基于算网状态感知的多集群GPU算力资源调度平台设计与实现

胡亚辉[1]，张宸康[1]，王越嶙[1]，洪雨琛[1]，范鹏飞[2]，宋俊平[2]，周旭[2]
（1.中国矿业大学（北京）人工智能学院，北京 100086；2.中国科学院计算机网络信息中心，北京 100083）

摘　要：针对大规模深度学习任务的多集群GPU调度中资源粒度粗放、缺乏统一vGPU视图及跨集群网络感知不足等问题，设计算网状态感知的多集群GPU算力调度平台。平台采用集中式架构，通过实时感知跨集群算力资源与网络状态并协同调度，实现细粒度全局资源编排调度。平台先构建设备、集群、vGPU及网络层多维度指标体系，实时采集核心利用率、显存、带宽等关键数据；设计节点级vGPU编排部署模块，突破“作业到集群”局限，达成“作业到节点”精准调度，提升GPU共享效率与资源利用率。实验表明，平台可实现多集群vGPU与网络信息的实时采集可视化，经DDPG强化学习及BestFit算法验证，具备高效资源管理能力。

关键词：多集群；图形处理器；算力资源；算网状态感知；编排调度
中图分类号：TN393
文献标志码：A
DOI：10.11959/j.issn.1000–436x.2025189

Design and implementation of a multi-cluster GPU computing resource scheduling platform based on network and computing state awareness

HU Yahui[1], ZHANG Chenkang[1], WANG Yuelin[1], HONG Yuchen[1], FAN Pengfei[2], SONG Junping[2], ZHOU Xu[2]
1. School of Artificial Intelligence, China University of Mining and Technology-Beijing, Beijing 100086, China
2. Computer Network Information Center, Chinese Academy of Sciences, Beijing 100083, China

Abstract: To address issues like coarse-grained resource allocation, lack of a unified vGPU view, and insufficient cross-cluster network awareness in multi-cluster GPU scheduling for large-scale deep learning tasks, a computing-network state-aware multi-cluster GPU scheduling platform was designed. Adopting a centralized architecture, fine-grained global resource orchestration was achieved through real-time perception and collaborative scheduling of cross-cluster computing resources and network states. Firstly, a multi-dimensional metric system covering device, cluster, vGPU, and network layers was built to collect real-time key data such as core utilization, memory, and bandwidth. Then a node-level vGPU orchestration module was designed, breaking the "job-to-cluster" limitation to realize precise "job-to-node" scheduling and improve GPU sharing efficiency. Experiments show that the platform enables real-time collection and visualization of multi-cluster vGPU and network information, and exhibits efficient resource management capabilities verified by DDPG reinforcement learning and the BestFit algorithm.

Keywords: multi-cluster, graphics processing unit, computing resource, network and computing state awareness, orchestration and deployment

收稿日期：2025–07–01；修回日期：2025–10–18
通信作者：胡亚辉，huyahui@cumtb.edu.cn
基金项目：国家重点研发计划基金资助项目（No.2024YFB2908700）；中央高校基本科研业务费专项资金资助项目（No.2025ZKPYZN02）；国家能源集团科技环保有限公司开放课题资助项目（No.YZ-2025-101）
Foundation Items: The National Key Research and Development Program of China (No.2024YFB2908700), The Fundamental Research Funds for the Central Universities (No.2025ZKPYZN02), Open Research Topics of CHN Energy Technology & Environment Limited (No.YZ-2025-101)

0 引言

近年来，深度学习技术取得了突飞猛进的发展，并在众多领域得到了广泛应用，如图像识别、自然语言处理、语音识别和智能决策等[1-2]。深度学习模型的规模和复杂性不断增大，尤其是大语言模型的出现和广泛应用，对计算资源的需求也呈指数级增长[3]。首先，不同深度学习框架和任务对图形处理器（GPU, graphics processing unit）资源的需求各异，导致资源分配效率低下和资源碎片化问题[4]。其次，深度学习任务的动态性和不确定性使管理和调度GPU资源变得极为困难。最后，单一集群的GPU资源无法满足大规模AI模型训练和推理业务，必须实现多集群GPU算力资源调度管理和分配，以高效利用大规模分布式计算资源。

针对日益增长的多集群GPU算力统一管理与调度需求，相关调度平台应运而生。目前，业界主流的方案包括Kubernetes[5]、Slurm[5]、Mesos[6-7]、Clusternet[8]和Volcano[9]等。这些平台在架构设计上主要分为3种类型[10]：混合架构[11-15]、分布式架构[16-19]、集中式架构[20-23]。混合架构通过对接不同的调度系统来统一管理多集群GPU资源，优点是兼容性强、灵活性高，能充分复用已有生态，降低演进成本；但架构复杂、调度开销大，难以实现全局最优。分布式架构由多个调度器协同工作，具备良好的扩展性和容错性，能够在本地快速做出资源决策，适合大规模多地域环境；但全局一致性较弱，调度器间的协调和同步成本较高。集中式架构存在单点瓶颈，扩展性和容错性不足；但依靠中央调度器统一管理，能实现全局最优和策略统一，资源利用率较高，适合规模较小或对一致性要求高的场景。因此，本文采用集中式架构构建多集群GPU调度平台。

上述平台中，Slurm和Volcano是2种常用的集中式多集群GPU调度平台。Slurm是一款为大规模高性能计算设计的分布式工作负载管理器，能够高效处理数十万级别的作业队列。然而，在多集群GPU调度方面其存在明显短板：首先，缺乏统一的跨集群资源视图；其次，调度粒度仅限于物理GPU卡级别，不支持虚拟图形处理器（vGPU, virtual graphics processing unit）等更精细化的资源划分与调度。

Volcano Global v1.11（华为于2025年4月发布）在Karmada基础上，旨在满足多集群AI批处理作业的需求。它采用2级调度架构：控制器负责处理作业与队列的优先级；Karmada调度器负责将任务调度到集群的具体执行。该架构在一定程度上提升了资源利用率，但其固有模式也带来了局限性。首先，其跨集群GPU调度应用场景和范围有限，只支持Volcano Job的跨集群调度；其次，由于Karmada调度器只能进行“作业到集群”的粗粒度决策，无法感知各子集群内部节点的实时负载，因而不能实现“作业到节点”的精细化调度，这限制了其对全局GPU资源利用率的进一步提升潜力；最后，上述GPU算力资源调度方案尚未考虑网络资源的状态，无法做到算网资源的协同调度，影响最终业务的服务质量。在此背景下，算网协同逐渐成为研究的热点，旨在优化计算资源与网络资源的调度与管理，从而提升整体系统的性能和资源利用率[24-26]。然而，现有的算网协同调度研究主要集中于算法层面，调度平台方面的工作较少，且尚未实现多集群节点级vGPU调度。

鉴于此，本文面向算网资源需求异构的各类深度学习框架和任务，设计基于算网状态感知的多集群GPU算力资源调度平台，开发GPU算力和网络资源采集功能，真正实现vGPU级别的细粒度算力资源及多集群网络资源的统一视图，支持多种调度算法，满足不同场景下的资源管理需求，最终提升算力资源利用率。本文的主要研究工作如下。

1) 基于算网状态感知的多集群统一纳管通用GPU算力调度平台设计与实现。采用集中式架构，对多集群vGPU资源进行统一纳管，实现了计算资源和网络资源的全局视图和集中调度，避免了计算资源碎片化，显著提升了计算资源利用率，同时支持多种调度策略，满足不同应用的需求。

2) 节点级vGPU感知模块的设计与实现。建立了从集群层、设备层延伸至关键的vGPU层的多维度GPU算力指标体系，涵盖10项算力资源状态指标。基于该指标体系，设计并实现了集群信息采集模块，能够实时、精准地获取vGPU核心利用率和显存分配等细粒度运行状态数据，为高性能调度决策提供了坚实的数据基础。

3) 设计并实现了基于算网状态感知的多集群细粒度GPU算力资源编排部署模块。该模块能够

直接利用采集到的节点级vGPU信息、网络状态信息以及用户需求，进行跨集群细粒度资源调度，实现了从传统的“作业到集群”到“作业到节点”的根本性转变，显著增强了多集群物理GPU在多个容器和作业间的有效共享能力。

4) 多种调度算法的集成与验证。平台提供了开放、可插拔的调度算法接口，集成了深度强化学习算法（如深度确定性策略梯度（DDPG, deep deterministic policy gradient））以及多种经典启发式算法（如BestFit和Utilization）。实验结果表明，本文平台能够实现多集群节点级vGPU算力信息和网络信息的采集与展示，方便用户监控和管理GPU资源，支持多种调度算法对算网状态感知的多集群GPU算力资源进行统一编排调度。

