano Job）` 、`PodGroup` 、`Queue`
   - Job：是对高性能任务的通用定义，它是批量计算作业的定义，支持定义作业所属队列、生命周期策略、任务模板以及持久卷等信息；
   - PodGroup：提供了Job 中 Task 的管理能力，是任务的分组，它与queue绑定，占用队列的资源，它与 Volcano Job 是一对一的关系；
   - Queue：为任务的分类提供了基础，是Cluster 级别的资源对象，可为其声明资源配额，也可由多 namespace 共享，并且提供软隔离；
3. Volcano Scheduler 通过一系列的 action 和 plugin 调度Job，并为它找到一个最适合的节点，支持针对Job的多种调度算法。
4. Volcano ControllerManager 管理 CRD 资源的生命周期。

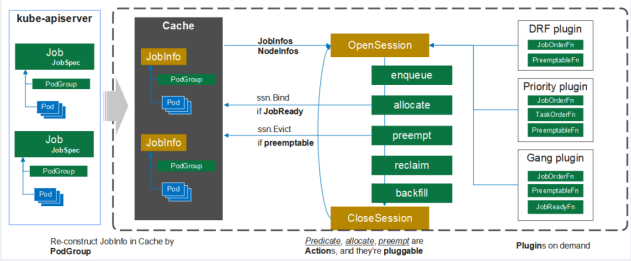
一个训练任务的创建过程



任务调度流程

1. 用户创建一个Volcano作业（Job）
2. Volcano Admission 拦截作业的创建请求，并进行合法性校验
3. Kubernetes 持久化存储Volcano Job到 ETCD
4. ControllerManager 通过 List-Watch 机制观察到 Job 资源的创建，创建任务（Pod）
5. Scheduler 负责任务的调度，绑定 Node
6. Kubelet Watch 到 Pod 的创建，接管 Pod 的运行
7. ControllerManager 监控所有任务的运行状态，保证所有的任务在期望的状态下运行

scheduler调度过程



scheduler调度过程

1. Volcano scheduler 的工作流程如下：
   a. 客户端提交的 Job 被 scheduler 观察到并缓存起来。
   b. 开启 session，即一个调度周期开始。
   c. 将没有被调度的 Job 发送到会话的待调度队列中。
   d. 遍历所有的待调度 Job，按照定义的次序依次执行 enqueue、allocate、preempt、reclaim、backfill等action，为每个Job找到一个最合适的节点。将该Job 绑定到这个节点。action中执行的具体算法逻辑取决于注册的plugin中各函数的实现。
   e. 关闭本次会话。
2. volcano-schedule 由一系列 action 和 plugin 组成，action 定义了调度各环节中需要执行的动作，plugin 根据不同场景提供 action 中算法的具体实现细节
3. Volcano Scheduler 具有高度的可扩展性，可以根据需要实现自己的action和plugin

volcano 常用调度策略主要有：

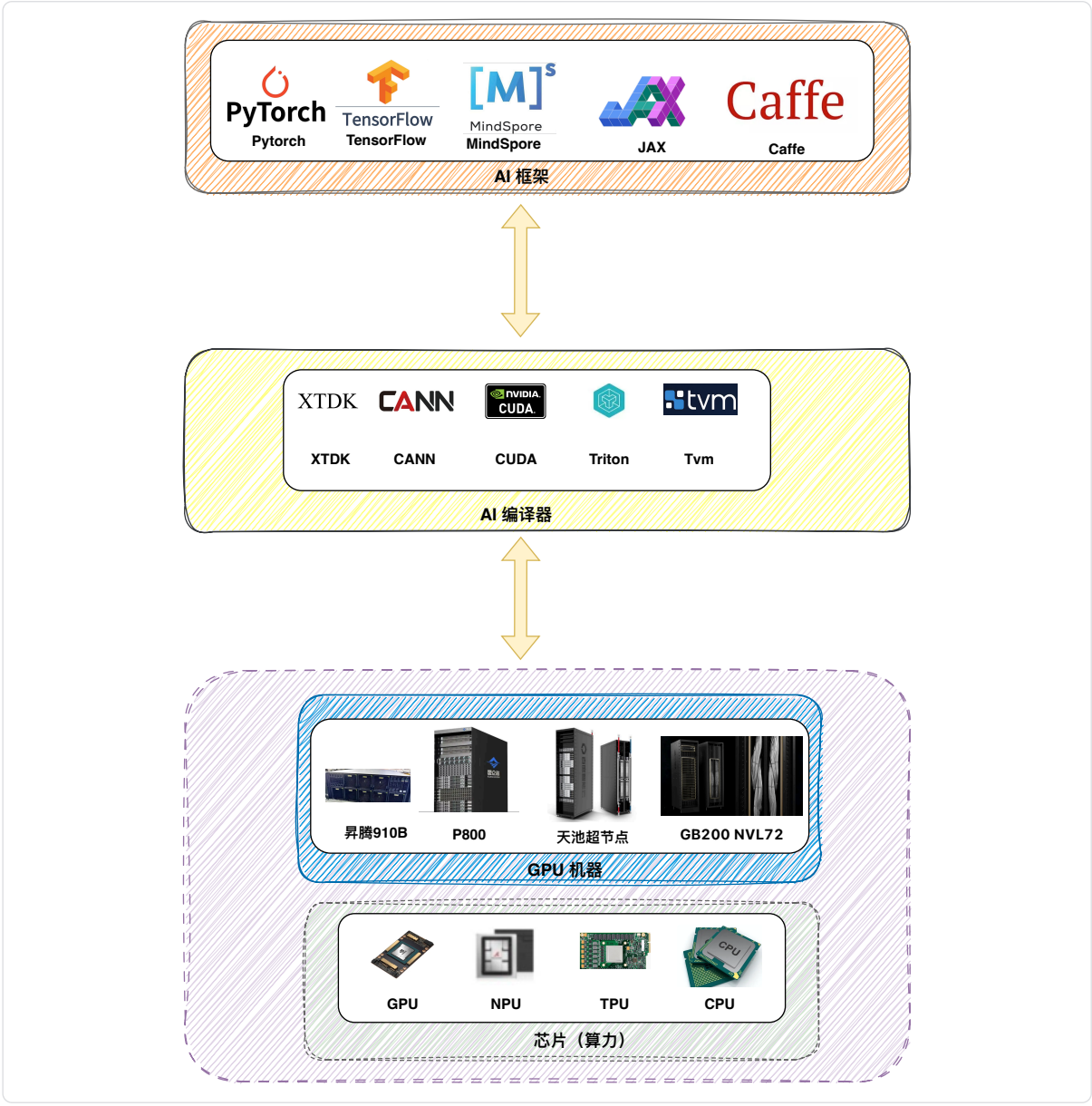
调度策略	核心功能和描述
Gang调度（组/批调度）	• 训练任务同时启动，避免资源死锁问题，保证 ps 和 worker 同时运行。 • 在资源不足的情况下，能保证最少的作业正常运行，在资源充足的情况下，可以运行更多的弹性训练任务。
binpack调度	• 尽可能将资源填充到少数节点，能够尽可能减小节点内的碎片，提高单节点利用率。
spread调度	• Spread算法防止了某些节点过载而其他节点空闲的情况，保证调度均衡。 • gpu卡出现故障时，Pod 分散在多个节点上可以提高整个系统的可用性和容错能力。
GPU/拓扑感知调度（拓扑感知调度）	• 调度时考虑GPU拓扑信息，包括连接类型（如 NVLink、PCIe）、带宽、NUMA亲和性等。
资源抢占调度	• 高优先级任务可抢占低优先级任务的资源（需开启抢占开关），抢占时会优雅终止低优先级Pod。
Backfill回填调度	• 高优先级任务因资源不足等待时，调度器会在不影响高优先级任务的前提下，调度低优先级任务利用空闲资源，提升集群资源利用率。
Priority优先级调度	• 支持为队列或任务配置优先级，高优先级任务/队列优先获得资源，适配“紧急任务优先执行”场景（如线上模型迭代训练）。
通信亲和调度（TOR调度）	• 优化网络通信延迟与带宽，减少跨机架、跨交换机的网络跳数。 • 训练任务内的多个worker 尽可能分布在同一个TOR下，如果跨TOR，尽可能优先占满同一TOR。 • 训练任务内的多个worker，按照worker顺序分布，相邻顺序的worker尽可能分布在同一TOR。
DRF调度	• 根据主导资源尽可能公平调度，不会因为一个胖业务，饿死大批小业务。