1　基于算网状态感知的多集群GPU算力资源调度平台设计

基于算网状态感知的多集群统一纳管通用GPU算力资源调度平台结合了vGPU资源采集模块，实现了对底层各个集群节点级vGPU资源的实时采集，将分散在多个物理位置或不同业务单元下的GPU资源池，通过一个中心化的控制平面进行统一管理。这种模式将原本割裂的资源汇聚成一个逻辑上的“全局资源池”，使调度决策可以实现全局最优而非局部最优。Karmada在此架构中扮演了至关重要的角色，它不仅负责将底层集群的各类算力资源信息采集并上报至上层调度平台，同时也是执行调度策略的引擎，将最终的任务部署至指定的集群和节点。同时每个集群连接一个基于IPv6的分段路由（SRv6,segment routing over IPv6）网关，可以监测集群之间的链路质量。该集中式模式的优势在于其能够提供一个单一的、真实的资源状态视图，极大地简化了调度问题的复杂性。

多集群GPU算力资源调度平台采用了Kubernetes、Karmada等现有调度管理软件，并开发了GPU采集模块、网络状态采集模块和MySQL算力资源信息数据库，实现了集群算网信息的采集整合，能够对用户算力资源需求进行合理编排调度。基于算网状态感知的多集群GPU算力调度平台框架如图1所示，主要由以下几个关键部分组成。

1) 集群。多个Kubernetes集群，组成了多集群调度平台的最底层。这些集群之间相互隔离，确保了各业务之间的独立性和安全性。

2) Karmada。Karmada是连接多集群调度平台与底层Kubernetes集群的组件，负责采集底层集群中各种算力资源信息，并上报给上层的多集群

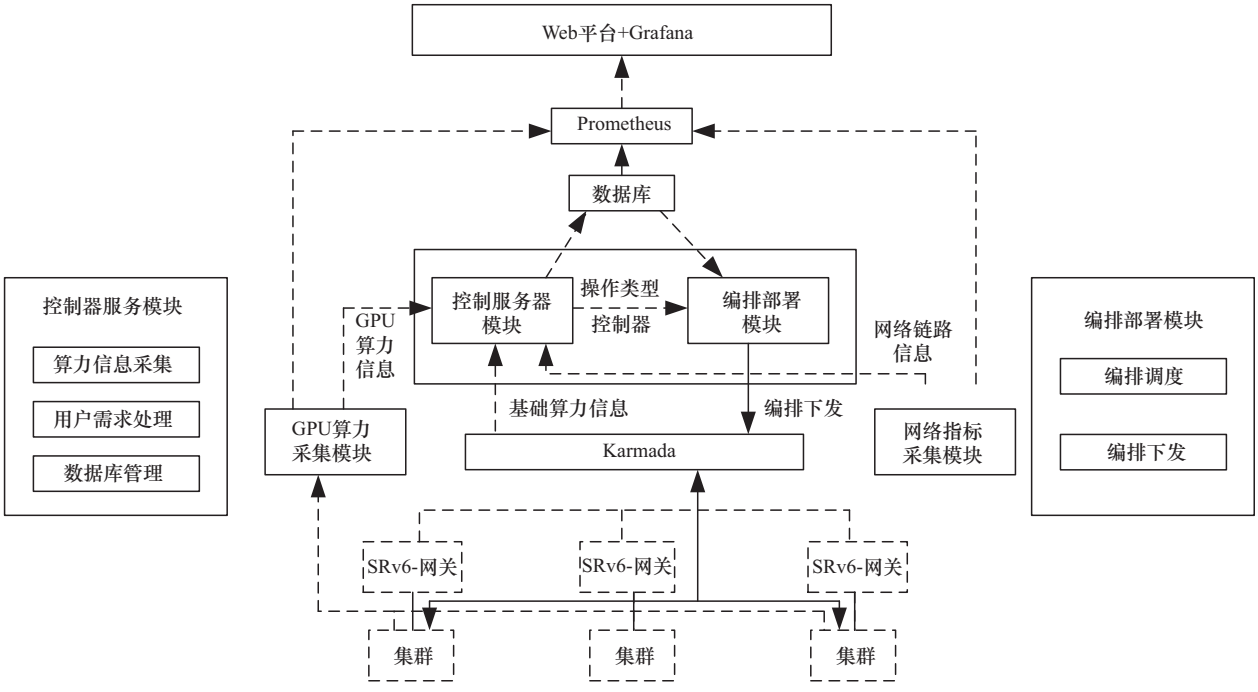


图1　基于算网状态感知的多集群GPU算力调度平台框架

调度平台。除此之外，Karmada还负责执行调度平台生成的调度策略，将任务部署至对应的集群中。

3) GPU算力采集模块。为了解决Kubernetes以及Karmada在GPU资源采集方面的不足，设计了GPU算力采集模块。该模块可收集各集群vGPU资源的使用情况，并通过开放接口供其他模块访问。通过对Karmada接口及GPU算力采集模块接口进行采集，获取GPU核心的利用率、显存的占用情况等关键性能指标。采集到信息后，一方面传输给控制器服务模块，对数据进行清洗、预处理，将其转化为规范、可用的格式，并将其存储至MySQL数据库；另一方面，传输给Prometheus模块，并以JSON格式输出，进行界面可视化。

4) 网络指标采集模块。为了使平台调度任务兼顾算力和网络指标，构建此模块以采集集群网络信息。从每个集群的SRv6网关处，获取集群网关出入口网络质量指标以及集群之间的链路状态信息，上报给控制器模块，用于数据库存储，同时传输给Prometheus用于可视化展示。

5) 控制器服务模块。该模块连接Karmada、Web平台、数据库和编排部署模块的中间件，负责算网信息采集、数据处理、用户需求处理（部署、删除、修改、查询）和数据库管理。当接收到删除请求时，控制器服务模块会向编排部署模块发送删除指令，删除数据库中该任务的信息。当接收到任务修改请求时，控制器服务模块会将修改后的需求传递给编排部署模块，并对数据库中对应的任务信息进行更新。当接收到查询请求时，控制器服务模块会在数据库中查询任务信息，并将查询结果返回给用户。

6) 编排部署模块。在接收到用户需求后，编排部署模块根据集群资源信息与用户需求信息，通过调度编排算法得到部署策略，最终通过Karmada执行生成策略。

7) 数据库。MySQL数据库是整个系统的数据存储核心，用于存储集群资源信息与用户需求信息。在数据库管理方面，控制器服务模块会定期清理过期数据，并定期进行备份，保证数据库的性能和可靠性。

8) Prometheus。Prometheus负责对GPU算力采集模块接口和网络指标采集模块接口进行监控，并存储相关数据。该信息将传给Grafana，以可视化方式展示GPU资源使用情况以及各个集群的网络状态。

9) Web平台+Grafana。该模块是用户提交需求与查看集群算力和网络资源信息的接口，包含任务部署界面和资源查看界面。通过需求提交接口，用户按照Web平台规范正确填写信息后，平台会将任务需求存储至数据库，并通过向后续模块接口发送请求进行任务部署。除此之外，用户还可以通过需求提交模块提交其他请求，包含对任务的删除、修改、查询等操作。通过资源查看界面，用户可以获取到集群基础资源与云原生资源的使用情况，以及实时的集群网络状态信息。

基于上述平台，用户任务调度需求通过Web平台向控制器服务模块发送请求。控制器服务模块在接收到请求后，会对请求进行解析和验证。对于部署请求，控制器服务模块会将任务需求传递给编排部署模块，编排部署模块根据网络状态以及底层集群资源状况进行合理的资源调度和任务编排，最终将任务部署到合适的集群节点上。在此过程中，控制器服务模块会持续跟踪任务的部署进度，并及时向用户反馈部署状态。

2 基于多维度指标体系的多集群算网信息采集模块

算网信息采集模块是本文平台实现精细化调度的基石，其设计旨在克服传统资源采集系统只能获取如物理GPU卡是否“已分配”的粗粒度局限性，构建一个覆盖全局、实时更新的“资源真值视图”。这个视图不仅要反映资源的静态分配情况，更要实时捕获vGPU核心利用率、显存分配、当前网络带宽和时延等精细化指标，从而实现从传统的“集群分配状态感知”到“节点分配状态感知”的根本性转变。

2.1 多维度算网状态指标体系构建

为支撑上述“资源真值视图”的构建，本文创新性地提出了一个层次分明、覆盖全面的多维度算网指标体系，涵盖GPU算力指标和网络指标。GPU算力指标从集群、设备、vGPU这3个层面，定义了的10项关键算力资源状态指标，网络指标主要包含用户到集群及集群之间的带宽、时延和抖动。集群算网指标体系如表1所示。

表 1　　集群算网指标体系

指标名称	指标意义
ClusterGPUMemoryPercentage	集群显存利用率
ClusterGPUCorePercentage	集群核心利用率
GPUDeviceCoreLimit	设备可分配核心数量
GPUDeviceMemoryAllocated	设备已分配显存量
GPUDeviceMemoryLimit	设备可分配显存量
GPUDeviceSharedNum	设备分配 GPU 数量
ClusterGPUMemoryPercentage	集群显存利用率
ClusterGPUCorePercentage	集群核心利用率
vGPUCorePercentage	vGPU 核心申请比例
vGPUMemoryPercentage	vGPU 显存申请比例
vGPUPodsDeviceAllocated	vGPU 资源申请概览
UserToClusterBandwidth	用户–集群网络带宽
UserToClusterLatency	用户–集群网络时延
UserToClusterJitter	用户–集群网络抖动
UserToClusterPacketLoss	用户–集群网络丢包率
ClusterToClusterBandwidth	集群–集群网络带宽
ClusterToClusterLatency	集群–集群网络时延
ClusterToClusterJitter	集群–集群网络抖动
ClusterToClusterPacketLoss	集群–集群网络丢包率

1) 集群层算力指标。此层指标侧重于宏观算力资源分配和整体健康状态，用于调度决策的第一级筛选和宏观资源均衡判断。关键指标包括集群显存利用率（ClusterGPUMemoryPercentage）和集群核心利用率（ClusterGPUCorePercentage）。

2) 设备层算力指标。此层指标聚焦于物理 GPU 设备的可用性和分配情况。它们直观地展示了单个 GPU 设备的资源使用和承载能力，指导节点级资源分配与整合。关键指标包括设备的已分配核心数量（GPUDeviceCoreAllocated）、设备可分配核心数量（GPUDeviceCoreLimit）、已分配显存量（GPUDeviceMemoryAllocated）以及设备当前分配的 GPU 实例数量（GPUDeviceSharedNum）。

3) vGPU 层算力指标。此指标是实现细粒度调度的关键。指标详细展示了所有使用 vGPU 的容器资源使用情况，包括容器所申请的 vGPU 核心占比（vGPUCorePercentage）和显存占比（vGPUMemoryPercentage）。

4) 用户–集群网络指标。此指标反映用户到集群用户网关的网络状态的情况，用于直观反映用户的网络质量。包括用户到集群的网络链路的带宽（UserToClusterBandwidth）、时延（UserToClusterLatency）、抖动（UserToClusterJitter）和丢包率（UserToClusterPacketLoss）。

5) 集群–集群网络指标。此指标是集群之间的网络链路状态信息，用于反映各个集群的网络质量。主要指标为各集群网关之间的带宽（ClusterToClusterBandwidth）、集群网关之间的时延（ClusterToClusterLatency）、集群网关之间的抖动（ClusterToClusterJitter）和集群网关之间的丢包率（ClusterToClusterPacketLoss）。

2.2　多集群算力采集模块功能设计

GPU 算力采集模块被设计为一个层次分明、高可靠的实时数据管道，以确保细粒度指标的采集效率和数据质量。其架构分为 4 个核心部分：数据源层 Kubernetes API 和 Device Plugin，负责与底层 K8S 集群进行通信的交互层，负责 GPU 信息探测与获取的信息探测层，负责数据处理与格式化以及对应端口开放的数据处理层。各部分协同工作以实现准确采集并提供可用的 GPU 信息。跨集群调度平台功能架构如图 2 所示。

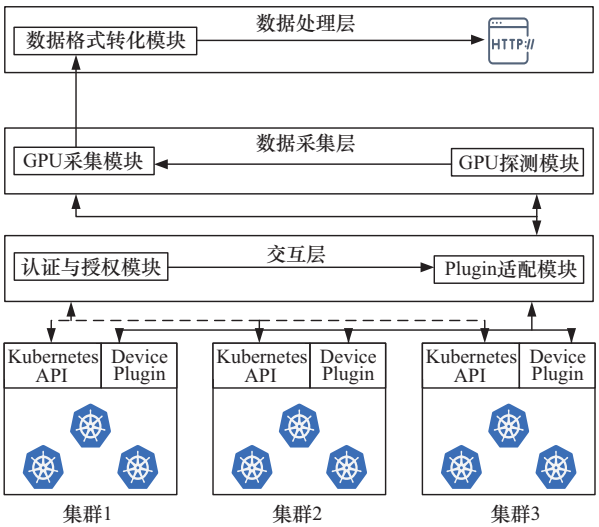


图 2　跨集群调度平台功能架构

1) 数据源层。GPU 算力采集模块的数据来源于部署在各个集群上的 Kubernetes API 以及在 GPU 虚拟化插件部署的 Device Plugin。Kubernetes API 部署于各个集群之上，为整个系统提供了强大的通信与管理能力。GPU 虚拟化插件中的 Device Plugin

则直接与GPU硬件进行交互，能够精准地获取GPU的详细信息。通过与Device Plugin的高效通信，系统不仅可以实时获取GPU的运行状态，还能对容器资源的使用情况进行精准监控。特别是使用了vGPU的容器，Device Plugin能够精确地获取其算力资源的使用情况，从而为资源管理和调度提供准确的数据支持。

2) 交互层。负责与底层的Kubernetes集群进行交互，确保算力采集模块可以与集群及其插件进行通信以获取集群资源信息。主要包含认证与授权、Plugin适配2种功能。认证与授权是整个算力采集模块的基础，主要功能是利用集群的证书通过合法验证，确保只有授权的用户能够访问系统资源，并建立与对应集群的连接。Plugin适配主要负责与底层集群上部署的Kubernetes Device Plugin建立通信连接，为后面信息的交互做准备。

3) 数据采集层。该层负责从底层的Kubernetes集群中收集GPU相关的数据，包含GPU采集模块和GPU探测模块。GPU采集模块通过与Kubernetes集群中的Device Plugin交互，获取GPU的实时状态和性能数据。GPU探测模块则负责筛选出健康的GPU资源，并将这些信息存储起来，供后续的资源调度和故障恢复使用。数据采集层通过对Kubernetes集群的深度集成和对GPU资源的精确监控，能够确保数据的准确性和实时性。

4) 数据处理层。该层负责对数据采集层采集到的GPU信息进行处理和格式化，包含数据格式化与HTTP服务2个主要功能。数据格式化负责将采集到的GPU信息转换为JSON或Prometheus格式的数据，方便对数据进行存储和监控。HTTP服务负责提供对外接口，通过HTTP协议将格式化处理后的数据进行暴露，方便其他模块获取和使用这些数据。

2.3　GPU算力信息采集流程

基于上述GPU信息采集模块，本节主要介绍如何完成GPU信息的采集。GPU算力采集流程如图3所示，整体流程包含6个核心阶段。

1) 建立客户端连接。GPU采集模块在启动时，会首先在指定目录下查找所有的配置文件（Config文件）。这些配置文件包含了连接到各个Kubernetes集群所需的详细信息。模块会使用这些文件来构建Kubernetes配置，并建立与对应集群的连接客户端。每个客户端都会与其对应的集群名称进行关联，并将这些关联信息存储在集群连接对象集中。客户端初始化连接算法如算法1所示。

算法1　客户端初始化连接

输入　无

输出　集群连接对象集ClusterClient

① path = kubernetesconfig// 设置配置config文件路径

② karmadaKubeConfigs = findConfigFiles(path) // 查找所有配置文件

③ clusterClient = ClusterClient struct // 初始化ClusterClient实例

④for kubeConfigPath in karmadaKubeConfigs do

⑤ Config = clientcmd. BuildConfigFromFlags (kubeConfigPath) //构建K8s配置

⑥ Client = kubernetes. NewForConfig(config) // 使用配置创建K8s客户端

⑦ clusterName = extractClusterNam (kubeConfigPath) //提取集群名称

⑧ clusterClient.Clients[clusterName] = client

⑨ return clusterClient

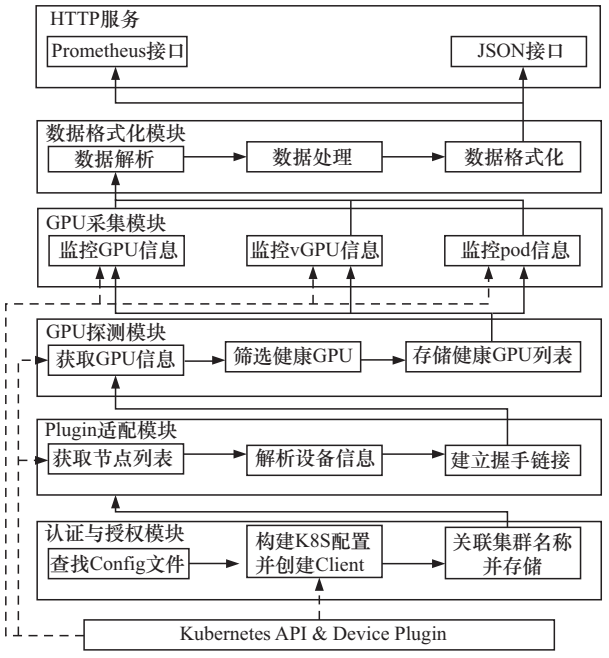


图3　GPU算力采集流程

2) Plugin通信。通过第1步构建的客户端连接集，GPU采集模块与各集群进行通信。Plugin通

信模块获取到各个集群的节点列表，筛选出部署 GPU 虚拟化程序的节点，并与集群上 GPU 虚拟程序所部署的 Kubernetes device plugin 建立握手连接。