3.3. 算子与编译层：模型如何真正跑在硬件上

算子层是连接模型与硬件的一道系统级抽象，承担着 **承上启下的关键作用，**向上承接模型计算语义与框架执行逻辑，向下映射并调度底层硬件能力，将高层算法需求高效转化为可执行的硬件指令。算子层主要功能如下：

- 向下：承接 GPU / 加速器能力
- 向上：决定训练和推理系统的执行效率
- 算子层的作用：把“模型结构”变成“可调度的执行单元”

如果说硬件决定“能跑多快”，那么算子和编译层决定“模型到底是怎么跑的”，其在AI-Infra体系中的作用如下：



3.3.1. 算子：执行范围，而不是公式

算子（Operator，简称 Op）是人工智能（AI）计算体系中**不可再分的最小计算单元**，是构成 AI 模型计算逻辑的基础“原子操作”。通俗地说，算子就是实现单个操作的函数或内核，比如数学运算中的 `加法（Add）` 、 `乘法（Mul）` 、 `矩阵乘法（MatMul）` 。

算子兼具双重属性：

- 具备明确的**数学语义**，该部分定义了其应当执行的逻辑与功能。
- 拥有**具体的实现内核**，即在不同硬件设备（如 CPU、GPU、NPU、TPU 以及各类专用加速器）上实际运行该算子的代码。

在 **AI 训练**中**算子是训练全流程的计算执行载体**，贯穿每一个环节；在 **AI 推理**服务中**算子是推理的核心执行单元，且极致追求效率。**

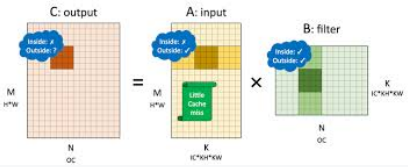
3.3.2. 常见算子及其执行特点

按性能瓶颈的不同，可分为三类，它们的核心差异在于计算、访存、通信三者的资源消耗占比：

计算密集型	访存密集型	分布式协同型

特点：

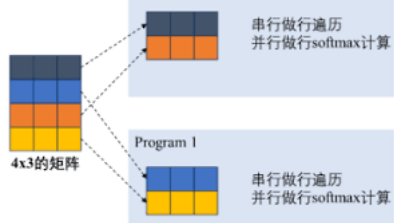
计算多、算力吃紧（如 GEMM、Attention、Linear、Convolution和FFT等）



GEMM

特点：

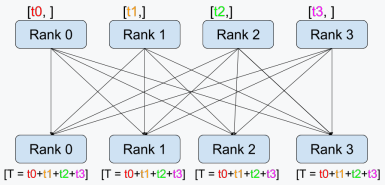
计算少、数据搬运多（如 LayerNorm、Embedding和SoftMax）



SoftMax

特点：

在分布式场景下，需要跨卡、跨机协同执行（如 AllReduce、AllGather和Broadcast等）

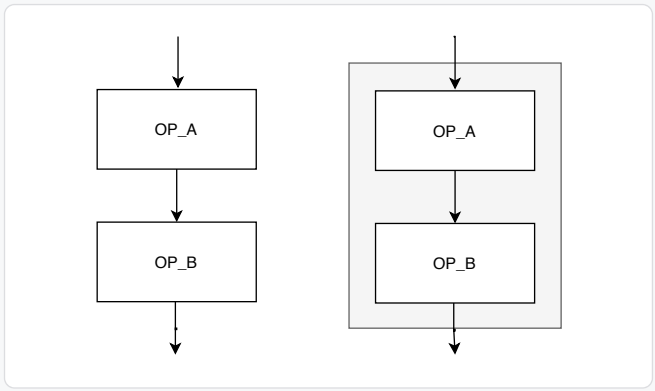


AllReduce

3.3.3. 算子融合

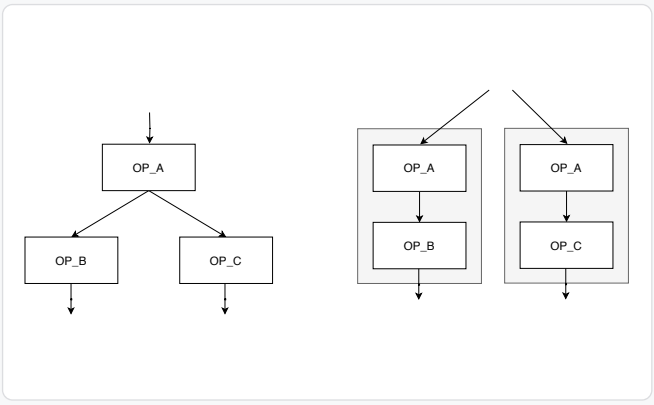
以上三类算子各有瓶颈，且多算子组合的核心痛点，不在于 “单个算子跑不快”，而在于 “**算子之间的衔接成本太高（算子组合会放大访存开销）**”，算子融合正是解决这一痛点的关键优化技术。

内存墙



主要是访存瓶颈引起，算子融合主要通过对计算图上存在数据依赖的“生产者-消费者”算子进行融合，从而提升中间 Tensor 数据的访存局部性，以此来解决内存墙问题。

并行墙



主要是由于芯片多核增加与单算子多核并行度不匹配引起。可以将计算图中的算子节点进行并行编排，从而提升整体计算并行度。特别是对于网络中存在可并行的分支节点，这种方式可以获得较好的并行加速效果。

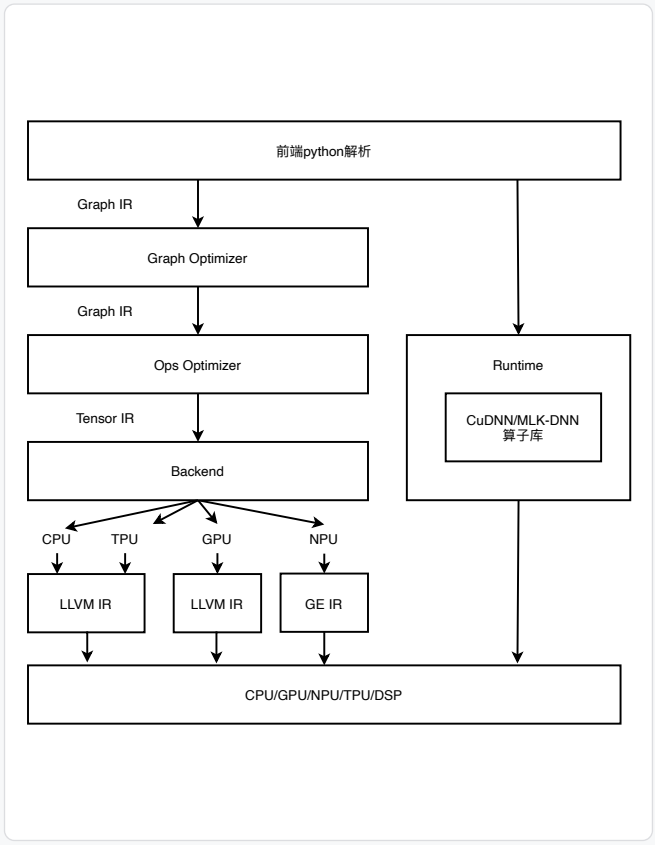
融合算子旨在**缓解模型训练中的“内存墙”与“并行墙”瓶颈**。其核心原理是通过将多个细粒度算子合并执行，显著**减少数据读取量与中间结果的频繁写回**，从而降低访存开销，提升计算效率。