3) GPU 探测。GPU 采集模块使用第 2 步构建 Plugin 连接与 Device Plugin 进行通信，获取各集群上所有 GPU 的信息，并通过检查与设备的握手情况以及注册信息筛选出健康的 GPU，并存储在列表中。

4) GPU 算力采集。GPU 算力采集模块会使用第 2 步构建 Plugin 连接来对第 3 步筛选出来的健康 GPU、vGPU 以及使用了 GPU 或 vGPU 的 pod 进行资源使用情况的详细采集。这些信息包括节点名称、GPU 型号、GPU 数量、资源申请量等关键数据。

5) 数据格式化。数据格式化模块会对第 4 步所采集到的资源信息进行解析处理。为满足不同系统和工具对 GPU 信息的获取需求，会将采集到的数据格式化为 JSON 格式与 Prometheus 格式的数据。由于使用的监控系统为 Prometheus，因此需要将采集到的 GPU 数据转换为 Prometheus 支持的时间序列格式，以便 Prometheus 系统定时从接口抓取数据并存储至数据库中。Prometheus 接口如图 4 所示，展示了从接口中获取到的 Prometheus 格式的 GPU 信息。JSON 格式以清晰且结构化的方式呈现数据，既便于数据库存储，也方便大多数应用程序的读取。在信息探测层，模块会将处理后的 GPU 数据按照 JSON 格式进行处理，例如将 GPU 的 ID、显存、计算核心数、使用情况等信息分别赋予不同关键字对应的值作为实际采集和处理后的数据。通过后续步骤暴露 RESTful API，外部系统可以通过 HTTP 请求方便地访问这些 JSON 格式的数据，从而实现数据的交互与共享。JSON 接口如图 5 所示，

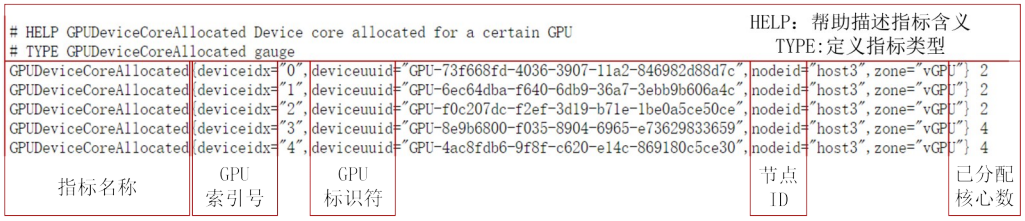


图 4　Prometheus 接口

图 5　JSON 接口

展示了GPUDeviceCoreAllocated指标数据的JSON格式，主要包括指标名称、GPU卡所在节点名称、GPU的索引号与标识符，以及该卡已被分配的vGPU卡数。

6) HTTP端口暴露服务。GPU采集模块会将处理过后的数据进行接口暴露，且针对JSON格式与Prometheus格式的数据提供不同的数据接口。

2.4 网络质量采集模块设计及采集流程

基于上述网络指标采集模块，本节主要介绍如何完成网络指标的采集过程。本文选取了带宽、时延、抖动和丢包率4个核心指标，作为多集群环境下网络资源质量的主要评估参数，流程如下。

1) 部署SRv6网关。本文基于隧道技术和SRv6技术实现不同有限域之间的互联互通。SRv6是一种基于IPv6转发平面的分段路由技术，通过在IPv6报文头中携带段列表的方式，可灵活地控制报文的转发路径，实现路径可编程、流量可控与业务可视化等功能[27]。与传统IPv6隧道方案相比，SRv6不需要维护复杂的隧道状态信息，具有高灵活性、高可靠性和良好的扩展性。系统部署阶段，每个集群都配置一个SRv6集群网关，用于管理集群内部与外部的通信连接；同时用户侧也部署一个SRv6用户网关，用于实现用户与各集群之间的网络连通。通过对SRv6网关进行统一配置和隧道建立，能够保证用户网关与集群网关、集群与集群之间均可进行互联通信，为后续网络指标的探测提供基础环境支撑。

2) 建立探测连接与网络探测。在SRv6网关部署完成后，网络指标采集模块会首先加载系统配置文件，读取所有已注册集群及其对应的网关地址信息。模块启动后会初始化探测连接对象集群（NetworkProbeClient），并为每个集群网关建立与用户网关、其他集群网关之间的通信通道。该模块采用类似ICMP协议的Ping机制对网络进行探测，通过周期性地向目标网关发送探测报文来测量网络质量。每组探测包含5个Ping包，模块会记录每个Ping包的发送与返回时间，计算其往返时延（RTT, round-trip time），并由此获得平均RTT、最大RTT和最小RTT等参数。基于这些数据，系统可以进一步计算传输时延、抖动和丢包率等指标，其计算式为

$$\text{Latency} = \sum_{n=1}^{5} \frac{\text{RTT}_n}{5} \tag{1}$$

$$\text{Jitter} = \max(\text{MaxRTT} - \text{AvgRTT}, \text{MinRTT} - \text{AvgRTT}) \tag{2}$$

$$\text{PacketLoss} = \frac{\text{SendNum} - \text{RecvNum}}{\text{SendNum}} \tag{3}$$

其中，Latency表示传输时延，Jitter表示抖动，PacketLoss表示丢包率。通过持续的周期性探测，模块能够实时感知用户到各集群之间链路的网络质量变化，为后续的编排与调度策略提供数据支撑。带宽指标则通过在集群网关和用户网关部署node-exporter采集，直接暴露给Prometheus。

3) 网络指标采集。网络探测过程结束后，模块会对探测结果进行统计与整理，提取出关键的网络质量指标。采集的内容包括用户网关到集群网关和集群网关之间的时延、抖动、丢包率信息，以及各个集群网关的出入口带宽。模块在采集过程中会自动检测探测结果的有效性，过滤掉超时、无响应或数据异常的探测样本，确保指标数据的准确性和稳定性。同时，模块具备一定的容错与重试机制，当部分探测失败或出现丢包时，会自动进行多轮重试，以尽量减少采集误差。采集得到的网络指标会被统一汇总到缓存队列中，等待后续的数据格式化与持久化处理。网络指标采集算法如算法2所示。

算法2 网络指标采集算法

输入 ClusterList集群列表；SRv6-Config网关配置文件；ProbeNum探测次数（默认5）

输出 NetworkMetric网络指标对象集合

① client = NewNetworkProbeClient () // 初始化网络探测客户端实例

② gatewayInfo = LoadSv6Config (SRv6-Config) // 读取SRv6配置文件，加载用户网关与集群网关地址

③ for cluster in ClusterList do // 遍历集群列表，建立与各网关的探测连接

④ for i = 1 to ProbeNum do // 按探测次数循环，发送Ping并记录时间

⑤ if ResponseReceived then RTT_i = RecvTime_i - SendTime_i else LossCount++ // 计算RTT或记录丢包

⑥ rttStats = ComputeRTTStats (RTTs) // 统计

RTT 数据，计算平均、最大、最小时延

⑦ Latency, Jitter, PacketLoss = CalculateMetrics (rttStats, LossCount) // 按公式计算时延、抖动、丢包率，获取带宽

⑧ validMetrics = FilterAnomalies (Metrics) // 过滤异常数据，触发重试机制

⑨ metric = NewNetworkMetric (validMetrics) // 将有效指标封装为 NetworkMetric 结构体

⑩ NetworkMetricSet = AggregateMetrics (metric) // 汇总所有指标对象，进入格式化与接口暴露流程

4) 数据格式化。采集模块会将原始的探测数据进行统一格式化处理，以便数据库存储和系统访问。模块主要输出 2 种格式：JSON 格式和 Prometheus 格式。JSON 格式的数据结构清晰，便于外部系统读取和存储；Prometheus 格式的数据可直接被 Prometheus 监控系统拉取，用于 Grafana 的实时可视化展示。通过统一格式化机制，网络指标数据可被不同模块复用，满足监控、调度与分析的多场景需求。集群网关网卡入口带宽如图 6 所示，集群网关网卡出口带宽如图 7 所示，展示了带宽的 Prometheus 格式数据。

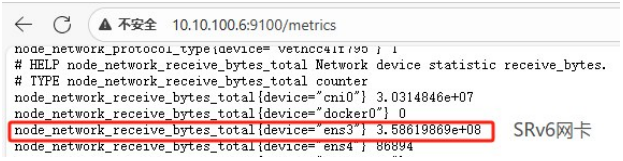


图 6　集群网关网卡入口带宽

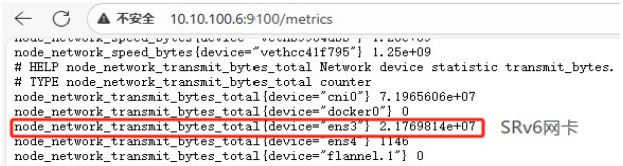


图 7　集群网关网卡出口带宽

5)HTTP 暴露服务。网络指标采集模块会通过 HTTP 服务接口对外提供访问能力。模块为 JSON 格式数据提供 RESTful API，外部模块可通过参数化查询方式获取集群范围内的网络指标，为多集群环境下的资源编排提供实时、可感知的网络信息支撑。JSON 格式的链路信息如图 8 所示。

图 8　JSON 格式的链路信息

3　基于算网感知的多集群节点级 vGPU 算力资源编排部署模块设计

由图 1 可以看出，编排部署模块是整个系统架构的核心，它不仅与前端的用户交互界面进行数据交互，实时接收来自用户提交的各类任务需求，还与 GPU 和网络资源采集模块进行数据交互，保证系统能够及时获取到集群的算力信息和网络信息以完成任务的编排与部署。与现有的多集群编排部署方案不同，本文 GPU 算力资源编排部署模块，利用 2.3 节获取的细粒度 vGPU 信息、网络资源信息，进行跨集群细粒度的资源调度，将决策粒度下沉至“作业到节点”，直接将任务部署到目标集群的目标节点之上，简化了整个部署流程。

3.1　编排部署模块架构

从图 1 整体架构来看，要实现“作业到节点”的跨集群部署，必须采用集中式的编排架构，这种集中控制平面能够提供全局最优的资源视图，确保调度决策不再受限于集群边界。编排部署模块包含编排调度和编排下发 2 个功能，负责将用户的高级

任务需求转化为可执行的底层部署策略，以形成策略生成与执行的闭环控制。

1) 编排调度。编排调度模块分为任务处理和调度决策。任务处理的主要任务是对用户提交的需求进行预处理，是确保系统稳定性和数据一致性的第一道防线。该层对用户提交的需求进行预处理和前置校验，包括检查需求的字段合法性以及任务所需的资源（GPU核心数、内存、网络带宽等）是否超出集群承载上限。校验通过后，算力和网络资源需求信息被转化为符合系统规范的内部编排文件（JSON格式），方便后续流程处理。

调度决策是系统的智能核心，整合了来自GPU算力采集模块的实时节点级vGPU资源信息、网络资源信息与用户任务需求，并调用GPU算力资源调度算法生成最优部署策略。该模块支持多轮决策，特别是当用户需求包含多个副本时，每轮决策完成后系统都会实时更新集群状态，确保后续决策的准确性和全局最优。

2) 编排下发。该模块负责策略的可靠下发与执行。它将调度决策层确定的策略转化为标准的Kubernetes/Karmada资源对象文件（例如，Deployment和PropagationPolicy YAML文件），并利用Karmada作为执行引擎。通过向Karmada服务部署API发送部署请求，系统实现了预定策略的跨集群高效部署。

调度流程的有效性依赖于快速的状态反馈，形成了调度闭环控制。执行层在通过Karmada将任务部署到目标集群或节点后，会及时向控制器模块反馈任务的执行情况。这个反馈回路（数据采集–决策–执行–反馈）保证了系统能够对动态环境进行实时响应，持续优化资源分配。

3.2 编排部署流程

任务编排部署流程如图9所示，遵循严格的6个阶段设计，确保从用户提交到最终部署的每一步都经过优化和校验，即用户需求提交、任务入队、数据处理、调度决策、部署策略生成以及部署完成。

1) 用户需求提交。当用户提交任务需求后，系统会将需求放入校验模块进行校验。若校验成功，任务将进入后续流程；若校验失败，系统会返回错误信息，以便用户进行修改。

2) 任务入队。通过校验的任务被放入先进先出（FIFO, first in first out）队列中，任务处理层依次对出队任务的数据进行处理，生成对应的JSON编排文件，方便后续流程进行处理。

3) 数据处理。数据处理完成后，编排部署模块与数据库以及GPU采集模块、网络信息采集模块进行通信，获取所有集群实时资源信息。

4) 调度决策。调度模块接收到任务需求信息、多集群vGPU资源信息以及网络状态信息后，使用调度算法进行部署策略。

5) 部署策略生成。部署模块使用生成的部署策略进行任务部署，并监控任务的执行情况。

6) 部署完成。当任务执行完毕或者出现异常结束等情况时，部署模块将结果返回给用户，使用户能够及时获取到任务的部署结果。

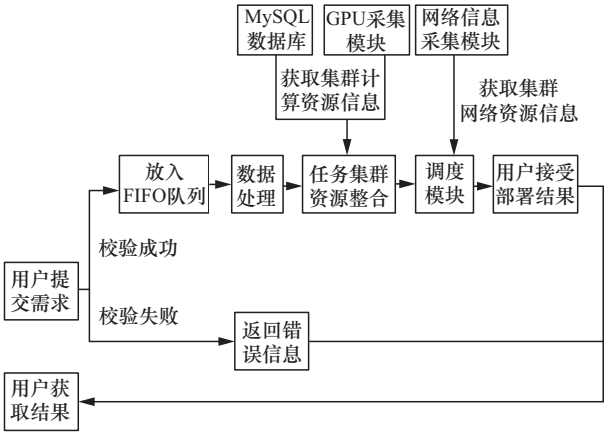


图9 任务编排部署流程

4 平台可视化展示与调度算法对比

本文平台的可视化超越了传统的数据图表展示，其核心理念是提供分层、即时、细粒度的资源态势感知。通过Prometheus进行数据抓取和存储，并结合Grafana构建专业仪表盘，平台旨在让运维人员和用户能够快速理解资源分布、识别潜在瓶颈，从而支持高效的资源管理和优化决策。

4.1 GPU算力大盘信息展示

GPU算力大盘提供集群、设备和容器3个层次的视图，通过提供“集群、节点、GPU卡”等多维度筛选选项，使用户可以快速定位资源问题。

宏观集群概览：该视图面向管理人员，展示集群状态、节点数量以及基础算力资源（CPU、内存、GPU核数、显存）的整体使用量和剩余量。支持宏观的容量规划和集群健康度判断。

中观节点状态：此视图聚焦于节点层面和物理

GPU 卡。展示了每块 vGPU 实例的已分配核心、显存大小、实际核心占用率以及设备共享任务数量。通过对比已分配量和实际利用率，运维人员可以识别资源瓶颈和过度共享的风险。

微观 vGPU 效能：该视图深入到容器层面，展示单个任务所申请和使用的 vGPU 核心百分比和显存百分比。这使用户或开发者能够精确定位应用级别的性能消耗。

为了测试多集群 GPU 算力资源调度平台的功能，本文在实际环境中对其进行了部署与测试。测试环境由 3 个异构 Kubernetes 集群组成，分别配置不同数量的 GPU，型号为 NVIDIA 2080ti 和 NVIDIA 1660 Super。为了实现 GPU 资源使用状况的实时精准监控，部署并测试了基于 Prometheus 和 Grafana 的算力监控系统。

每块 vGPU 卡的核心和显存分配情况如图 10 所示，GPUDeviceCoreAllocated 和 GPUDeviceMemoryAllocate 分别表示已被分配的 CPU 核以及 CPU 内存。集群 3 上部署了 7 个任务，每个任务具体占用的 GPU 核数从 10 核到 50 核不等，以 GPU 内存大小从 1000 MiB 到 2700 MiB 不等。

GPU 设备层面部分指标大盘视图如图 11 所示。GPU Memory Utilization 表盘展示了每块 vGPU 显卡已使用的显存比例，GPU Memory Allocated 表盘展示了每张虚拟卡具体已使用显存的大小。此外，在仪表盘中加入了集群与节点 2 个维度的选项，可