3.3.4. AI 编译器

AI 编译器**是一种领域特定的编译器，专门用于将神经网络模型**（通常由Python等高级语言定义）**转换为能够在各种硬件设备上高效执行的代码**（例如AI芯片，XPU，NPU，GPGPU，SOC等）。AI编译器的诞生源于人工智能技术快速发展与传统编译器局限性之间的核心矛盾：

- **静态优化 vs 动态需求：**传统编译器（如GCC、LLVM）主要针对通用CPU进行**静态代码优化**，而AI计算依赖大规模并行计算、特殊数据类型（如低精度张量），且计算图结构常**动态变化**（如动态形状、条件分支），传统优化策略难以高效适配。
- **硬件异构化挑战：**AI 芯片爆发式增长（GPU、NPU、TPU、FPGA等），各硬件指令集、内存架构差异巨大。传统编译器需为每个硬件重写后端支持，开发成本极高，且难以实现跨平台性能移植。
- **计算范式革新：**AI 模型以**计算图**为核心抽象，而非传统线性代码。传统编译器缺乏对图结构、算子融合、自动微分等 AI 特定语义的原生支持。

AI编译器与传统编译器不同之处在于AI编译器通常以AI框架（如PyTorch、TensorFlow、Paddle、VLLM、SGLang）提供的计算图作为输入起点，其主要工作原理如下：



概括说明：

- **编译器前端（Frontend）**：对 Python 代码进行解析，将高层次的代码转换为一个中间表示（IR），以便进一步处理。
- **Graph Optimizer（图优化）**：接收到 Graph IR 后，会对解析后的计算图进行优化。优化的目的是减少计算图的冗余部分，提高执行效率。这可能包括算子融合、常量折叠等技术。
- **Ops Optimizer (算子优化)**：将图拆解为具体的算子，并进行算子生成以及算子级别的优化。
- **Backend (后端适配)**：针对具体硬件生成底层代码。
  - **LLVM IR**：通用的编译器基础设施，支持 CPU、TPU 和 GPU。
  - **GE IR (Graph Engine IR)**：通常指华为昇腾（Ascend NPU）专用的图引擎中间表示。
- **Runtime (运行时)**：不进行复杂的全局编译，而是根据 Python 指令直接调度硬件资源。
- **CuDNN / MLK-DNN (现一类名为 oneDNN) 算子库**：
  - 这是硬件厂商（NVIDIA/Intel）提前写好的、性能已经“磨到极致”的函数库。
  - *逻辑*：当 Python 说要算个卷积，Runtime 就直接去库里捞一个写好的 C++/CUDA 算子丢给 GPU 执行。
- **底层硬件抽象层**：CPU/GPU/NPU/TPU/DSP等异构芯片。

有关算子与编译器的更多详细信息可参见 【第三课】AI芯片&算子&编译器大观（下）

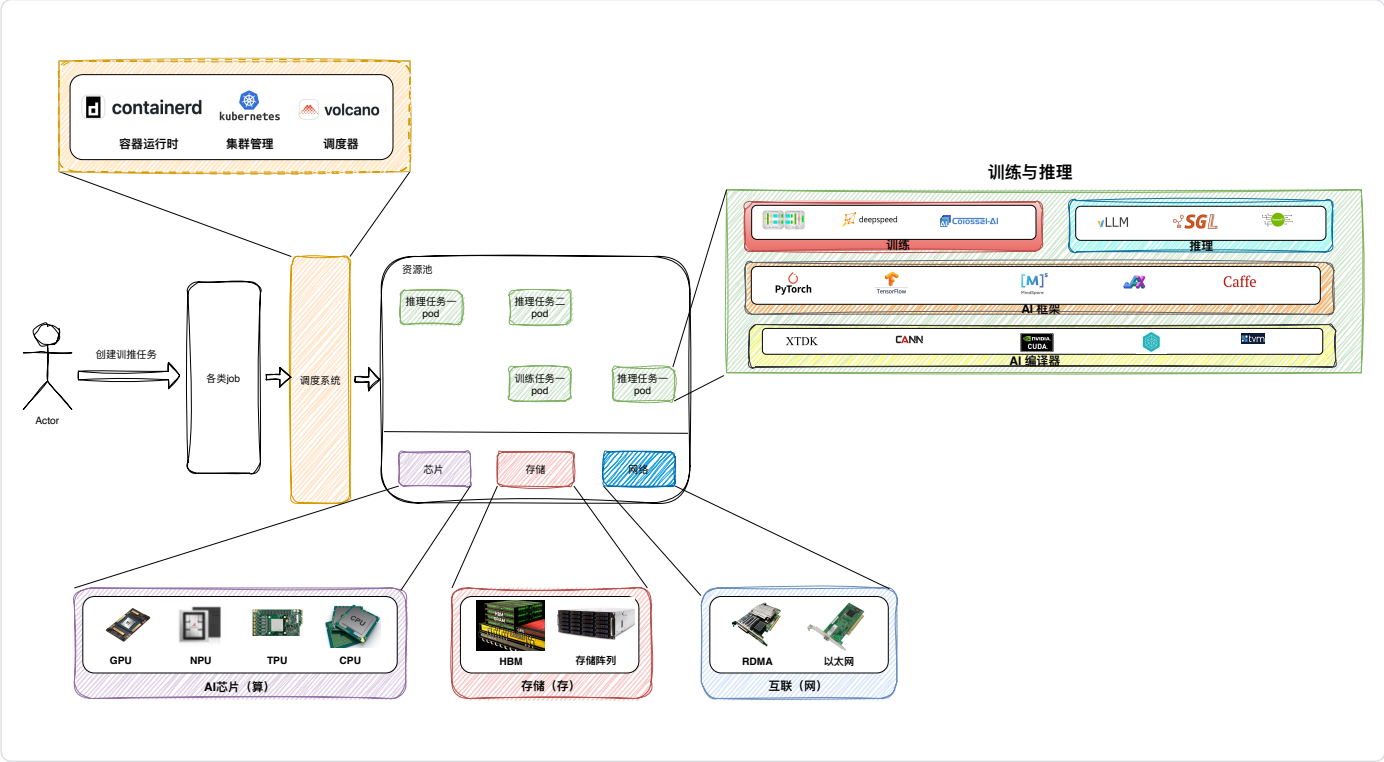
3.4. 训练与推理系统层：托举模型规模化运行

在硬件、算子与调度之上，真正让大模型“跑起来、跑得稳、跑得起规模”的，是训练与推理系统层。这一层承接底层硬件能力与上层模型需求，聚焦模型任务如何被系统高效承载，如何让模型真正做到**跑得稳、跑得快、跑得久**。

它通过解决分布式训练、推理服务化以及性能与成本平衡等关键工程问题，将“可用算力”转化为可持续交付的模型能力，成为大模型规模化落地的最后一道托举。

- 在训练侧，核心挑战在于如何在大规模集群中高效调度并稳定运行长时间训练任务；
- 在推理侧，则关注如何构建高性能、可扩展的在线服务体系以承载海量请求。

下文将从工程视角，系统介绍模型训练与推理的关键技术能力。



3.4.1. 模型训练

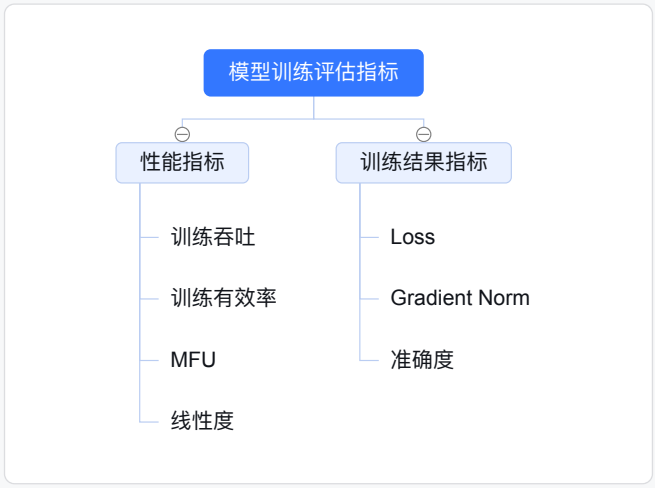
训练全景图

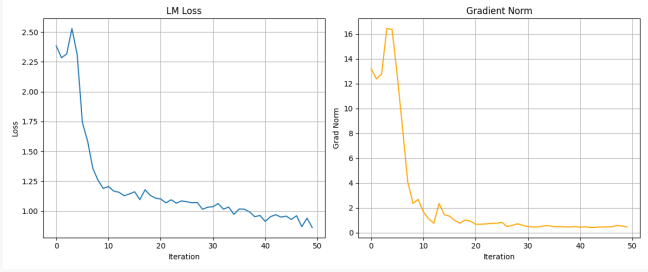
训练分类

大模型的训练可以分为两个阶段：**预训练**阶段和**后训练**阶段。

- 预训练：该阶段以**自监督学习为主，使用的数据量最大，是最消耗算力的环节**。主要包含环节：收集训练数据、数据清洗和配比、自监督训练和退火训练等。
- 后训练：该阶段**主要提升优化效果，提升生成质量和指令遵循能力和在细分领域的表现**。主要包括环节：监督微调、拒绝采样微调、基于RLHF/DPO进行对齐训练和特定能力增强等。

模型训练评估指标





	Llama 3-V 8B	Llama 3-V 70B	Llama 3-V 405B	GPT-4V	GPT-4o	Gemini 1.5 Pro	Claude 3.5
MMMU (val, CoT)	49.6	60.6	64.5	56.4	**69.1**	62.2	68.3
VQAv2 (test-dev)	78.0	79.1	**80.2**	77.2	–	**80.2**	–
AI2 Diagram (test)	84.4	93.0	94.1	78.2	94.2	94.4	**94.7**
ChartQA (test, CoT)	78.7	83.2	85.8	78.4	85.7	87.2	**90.8**
TextVQA (val)	78.2	83.4	**84.8**	78.0	–	78.7	–
DocVQA (test)	84.4	92.2	92.6	88.4	92.8	93.1△	**95.2**

性能指标：

- **训练吞吐**：**模型在单位时间内能够处理的数据量**。常见的度量方式包括second per step以及tokens per second。
- **训练有效率** = 1 − 无效训练时长占比（ $\frac{\text{无效训练时长}}{\text{总训练时长}}$ ）。无效训练时长可理解为由于故障引发的训练暂停、重启与重复计算所消耗的时间。
- **MFU**（Most Frequently Used）：$MFU = \frac{\text{实际吞吐量}}{\text{理论最大吞吐量}} = \frac{\text{模型理论算力使用}}{\text{理论算力峰值}}$ ，主要描述**模型算力利用率**，基于访问频率的资源 / 缓存管理策略，通过统计**对象 / 资源的被访问次数**作为核心权重，权重越高（访问越频繁），缓存和资源调度优先级越高。一般而言，中等规模集群，Dense模型追求50~80%的MFU，而MOE模型一般为30~50%。
- **线性度** = $\frac{\text{多机多卡总吞吐率}}{\text{单机总吞吐率} * \text{集群机器数量}}$ ，描述**训练规模在扩展时资源利用率情况**，线性度的取值范围为0~1，数值越接近于1，其性能指标越好。

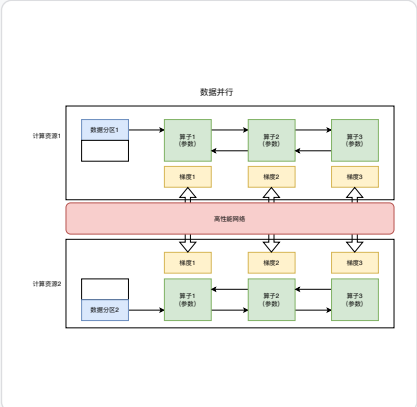
训练结果指标（如左图所示）

- **Loss**：反映模型对当前数据集的**拟合程度**，期望loss在训练过程中稳定下降，尽量不出现突刺与震荡。
- **Gradient Norm**：对**梯度大小的衡量**。该指标反映优化过程是否稳定，可以监控梯度爆炸和消失等现象。
- **准确度**：模型**训练结果**核心指标。一般而言准确度越高，模型越聪明。

随着模型参数越来越多，训练数据集越来越大，单个机器上有限的资源已经无法满足训练需要，必须引入分布式训练来系统地解决海量的资源和计算资源问题，更多详情请参考 【第五课】LLM 训练初探

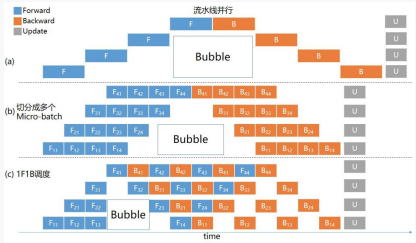
分布式训练并行策略

数据并行DP



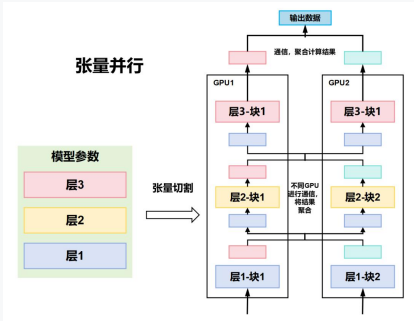
数据并行：数据切分，提升了整体训练吞吐量，与单个计算设备相比区别，反向计算中梯度需要在所有计算设备中进行同步。

流水线并行PP



流水线并行：按层切分，通过**将模型的网络层按顺序切分到多个计算设备上**，让每个设备只需持有模型的一部分参数，**多个设备同时处理同一批训练数据的不同“微批次”**，行程计算重叠，从而解决了大模型参数量远超单设备显存容量的问题。

张量并行TP



张量并行：模型切分，前提是保证模型各种参数切分后的结果一致性（数学证明）。一般来说每次切分计算完后，结果都会进行一次同步。

- 通信模式：Allreduce同步梯度
- 通信量与模型规模正相关，单卡可达10GB+
- 一个step一次通信

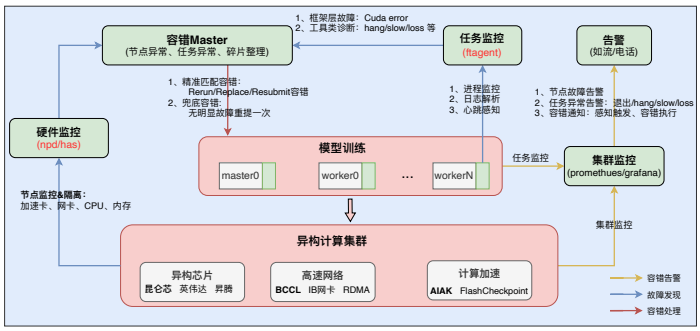
- 通信模式：点到点，正向传激活，反向传梯度
- 通信量与层间交互相关，一般在MB级别
- 一个step几十次通信