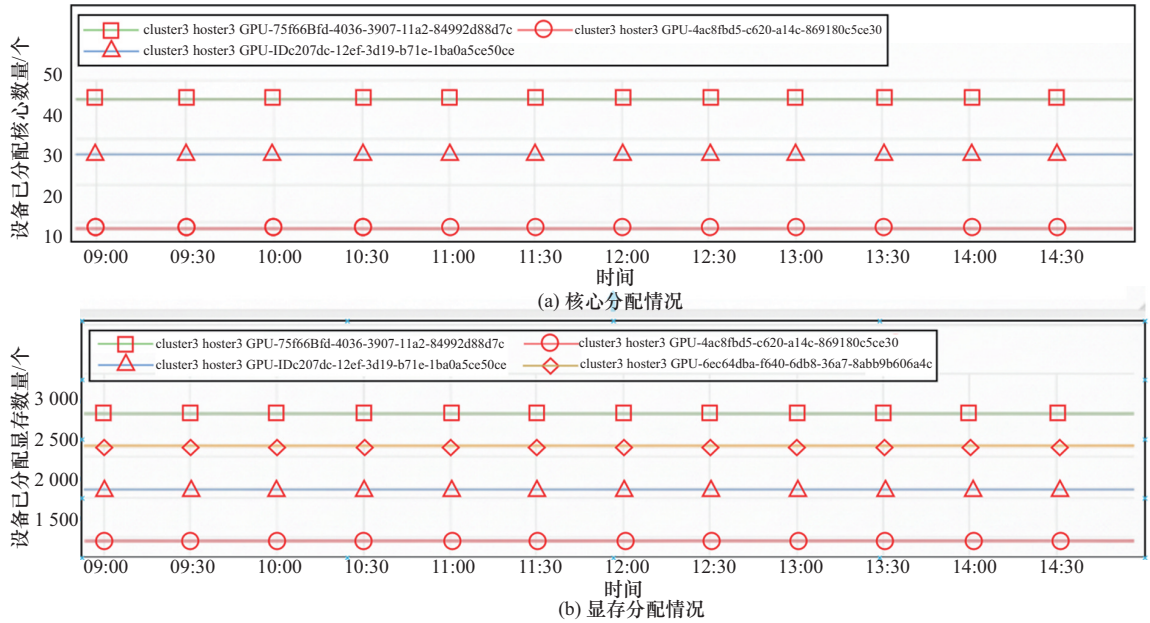


图 10　每块 vGPU 卡的核心和显存分配情况

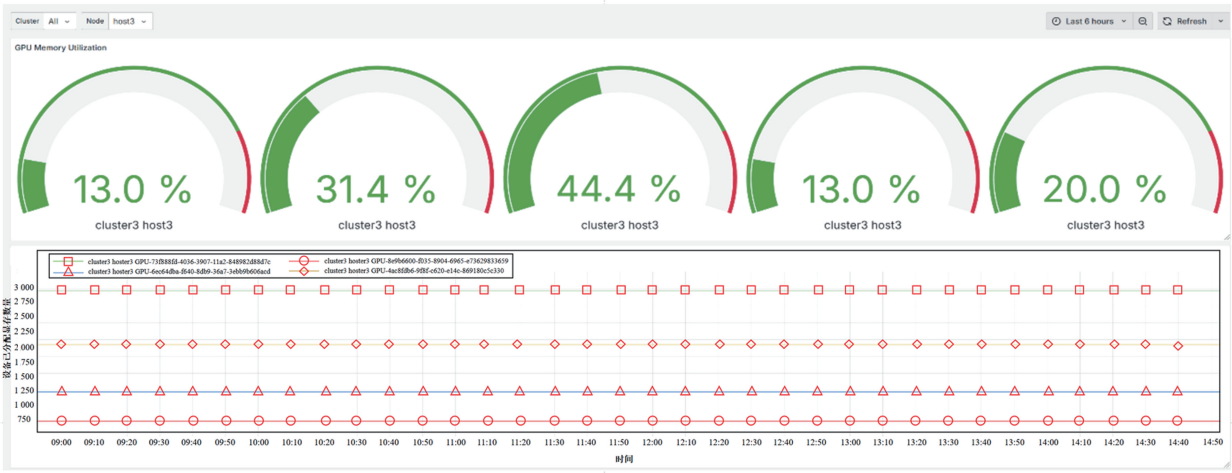
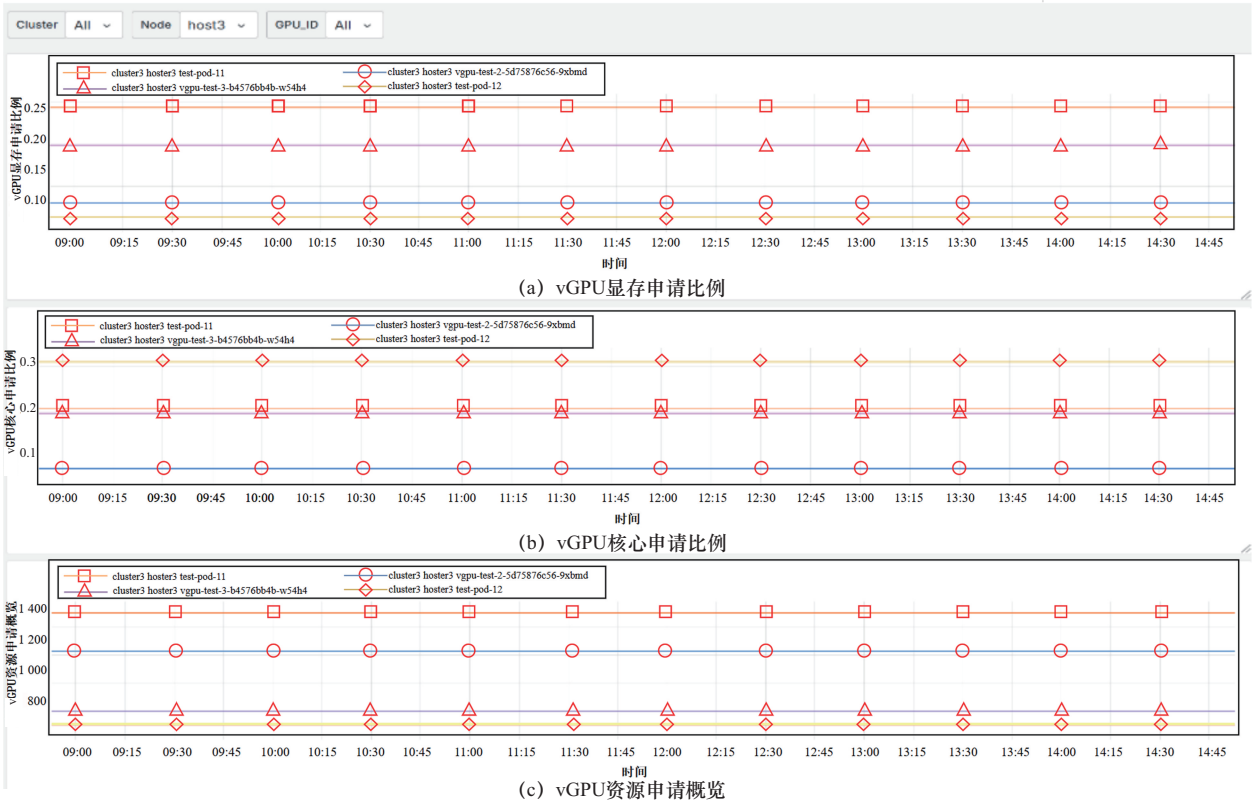


图 11　GPU 设备层面部分指标大盘视图



(a) vGPU显存申请比例

(b) vGPU核心申请比例

(c) vGPU资源申请概览

图 12　容器层面的 vGPU 资源面板

以通过这些选项查看多集群平台下某个集群或某个节点的 GPU 资源状况。

容器层面的 vGPU 资源面板如图 12 所示，其中 vGPUCore-Percentage 和 vGPUMemoryPercentage 表示容器中 GPU 核心和显存的使用百分比，显存大小由 vGPUMemoryAllocated 指标表示。同时，通过集群、节点和 GPU 卡 3 个选项可以筛选和监控不同维度上的 vGPU 资源使用情况。由图 12 可以看出，cluster3 上共部署了 4 个 GPU 资源需求的任务以及每个任务的上述 3 个指标。通过这些不同层级的可视化面板，可以全面了解 GPU 资源的使用情况，优化资源分配，并及时发现潜在的资源瓶颈，进行资源池的扩充。

上述结果表明，本文方案能够实时获取多集群节点级 vGPU 核心、显存的实时状态，实现多集群 GPU 的统一纳管，为任务的编排部署提供了基础。

4.2　网络状态信息展示

集群网关和用户网关之间的链路信息如图 13 所示，展示了 3 个集群网关以及 1 个用户网关之间的链路质量信息。gw-1、gw-2 和 gw-3 为集群网关，gw-4 为用户网关；3 个集群网关相互连接，用户网关与 gw-3 相连。可以清楚展示集群网关之间以及用户网关与集群 3 的网关之间链路的带宽、时延、抖动和丢包信息。