- 通信模式：Allreduce同步矩阵乘结果
- 通信量与batchsize有关，矩阵可达GB级别
- 一个step几十次通信

在大规模训练中，训练容错主要用来实现“**跑的稳**”，其核心目标：**保证大规模分布式训练不中断、不丢进度**

训练容错

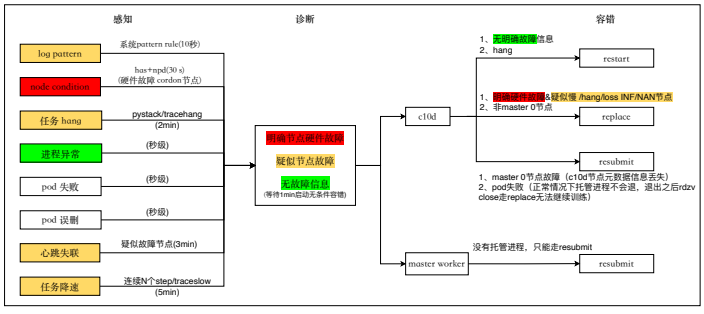
容错架构图



概括总结

1. 训练容错分为对集群**节点**和训练**任务**的监控和异常检测
2. 训练容错整体处理分三种：
   a. **容错监控**：通过集群和训练任务监控，将异常情况通知给客户，例如节点发生故障、训练任务异常退出/hang/slow/loss跑飞的情况，以及容错流程的信息会第一时间发送到群聊，必要时打电话通知用户。
   b. **异常检测**：通过NPD/HAS检测计算集群节点的加速卡、网卡、CPU、内存等硬件故障，当发现节点存在故障之后及时屏蔽维修，并将具体的故障原因暴露给容错Master。在任务侧通过ftagent组件监测训练进程，流式分析用户训练日志，以及心跳感知来检测任务是否异常。
   c. **容错处理**：容错Master根据收集到的节点问题和任务异常信息，通过综合分析之后采取restart、replace或resubmit来恢复训练。
3. 容错处理过程主要是 `感知` => `诊断` => `容错`

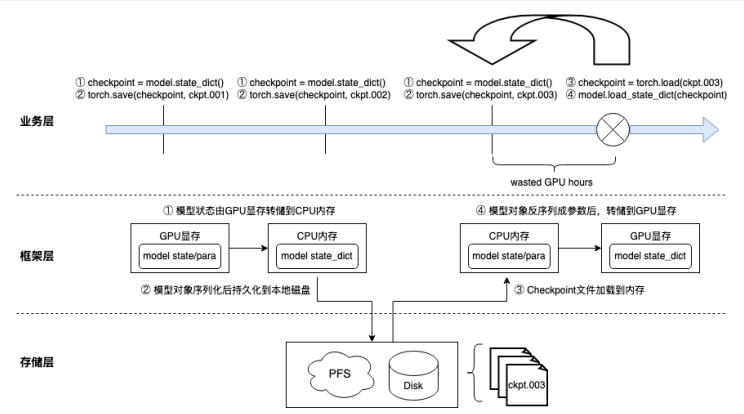
容错处理过程

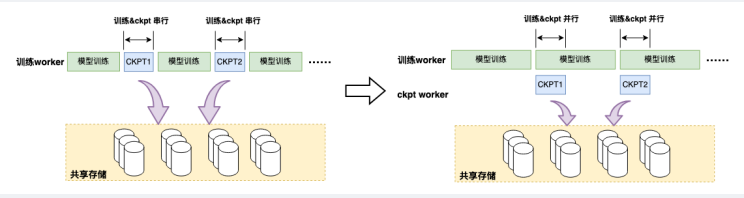


概括总结

1. 八种故障感知方式，四种诊断定位方法以及三种容错手段
2. 故障感知主要包括：任务输出的日志错误(system pattern)、节点上报的故障(node pattern)、任务hang和降速、pod失败退出或者误删、以及训练心跳失联或者进程异常等。
3. **四种诊断方法**：
   - HAS（硬件级别）
   - NPD（硬件级别）
   - Tracehang（任务级别）
   - Traceslow（任务级别）
4. **三种容错手段**：
   - resubmit：删除所有的pod对任务进行重新创建，对全部worker进行一次调度；
   - replace：替换有问题的pod，对部分worker进行一次调度；
   - restart：不替换pod，对任务进程进行重启，不需要调度；

checkpoint过程





概括总结

1. **Checkpoint保存流程：**
   a. **模型信息转储**（秒级，受限于PCIE带宽）：通过model.state_dict()将要保存的对象（如模型的state_dict、优化器的state_dict，训练轮次，超参数等）转储到CPU内存的字典中，此时并不会修改原始模型的位置。
   b. **模型对象序列化&数据持久化**（分钟级，受限于存储介质写带宽）：将检查点对象字典（已经在CPU内存中）传递给torch.save()，将其字典序列化为二进制流并持久化到本地磁盘文件或远程存储。
2. **Checkpoint恢复流程：**
   a. **数据加载**：通过torch.load()函数从文件中读取保存的Checkpoint二进制数据。
   b. **模型对象反序列化**：将二进制数据反序列后转为模型对象字典。
   c. **模型信息恢复**：通过model.load_state_dict()加载模型对象字典，并将模型状态恢复至Checkpoint的位置。
3. **异步Checkpoint流程**：
   a. **Checkpoint保存与模型训练重叠**，降低Checkpoint占用GPU训练的有效时间。
   b. 用户进程调用save_checkpoint保存Checkpoint，模型状态由GPU显存转储到CPU内存
   c. 模型训练与Checkpoint保存并行进行
      i. 用户进程继续开始后续迭代训练
      ii. 同步将内存中的模型状态转储到共享存储
   d. 模型训练结束之前，通知后台进程训练结束并等待最后一次Checkpoint持久化完成

3.4.2. 千卡训练

实验使用宁夏P800集群，k8s调度，AIAK-Training-LLM + AIAK-Megatron训练框架，进行Qwen2.5-72B模型pretrain。

- 主要进行训练性能调优、MFU线性扩展度验证以及自适应框架准确度摸底。其中，调优过程主要在32卡下进行。之后扩展dp至64卡、128卡、256卡、512卡、1024卡及2048卡，过程中测量性能并计算MFU线性度。最后，结合自适应框架能力，确认自适应框架准确度。

集群环境准备

k8s组件依赖

- aibox（training-operator）
- npu-mananger
- rdma-manager
- volcano

故障容错

- node-problem-detector：故障检测
- node-remedier：故障自愈软件
- Prometheus：监控指标采集
- BLS：日志收集

配置调整

- volcano开启rdma tor拓扑感知
- runtime取消memlock限制
- 内核参数调整，例如：ip_local_port_range范围调整支持大规模集群训练。

资源管理

- 使用CFS/PFS做共享存储，实现模型权重等资源共享
- 队列管理

训练性能调优

首先进行理论分析，然后在32卡场景下，结合profile数据分析调优，以下是实际调优内容：

1. **offload调优+重计算调优**：关闭offload和精细调整重计算策略使训练性能提升了约 42%。
2. **global-batch-size增大**：训练性能随着 global-batch-size 增大而增大。实测表明，当 gbs 从较小值增至较大值时（gbs <= 128），性能获得显著提升；但继续增大至更大规模，性能的增幅则趋于平缓。
3. **并行策略调优+重计算重调**：通过优化并行策略并调整重计算配置，训练性能进一步提升了约 5%。
4. **多流调优**：xpu 下默认为单流，在开启多流时，会提升部分通信的overlap，实测性能进一步提升了约 7%。

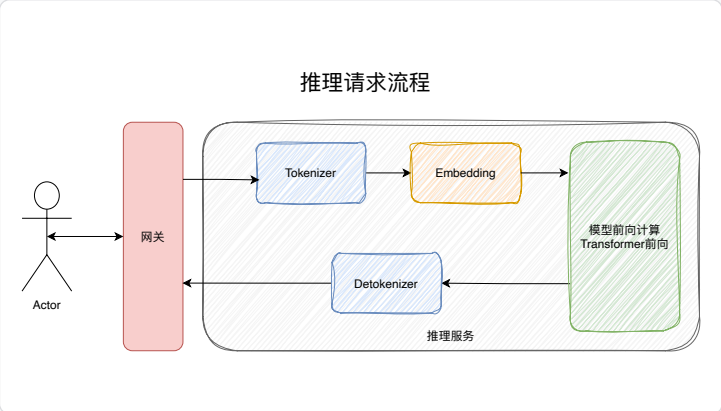
训练性能调优结果

初步实验数据：2k卡 MFU 约为 58%，从32卡到1024卡，MFU 线性度约为 96%，从32卡到2048卡，MFU 线性度约为 91%。

3.4.3. 推理服务

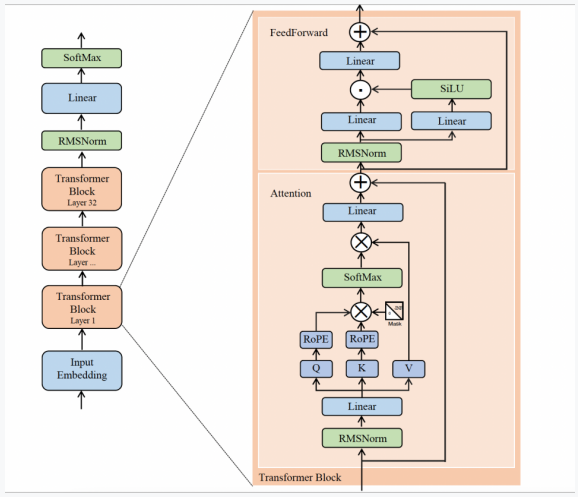
我们已在第四课中围绕推理服务搭建了完整的基本框架认知，详细可查阅 【第四课】LLM 推理初探，本节将从系统视角对推理服务展开进一步介绍。

推理服务



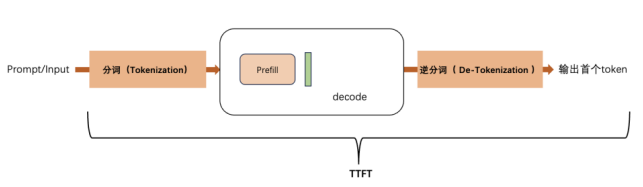
说明：

- 推理过程中只有前向计算，没有反向计算（因为推理是使用模型参数，不会更新模型参数）
- 通过API网关实现推理请求路由和负载均衡等能力。
- 模型前向计算中加速手段与训练中类似，都有TP、DP、PP等方式，但具体实现效果有差异。



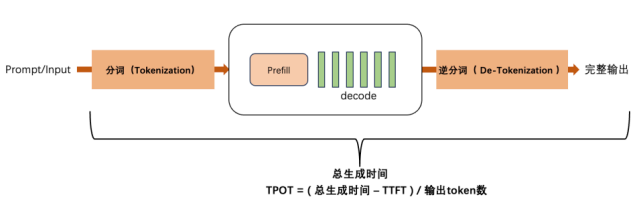
推理详细过程

推理评估指标



TTFT示意图

TTFT：（Time to first token，**首 Token 延迟**），从发送请求到系统生成第一个输出 token 的时间。衡量系统对单个请求的响应速度，TTFT 越低，用户体验越好。



TPOT示意图

TPOT（time per output token，**单 Token 生成时间**）：系统生成每个输出 token 所需的时间，不含首个token。TPOT 越低，模型生成文本的速度越快。

```
Maximum request concurrency: 2000
100%|          | 10000/10000 [10:13<00:00, 16.29it/s]
============ Serving Benchmark Result ============
Successful requests:                     10000
Maximum request concurrency:             2000
Request rate configured (RPS):           25.00
Benchmark duration (s):                  613.99
Total input tokens:                      20490000
Total generated tokens:                  20467954
Request throughput (req/s):              16.29
Output token throughput (tok/s):         33335.71
Total Token throughput (tok/s):          66707.33
---------------Time to First Token----------------
Mean TTFT (ms):                          3346.01
Median TTFT (ms):                        1907.92
P99 TTFT (ms):                           13040.85
-----Time per Output Token (excl. 1st token)------
Mean TPOT (ms):                          49.85
Median TPOT (ms):                        50.61
P99 TPOT (ms):                           55.32
---------------Inter-token Latency----------------
Mean ITL (ms):                           95.23
Median ITL (ms):                         96.21
P99 ITL (ms):                            146.84
==================================================
```

vllmbenchmark测试结果

输出吞吐：系统每秒能够生成的输出 token 数量。

- 上图中输出吞吐值：33335.71 token/s

推理加速方式

模型 /算法层优化

1. 模型压缩：量化、蒸馏、剪枝
2. 推理算法优化：投机解码、MoE（稀疏）、Sparse Attention、Chunk-prefill等
3. 并行加速：TP、DP、SP等

显存 & KV Cache 优化

1. KV Cache 管理
2. 显存分配优化
3. KV 传输优化

算子 / 内核层优化

1. 算子融合：Fused QKV、MLP fusion
2. Attention优化：FlashAttention / FlashMLA、Paged Attention
3. 编译&Graph优化：CUDA Graph

请求调度 & 并发优化

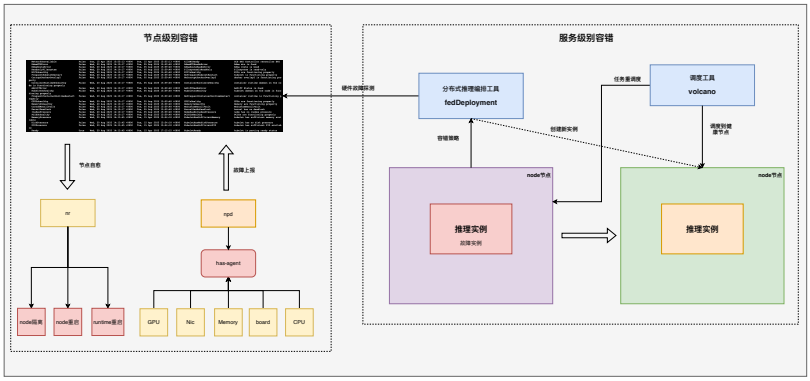
1. Batching 策略：Micro-batching，continue-batching
2. PD分离

系统 & 基础设施层优化

1. 硬件与拓扑：NUMA / PCIe / NVLink 感知、多卡亲和性、拓扑感知调度
2. 通信与网络：NCCL/XCCL 调优、拓扑分层 AllReduce（多实例）
3. 运行时 & 容器：container-runtime、HugePage / shm
4. 可观测性：TTFT / TPOT / tokens/s、GPU 利用率、队列延迟

推理容错

推理容错整体架构



概括总结

1. 开启推理容错时，当节点或节点加速芯片故障时封锁此节点
2. 自动驱逐故障节点上推理服务到健康节点上保证服务高可用
3. 节点同错需要has-agent（故障感知），Node-Problem-Detector（检测能力扩展），CCE-Node-Remedier（节点屏蔽修复）组件支持