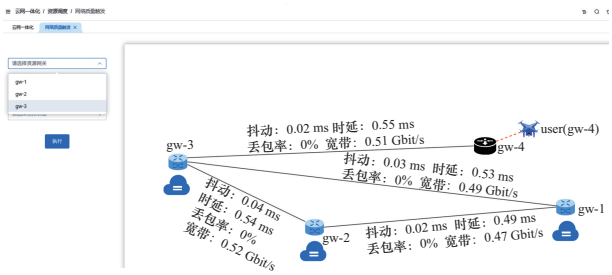


图 13　集群网关和用户网关之间的链路信息

4.3　Web 平台可视化

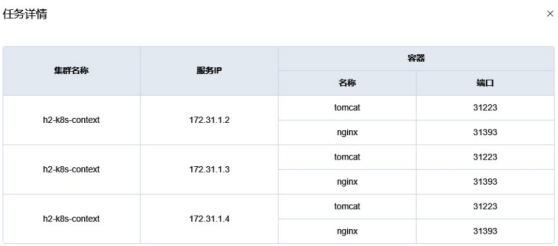
除了上述 GPU 算力资源和网络资源可视化展示，本文在 Web 平台加入了任务调度交互界面与集群其他维度信息的可视化展示。

通过任务部署界面，向控制器下发任务请求，通过填写需求名称、用户所在网关、任务优先级以及任务模板 4 个信息完成任务的部署。可视化任务部署界面如图 14 所示，用户可以指定任务的名称，设置任务的优先级以确定任务部署的先后顺序，确定任务的部

署网关（默认为网关4，即用户网关；网关1~3为集群网关），并完成应用部署请求信息的提交。

图 14　可视化任务部署界面

任务部署以后，通过任务详情界面查看任务部署的集群、对应的服务 IP 以及开放的端口号。使用查询得到的 IP 以及端口，实现对服务的访问。任务详情如图 15 所示，3 个集群上分别部署了 IP 地址为 172.31.1.2、172.31.1.3、172.31.1.4 的 3 个任务，每个任务中包含 2 个容器，通过集群任意节点的 IP 地址加上容器的端口号来访问目标容器。



集群名称	服务IP	容器	
		名称	端口
h2-k8s-context	172.31.1.2	tomcat	31223
		nginx	31393
h2-k8s-context	172.31.1.3	tomcat	31223
		nginx	31393
h2-k8s-context	172.31.1.4	tomcat	31223
		nginx	31393

图 15　任务详情

集群层面 GPU 算力资源信息如图 16 所示，更为详细地展示了各集群的计算资源使用情况，包括每个集群的 CPU 核数、CPU 内存、GPU 核数、GPU 内存 4 个部分信息的使用量和剩余量。这里展示的仅为简要的 GPU 信息，更为具体的情况则需要访问 4.1 节提供的可视化界面进行查看。集群网络质量信息展示见 4.2 节。

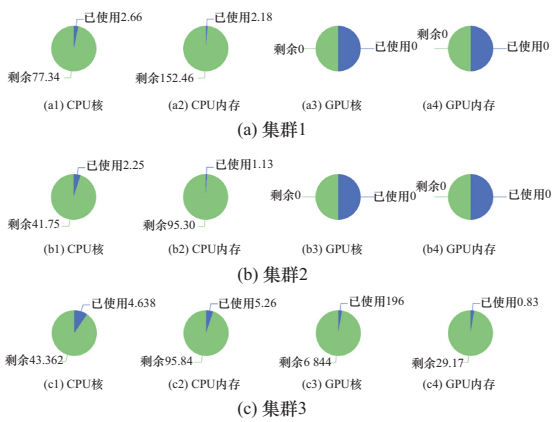


图 16　集群层面 GPU 算力资源信息

4.4　基于算网状态感知的 vGPU 调度算法对比分析

平台架构的设计核心是开放性和策略自适应性。为了满足不同场景下的资源管理需求，平台提供了开放、可插拔的调度算法接口。这意味着新的调度策略，无论是启发式、优化算法还是强化学习模型，都可以作为插件集成，不需要修改核心架构，从而保证系统能够适应未来调度算法的快速演进。目前，已在平台上实现的具体算法包括 DDPG、BestFit、Greedy、Random 和 Utilization。需要注意的是，所有算法在进行任务调度时，首先基于算网状态感知信息过滤掉不满足条件的集群，然后在剩余集群中进行 vGPU 的调度。

DDPG 算法：传统的强化学习算法，结合了 Q 学习和策略梯度方法。在多集群 GPU 算力资源调度中，DDPG 算法通过与环境的交互来学习最优的任务部署策略。智能体根据当前多集群的剩余算力资源状态、网络资源状态以及任务请求的资源需求采取行动（即决定将任务部署到哪个集群）。该算法利用深度神经网络来近似 Q 函数和策略函数，通过不断探索和利用环境反馈的奖励信号（如资源利用率），逐步优化调度策略，以实现高效的算力资源分配和任务执行。

BestFit 算法：一种经典的启发式调度算法，其核心思想是将任务分配给最合适的集群，以提高资源利用率。在多集群 GPU 算力资源调度场景下，BestFit 算法首先遍历所有可用的集群，寻找资源空闲量最低且能够满足任务资源需求的集群。具体来说，当有一个任务需要部署时，算法会检查每个集群的剩余 GPU 算力资源以及网络连接状态，在所有网络资源满足要求的集群中，选择资源空闲量最小的集群来部署该任务。这种方法旨在紧凑地安排任务，减少资源浪费，提高整体算力资源的利用率，特别适用于任务资源需求相对固定且集群资源变化不频繁的情况。

Greedy 算法：基本策略是在搜索满足任务资源需求的集群过程中，查找到一个满足任务需求的集群时便停止搜索并进行任务的部署。这种算法不考虑全局最优解，而是着眼于局部最优选择，以加快任务调度的速度。

Random 算法：当有任务需要部署时，该算法会在所有满足任务算力和网络资源需求的集群中随

机选择一个集群进行任务部署。由于缺乏对资源利用率和任务性能的考虑，Random算法可能导致资源分配不均衡和任务执行效率低下。

Utilization算法：首先获取所有满足网络资源需求的集群，并计算每个集群的资源余量比例（即剩余资源量与总资源量的比值）以及任务资源需求比例（即任务所需资源量与集群总资源量的比值），然后选择两者比例最接近的集群来部署任务。这种方法旨在实现资源的均衡利用，避免某些集群资源过剩而其他集群资源不足的情况，从而提高整个多集群系统的资源效率和性能。

Volcano-divided算法：依托Volcano调度框架的多任务并行处理能力，通过对任务进行分片后分发至不同集群或节点，从而实现算力资源的并行利用。具体而言，当有大规模任务提交时，调度器会将其拆解为多个子任务，根据各集群的可用GPU资源情况进行分配。这样可以有效提升任务的执行并发度，并减少单集群的资源负载压力，适合需要快速响应和分布式执行的大规模并行作业场景。但由于任务被分片，子任务间可能存在依赖关系，因而在任务同步与结果整合方面需要额外开销。

Volcano-duplicated算法：同样基于Volcano调度框架，其核心思想是将任务副本同时分发到多个集群或节点，以提高任务完成的成功率和系统的容错性。在多集群GPU调度中，这种方法能够保证即使部分集群出现故障或任务失败，仍有其他副本能够继续执行，因而具有较强的可靠性和鲁棒性。该策略特别适合对结果可靠性要求较高的场景，但其劣势在于会占用更多的算力资源，可能导致整体资源利用率下降，不适合算力紧张的环境。

为全面验证本文平台在细粒度数据支撑下的调度能力，实验在实际物理环境上进行了部署和测试。测试环境由3个异构Kubernetes集群组成，配置了NVIDIA 2080ti和NVIDIA 1660 Super等不同型号的GPU。这种异构性验证了平台对多集群环境的通用纳管和精细调度能力。随机生成任务，模拟对GPU算力资源的不同大小需求，以及对集群网络资源（时延、抖动、丢包）的需求。然后分别采用DDPG、BestFit、Greedy、Random和Utilization等5种调度算法进行部署，并以“任务部署成功个数”作为关键效能指标进行评估。同时，为了验证多集群节点级资源调度与传统集群级调度的优劣，在实际物理环境中又部署了Vocalno调度平台，以进行对比实验。

不同算法任务部署成功个数如表2所示，DDPG算法任务成功部署个数最多，Random部署的任务数最少。这是因为Random算法随机选择集群进行部署，导致后续任务所需的算力资源不足，因此最为浪费资源。BestFit和Utilization算法能够从节省资源的角度去进行任务部署，因此部署任务个数仅次于DDPG。Greedy算法则是因为寻求局部最优解，最终部署的任务个数仅多于Random方法。此外，本文的节点级调度平台任务部署成功数量多于Volcano Global v1.11的2种集群编排调度模式，即Volcano-divided算法和Volcano-duplicated算法。实验结果表明，本文GPU算力资源调度平台支持多种调度算法，并提升多集群资源利用率和任务部署成功率。

表2 不同算法任务部署成功个数

算法	成功个数/个
Greedy	54
BestFit	54
Random	53
Utilization	58
DDPG	59
Volcano-duplicated	39
Volcano-divided	53

5 结束语

本文设计并实现了一个基于算网状态感知的通用多集群GPU算力调度平台，旨在解决大规模AI基础设施在面对异构深度学习任务时，资源调度粒度粗放、缺乏统一细粒度资源视图的挑战；构建了一个多维度算网指标体系，算力指标覆盖集群层、设备层和关键vGPU层的算力资源状态，网络指标涵盖用户到集群、集群到集群的网络状态。该体系通过实时采集vGPU核心利用率和显存分配、网络带宽和时延等细粒度指标，重塑了传统粗粒度的资源感知范式，为实现性能感知型调度提供了精确的数据支撑。此外，平台采用集中式编排架构，并设计了基于算网感知的跨集群节点级vGPU编排部署模块，实现了从传统“作业到集群”至“作业到节点”的根本性转变，显著增强了物理GPU的共享效率和整体资源利用率。实证分析表明，平台克服了Volcano等现有主流方案在多集群GPU调度中存在的局限，为大规

模AI算力基础设施提供了高效、可控、精细化的创新解决方案。在此基础上，未来的研究方向应包括异构集群的拓扑感知调度，以满足异构的大规模AI模型训练日益依赖跨集群联合算力的需求。

参考文献：

[1] MEHRISH A, MAJUMDER N, BHARADWAJ R, et al. A review of deep learning techniques for speech processing[J]. Information Fusion, 2023, 99: 101869.