故障场景

场景一：推理服务进程中断

1. fedDeployment能感知到pod状态异常，并清理异常实例组，重建新的实例组

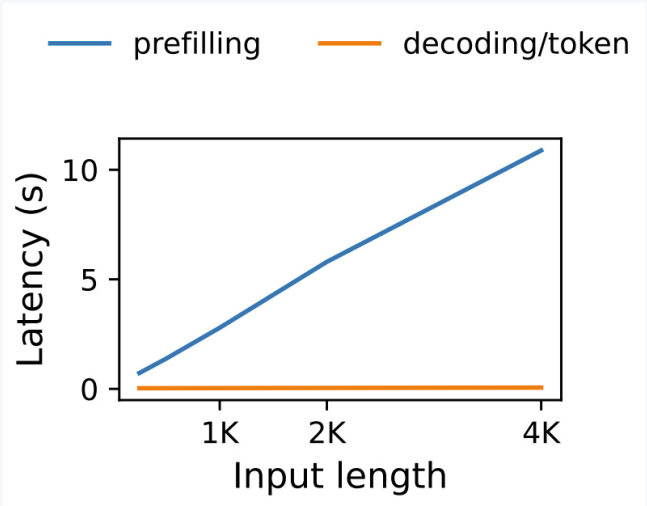
场景二：节点故障

1. NPD周期性调用has-agent检测脚本，检测到节点故障后，更新node condition
2. NR根据配置策略，自动封锁故障节点，并驱逐节点pod
3. pod驱逐后触发 fed 协调逻辑，在健康的节点上重新部署

场景三：GPU卡故障

1. 自动封锁坏卡所属节点，新pod无法再调度到所属节点
2. 驱逐坏卡上的Pod，在健康节点上重新部署
3. 所属节点其他健康的GPU卡上的Pod，提醒所属节点有坏卡，不驱逐好卡上的pod，由用户手动驱逐并重建

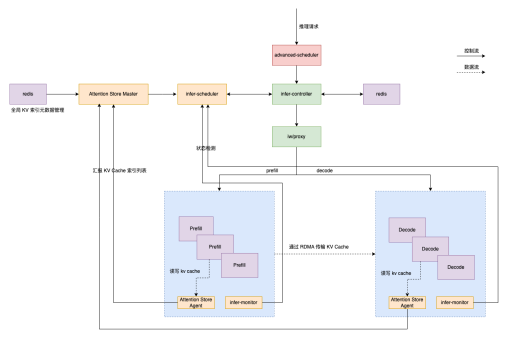
3.4.4. PD分离



推理过程划分为**Prefill阶段**（处理完整prompt输入）和**Decode阶段**（自回归token生成）。

- Prefill 阶段：**计算密集型**（compute-bound），Prefill 的计算压力更大。
- Decode 阶段：**访存密集型**（memory-bound），由于逐 token 生成的特性，Decode 阶段需频繁访问 KV Cache，因此需要尽可能多地通信资源以保障推理效率。

厂内PD分离架构

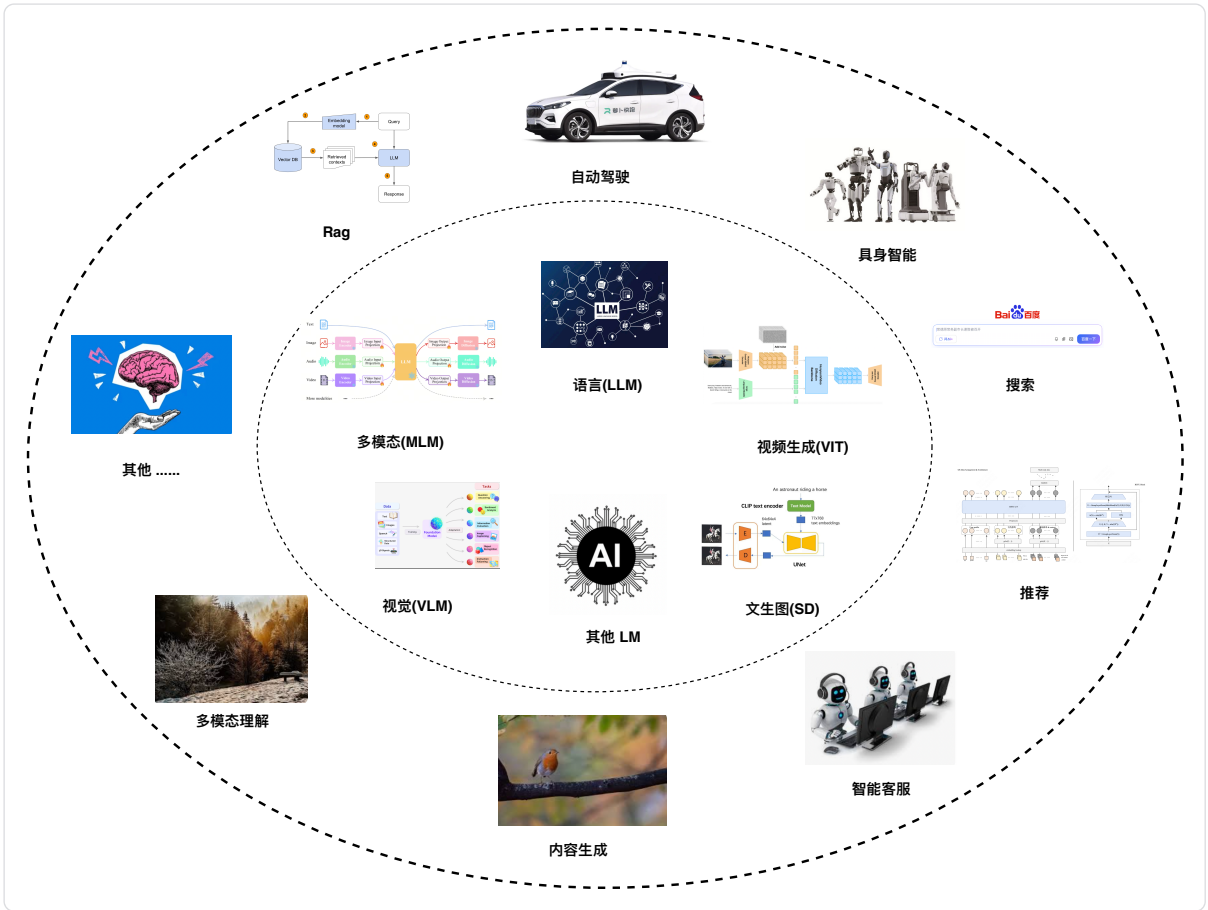


厂内PD分离整体架构

组件	作用
advanced-schedular	作为用户请求的入口，负责接收并初步处理请求，将请求转发给 ir
inference-controller	作为推理服务的调度与管控中心，负责接收和管理用户推理请求，流程的控制
infer-worker	主要负责从推理控制器(inference-controller)拉取推理请求，调用模
infer-proxy	推理工作节点，直接与PD实例交互
infer-monitor	秒级观测实例状态，实时感知引擎状态，实现状态上报，同时防抖
infer-scheduler	全局Cache-Aware负载均衡调度，负责为推理请求分配最佳P实例
attentionstore-master	kv cache存储服务，为IS提供请求前缀命中信息；Master 收集 Ag 化索引数据，以实现多副本高可用
attentionstore-agent	独立于用户推理实例部署，响应一个节点上所有实例的 KV 缓存读 Master
aiak-sglang	开源的推理引擎，用于组建推理服务集群，部署 PD 引擎

3.5. 平台与应用层：AI 能力真正被使用

前面整体介绍内容中更多解决的是能不能跑、跑得快不快的问题，**而真正决定 AI-Infra 是否具备长期生命力的，往往是最上层的**平台与应用层。**平台层是“生产力的工具箱”**，解决如何高效、规范地生产AI模型的问题；**应用层是“价值的交付物”**，解决如何将AI模型变成可消费、可解决实际问题的产品与服务。



平台层直面用户，通常需要提供的能力如下：

- 统一的训练与推理入口
- 标准化的使用方式和接口
- 全链路可观测与运维能力
- 与上层业务系统的无缝对接

通过平台化封装，复杂的基础设施能力才能被持续复用。通常平台层通过AI平台和AI应用与解决方案两种形式提供AI能力供用户使用。

AI平台	AI应用与解决方案
• 提供一套完整的工具集，以平台形式对外提供服务 • 核心产出物/功能： ◦ 可视化建模工具：可视化图形界面，降低模型开发和操作门槛。 ◦ 自动化机器学习（AutoML）：自动进行特征工程、模型选择和超参调优。 ◦ 模型训练与分布式调度：高效管理底层算力，执行训练任务。 ◦ 模型管理与仓库：对模型版本、性能、镜像进行统一管理。 ◦ 一键部署与服务化：将模型封装为 RESTful API 等标准化服务接口。 ◦ 运维监控中心：监控服务性能、流量、资源消耗，并支持弹性伸缩和灰度发布。	• 主要面向最终用户或业务部门，提供开箱即用的 AI 功能 • 产物形式： ◦ API/SDK：将AI能力（如图像识别、语音合成、自然语言处理）封装为API，供开发者简单集成。例如百度AI开放平台的各类API。 ◦ 行业解决方案：针对金融、医疗、制造、零售等特定行业的打包方案，如智能客服系统、工业质检平台、智慧医疗辅助诊断系统。 ◦ 终端用户应用：直接可用的软件或硬件产品，如智能翻译机、AI修图App、企业级知识管理机器人（如Kimi、文心一言等对话应用）。 ◦ 低代码/无代码AI工具：让业务人员通过配置即可创建简单的AI应用，如智能表单识别、自动报告生成工具。

本章小结

本章系统阐述了现代AI基础设施（AI-Infra）作为一个**分层解耦、协同演进的完整技术体系**。整个体系如同一个精密的“金字塔”体系，自下而上包含五个关键层次：

1. **算力与硬件层（地基）**：以GPU/NPU、高速网络（NVLink/RDMA）和高性能存储（NVMe SSD/并行文件系统）构成物理基石，决定了模型规模的**理论天花板**
2. **资源管理与调度层（调度中心）**：通过容器化、Kubernetes及Volcano等调度器，将离散的硬件资源抽象为**弹性、稳定、高可用的全局算力池**，解决“资源可用”问题
3. **算子与编译层（转换器）**：承担承上启下的关键角色，通过算子优化与编译技术，将抽象的模型计算高效“翻译”成硬件指令，是**释放硬件潜力的核心**
4. **训练与推理系统层（执行引擎）**：承载模型任务，攻克大规模分布式训练中的并行、通信、容错等挑战，以及高并发推理中的调度、内存与延迟优化，确保模型能**“跑得稳、跑得快、跑得久”**
5. **平台与应用层（价值界面）**：作为与业务交汇的最终界面，通过AI平台提供工程化工具链，降低开发门槛；并通过API、行业解决方案等形态将技术能力**“产品化”**，直接驱动业务增长体系中下层为上层提供可靠的能力支撑，上层则不断对下层提出新的需求与优化方向。

4. 平台实践案例：AI-Infra 如何真正跑起来

为了将 AI-Infra 落地的更好，诸如 **AIHC 百舸（百度）**、**PAI（阿里云）**、**TI-ONE（腾讯云）**等 **AI-Infra** 基础设施异构平台应运而生。它并非简单地交付一套集群或工具链，而是围绕企业级大模型训练与推理场景，提供从**资源管理、运行环境、训练推理系统到运维与交付的一体化能力**，使 AI Infra 真正“跑起来、跑得久、跑得稳”。

接下来，我们将以百舸私有化平台（**AIHC-Private**）为例，重点介绍平台化 **AI-Infra 的核心能力设计与实践路径**，以及这些能力如何在真实业务中落地并持续演进。

概括总结：

- **AIHC-Private是**混合云团队推出的 **面向 ToB 场景的异构算力解决方案**
- **功能体系完整**，覆盖资源与用户管理、模型训练与推理、调度编排、数据中心与运维管理等核心能力，能够支撑从算力管理到模型服务的全流程需求。
- 在能力与场景上，适配 **千卡/万卡级大规模集群**，支持训推加速、多芯统管与任务快速编排，兼顾轻量交付与资源利用率提升。
- 支持 **容器与虚拟机** 两种资源形态，具备高性能计算、存储与网络能力，以及专有调度和训推容错机制，并通过 **管理端全局视角 + 用户端多租户视角**，配合完善的权限与 License 管理，实现稳定、可控的规模化落地。

5. AI-Infra 发展趋势和总结

5.1. AI-Infra 的未来演进方向

百度世界2025大会百度云AI云

AI Infra 技术趋势

1. **AI 原生基础设施架构升级**
   - **算力、数据、模型的协同化程度持续加深**。这不仅是软硬件的简单堆砌，更是一次系统级的重构。《AI原生基础设施建设指南（2026）》中提到“AI原生基础设施并非现有IT设施的简单升级，而是从设计之初就以规模化支撑AI原生应用为核心，通过软硬件、网络、数据、算法的深度协同，为AI应用提供全生命周期支持的一体化体系。”未来，基础设施将具备统一调度计算、数据、模型的能力，让开发者像使用一台“超级计算机”一样简单地调用AI能力。
   - **智能资源调度与弹性扩缩容机制**。主要体现在推理系统，根据实时流量动态调整实例数量，避免资源闲置。
   - **容错与自动恢复机制**。支持秒级 Checkpoint 和断点续训，避免因节点故障导致任务归零。
2. **超节点**
   - 随着算力需求的不断增加，scale-out方式构建的集群满足不了性能需求（就算使用RoCEv2或者IB，网络带宽达到Tbps级别，延时还是会在ms区间），所以我们需要通过采用sacle-up的方法，把单台服务器的性能拉高，这就是超节点技术。
   - **“超节点”是应对大模型万亿参数的关键技术**，它将数千甚至上万张AI芯片紧密耦合，像一个巨型单机一样工作，实现了算力资源的调度与利用效率的双重提升，极大提升了计算效率和规模，是解决更大算力需求的利器。
   - **超节点已成为AI时代下企业进行AI大模型训练的主流解决方案**，他的Scale-up+Scale-out的架构是当下AI基础设施的建设新范式。
   - 目前，英伟达通过与OpenAI等巨头深度“资本+生态”绑定，强化其主导地位；而国内则通过开源策略（如开放顶尖模型）和自主研发（如百度天池超节点，华为昇腾超节点），**加速构建自主生态，未来发展前景较大**。
3. **多模态融合**
   - **打破数据类型壁垒，实现跨模态数据流统一调度**，例如：百度文心一言4.5：统一处理文本、图像、视频输入，推理延迟降低40%。火山引擎将文本、图像、音视频等非结构化数据统一存储在“数据湖”中，并利用计算引擎进行高效调度，直接服务于AI训练和推理，针对性解决了非结构化数据处理的痛点。
   - **提供国产芯片兼容层与AI编译优化**，实现 PyTorch/TensorFlow 模型向昇腾、寒武纪、昆仑芯等平台的无缝迁移。
4. **智能体 Agent Infra**
   - **智能体成为AI应用主流形态**，推动多智能体协同解决复杂场景问题。赛迪顾问《2025中国AI Infra平台市场研究报告》中有提到“AI Infra的‘两年一核心范式革新’规律：2022年对话式训练，2024年动态算力分配，**2026年将围绕多任务Agent与强化学习展开**。”

5.2. 总结和思考

- **AI-Infra 绝非单一硬件的简单堆砌，而是支撑 AI 训练与推理全流程的核心算力底座**，是大模型时代的 **“核心竞争力”**，缺乏高效的 AI Infra，难以平衡大模型的成本与效果，更无法支撑超大规模任务落地。
- **AI-Infra的核心价值在于垂直整合**，作为硬件与模型的中间衔接层，若仅做孤立的中间层优化易陷入内卷，唯有向硬件端协同适配、向模型端参与设计，才能建立壁垒。

- 未来，AI-Infra 将会迎来**新一轮的AI范式革新**，通过软件最大化硬件潜力，为大模型的低成本、高效率落地提供核心支撑。