[2] SCHULZE BUSCHOFF L M, AKATA E, BETHGE M, et al. Visual cognition in multimodal large language models[J]. Nature Machine Intelligence, 2025, 7(1): 96-106.

[3] 中国信息通信研究院. 中国算力发展指数白皮书 [R]. 2023.
China Academy of Information and Communications Technology. China Computing Power Development Index White Paper [R]. 2023.

[4] 中国算力大会. 中国算力发展报告 [R]. 2024
China Computing Power Conference. China Computing Power Development Report [R]. 2024.

[5] Kubernetes. Production-grade container orchestration [R]. 2023-03-21.

[6] HINDMAN B, KONWINSKI A, ZAHARIA M, et al. Mesos: a platform for fine-grained resource sharing in the data center[C]//Proceedings of the 8th USENIX Symposium on Networked Systems Design and Implementation (NSDI 11). Berkeley: USENIX Association, 2011: 295-308.

[7] MESOS A. Program aginst your datacenter like it's a single pool of resources [R]. 2023-03-23.

[8] 徐迪. Clusternet：一款开源的跨云多集群云原生管控利器！ [R]. 2024-03-23.
XU Di. Clusternet: An Open-Source Cross-Cloud Multi-Cluster Cloud-Native Management Tool! [R]. 2024-03-23.

[9] 华为云计算. Volcano 火山:容器与批量计算的碰撞[R]. 2020-06-05.
Huawei Cloud Computing. Volcano: Where Containers Meet Batch Computing [R]. 2020-06-05.

[10] KHALLOULI W, HUANG J W. Cluster resource scheduling in cloud computing: literature review and research challenges[J]. The Journal of Supercomputing, 2022, 78(5): 6898-6943.

[11] DELGADO P, DIDONA D, DINU F, et al. Job-aware scheduling in eagle: divide and stick to your probes[C]//Proceedings of the Seventh ACM Symposium on Cloud Computing. New York: ACM Press, 2016: 497-509.

[12] DELGADO P, DINU F, KERMARREC A M, et al. Hawk: hybrid datacenter scheduling[C]//Proceedings of the USENIX Annual Technical Conference. Berkeley: USENIX Association, 2015: 499-510.

[13] KARANASOS K, RAO S, CURINO C, et al. Mercury: hybrid centralized and distributed scheduling in large shared clusters[C]//Proceedings of the USENIX Annual Technical Conference. Berkeley: USENIX Association, 2015: 485-497.

[14] THINAKARAN P, GUNASEKARAN J R, SHARMA B, et al. Phoenix: a constraint-aware scheduler for heterogeneous datacenters[C]//Proceedings of the 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS). Piscataway: IEEE Press, 2017: 977-987.

[15] KAUFMANN M, KOURTIS K, SCHUEPBACH A, et al. Mira: sharing resources for distributed analytics at small timescales[C]//Proceedings of the 2018 IEEE International Conference on Big Data (Big Data). Piscataway: IEEE Press, 2018: 231-241.

[16] BOUTIN E, EKANAYAKE J, LIN W, et al. Apollo: Scalable and coordinated scheduling for cloud-scale computing [C]// Proceedings of the 11th USENIX symposium on operating systems design and implem entation (OSDI 14). Berkeley: USENIX Association, 2014: 285-300.

[17] GAREFALAKIS P, KARANASOS K, PIETZUCH P, et al. Medea: scheduling of long running applications in shared production clusters[C]// Proceedings of the Thirteenth EuroSys Conference. New York: ACM Press, 2018: 1-13.

[18] PARK G. A generalization of multiple choice balls-into-bins[C]//Proceedings of the 30th Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing. New York: ACM Press, 2011: 297-298.

[19] SCHWARZKOPF M, KONWINSKI A, ABD-EL-MALEK M, et al. Omega: flexible, scalable schedulers for large compute clusters[C]// Proceedings of the 8th ACM European Conference on Computer Systems. New York: ACM Press, 2013: 351-364.

[20] GRANDL R, ANANTHANARAYANAN G, KANDULA S, et al. Multi-resource packing for cluster schedulers[J]. ACM SIGCOMM Computer Communication Review, 2014, 44(4): 455-466.

[21] NIU Z J, TANG S J, HE B S. Gemini: an adaptive performance-fairness scheduler for data-intensive cluster computing[C]//Proceedings of the 2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom). Piscataway: IEEE Press, 2015: 66-73.

[22] VAVILAPALLI V K, MURTHY A C, DOUGLAS C, et al. Apache hadoop YARN: yet another resource negotiator[C]//Proceedings of the 4th annual Symposium on Cloud Computing. New York: ACM Press, 2013: 1-16.

[23] VERMA A, PEDROSA L, KORUPOLU M, et al. Large-scale cluster management at google with borg[C]//Proceedings of the Tenth European Conference on Computer Systems. New York: ACM Press, 2015: 1-17.

[24] 张旭, 赵晨熙, 冯川, 等. 基于带内网络遥测数据的流量预测与负载均衡[J]. 通信学报, 2025, 46(8): 41-52.
ZHANG X, ZHAO C X, FENG C, et al. Traffic prediction and load balancing based on in-band network telemetry data[J]. Journal on Communications, 2025, 46(8): 41-52.

[25] 卫敏, 赵倩颖, 唐静, 等. 算力网络信息通告技术研究综述[J]. 通信学报, 2025, 46(9): 17-31.
WEI M, ZHAO Q Y, TANG J, et al. Summary of research on information notification technology of computing power network[J]. Journal on Communications, 2025, 46(9): 17-31.

[26] 鄢智勇, 陈浩, 丁立戈, 等. 面向算力互联网的高可用智算互联互通平台体系架构[J]. 通信学报, 2025, 46(9): 65-80.
YAN Z Y, CHEN H, DING L G, et al. Architecture of high availability intelligent computing interconnection platform for computing Internet[J].

Journal on Communications, 2025, 46(9): 65-80.

[27] 金磊, 陈贤, 王佶, 等. SRv6 域内路由保护方法分析[J]. 通信学报, 2024, 45(S2): 102-107.
JIN L, CHEN X, WANG J, et al. Analysis of SRv6 intra-domain routing protection method[J]. Journal on Communications, 2024, 45(S2): 102-107.

[作者简介]

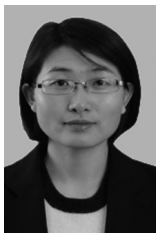
胡亚辉（1982– ），女，河南信阳人，博士，中国矿业大学（北京）副教授、硕士生导师，主要研究方向为数据中心网络优化传输、边缘计算与边缘智能、算力网络、网络流量分析与优化、5G等。

张宸康（1996– ），男，河南商丘人，中国矿业大学（北京）硕士生，主要研究方向为云计算与算网融合技术。

王越嶙（1999– ），男，新疆哈密人，中国矿业大学（北京）硕士生，主要研究方向为云计算与算网融合技术。

洪雨琛（2000– ），男，湖南湘潭人，中国矿业大学（北京）硕士生，主要研究方向为云计算与算网融合技术。

范鹏飞（1982– ），男，内蒙古巴彦淖尔人，中国科学院计算机网络信息中心高级工程师、硕士生导师，主要研究方向为未来网络架构与技术、算网融合技术与应用等。



宋俊平（1985– ），女，河北沧州人，博士，中国科学院计算机网络信息中心助理研究员，主要研究方向为算力网络、网络人工智能等。

周旭（1976– ），男，四川成都人，博士，中国科学院计算机网络信息中心研究员、博士生导师，主要研究方向为计算机网络体系结构、5G移动通信、网络人工智能技术等。