软件学报 ISSN 1000-9825, CODEN RUXUEW
[doi: 10.13328/j.cnki.jos.007202] [CSTR: 32375.14.jos.007202]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

基于性能建模的深度学习训练任务调度综述*

杨紫超[1,2], 吴 恒[2,3,4], 吴悦文[2], 张文博[2,3,4]

[1](中国科学院大学, 北京 100049)

[2](中国科学院 软件研究所 软件工程技术研究开发中心, 北京 100190)

[3](中国科学院大学南京学院, 江苏 南京 211135)

[4](中科南京软件技术研究院, 江苏 南京 211135)

通信作者: 张文博, E-mail: zhangwenbo@otcaix.iscas.ac.cn

摘 要: 近年来, 深度学习研究成果在全球范围内得到广泛应用. 为了提高大规模深度学习模型的训练效率, 业界通常采用建设 GPU 集群并配置高效的任务调度器的策略. 然而, 深度学习训练任务具有性能异构性和放置拓扑敏感性等复杂性能特性. 对性能无感知的调度容易导致资源利用率低下、训练效率差等问题. 为了应对这一挑战, 近期涌现出大量基于性能建模的深度学习训练任务调度器. 这些调度器通过构建精确的性能模型, 深入了解任务的复杂性能特性, 并据此设计更优化的调度算法, 从而形成更高效的调度方案. 首先基于建模设计思路, 对目前调度器使用的性能建模方法进行分类综述. 随后, 根据调度器利用性能建模的调度优化途径, 对现有的任务调度工作进行系统性分析. 最后, 对性能建模与调度在未来的研究方向进行展望.

关键词: 深度学习训练; 性能建模; 任务调度

中图法分类号: TP18

中文引用格式: 杨紫超, 吴恒, 吴悦文, 张文博. 基于性能建模的深度学习训练任务调度综述. 软件学报. http://www.jos.org.cn/1000-9825/7202.htm

英文引用格式: Yang ZC, Wu H, Wu YW, Zhang WB. Survey on Task Scheduling of Deep Learning Training Based on Performance Modeling. Ruan Jian Xue Bao/Journal of Software (in Chinese). http://www.jos.org.cn/1000-9825/7202.htm

Survey on Task Scheduling of Deep Learning Training Based on Performance Modeling

YANG Zi-Chao[1,2], WU Heng[2,3,4], WU Yue-Wen[2], ZHANG Wen-Bo[2,3,4]

[1](University of Chinese Academy of Sciences, Beijing 100049, China)

[2](Technology Center of Software Engineering, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China)

[3](University of Chinese Academy of Sciences, Nanjing, Nanjing 211135, China)

[4](Nanjing Institute of Software Technology, Nanjing 211135, China)

Abstract: In recent years, research achievements in deep learning have found widespread applications globally. To enhance the training efficiency of large-scale deep learning models, industry practices often involve constructing GPU clusters and configuring efficient task schedulers. However, deep learning training tasks exhibit complex performance characteristics such as performance heterogeneity and placement topological sensitivity. Scheduling without considering performance can lead to issues such as low resource utilization and poor training efficiency. In response to this challenge, a great number of schedulers of deep learning training tasks based on performance modeling have emerged. These schedulers, by constructing accurate performance models, delve into the intricate performance characteristics of tasks. Based on this understanding, they design more optimized scheduling algorithms, thereby forming more efficient scheduling solutions. This study begins with a modeling design perspective, providing a categorized review of the performance modeling methods employed by current schedulers. Subsequently, based on the optimized scheduling approaches from performance modeling by schedulers, a systematic analysis of existing task scheduling efforts is presented. Finally, this study outlines prospective research directions

* 基金项目: 山东省重大创新工程 (2021CXGC010101); 国家自然科学基金 (62302489)
收稿时间: 2023-09-25; 修改时间: 2023-11-06, 2024-02-07; 采用时间: 2024-04-09; jos 在线出版时间: 2024-06-20

for performance modeling and scheduling in the future.

Key words: deep learning training; performance modeling; task scheduling

深度学习是借助人工神经网络架构, 对数据进行表征训练和目标预测推理的前沿技术, 其在医学图像分析[1]、自动驾驶[2]、人脸识别[3]、语音翻译[4]及自然语言对话[5]等诸多领域具有广泛的应用. 在深度学习的数据表征训练环节, 通常需处理大规模样本数据, 并依赖 GPU 等高性能加速器进行并行计算, 这使得 GPU 集群[6,7]成为推动深度学习应用的核心基础设施. 然而, GPU 集群内包含大量异构计算资源, 并需应对多样化的深度学习训练任务, 因此, 如何合理规划任务与资源, 以提升训练效率及资源利用率, 已成为产业界和学术界共同面临的挑战.

任务调度作为一种在多维约束下对上层任务与底层计算资源进行优化匹配的技术, 对于提升训练效率及资源利用率具有重要作用. 近年来, 基于性能建模的调度方法受到了广泛关注, 它通过构建精确的任务性能模型, 为调度决策提供有力支持. 本文旨在对这些调度器中使用的性能建模与调度方法进行系统性的综述, 以帮助研究者了解现有性能建模设计思路, 以及调度方法如何利用性能建模以提升调度效果.

尽管已有综述工作对大数据和高性能计算中的任务性能建模与调度进行了概述[8−11], 但它们未充分考虑深度学习训练中迭代执行与异构算力等独特性能特性, 因此难以直接应用. 另外, 部分综述聚焦于深度学习推理任务的调度[12−14], 而推理阶段与训练阶段的优化目标存在显著差异, 前者更关注实时性, 后者则更强调如何加速大规模任务的完成. 还有一些工作关注于单任务性能调优和显存优化[15−17], 这与本文关注的集群整体优化视角有所不同. 虽然少量综述工作对深度学习训练任务调度进行了总结[18,19], 但主要依据调度目标进行分类, 而本文则着重探讨如何利用性能建模来提升调度效果.

本文第 1 节介绍相关背景及挑战, 分析深度学习训练特性, 阐述任务调度利用性能建模进行优化的方式, 并介绍性能建模与调度的挑战. 第 2 节依据建模方法切入点对任务性能建模方法进行了分类分析. 第 3 节根据如何利用性能建模来优化调度的策略, 对各种调度方法进行了系统的分类和讨论. 第 4 节对未来的研究方向进行了展望. 第 5 节对全文进行总结.

1 背景与挑战

1.1 基于性能建模的任务调度流程

图 1 展示了基于性能建模的深度学习训练任务调度的工作流程. 用户提交深度学习训练任务后, 性能建模模块会筛选出未进行建模的深度学习训练任务, 并为每个任务建立性能模型. 性能建模本质是确定每个任务在任意资源配置下的性能指标, 其中资源配置可包含 GPU 类型、GPU 数量、GPU 通讯带宽、CPU 核数、内存大小等. 本文关注的性能指标主要有 3 类, 包括任务执行时间、吞吐率和收敛效率, 其中吞吐率指单位时间内训练迭代次数.

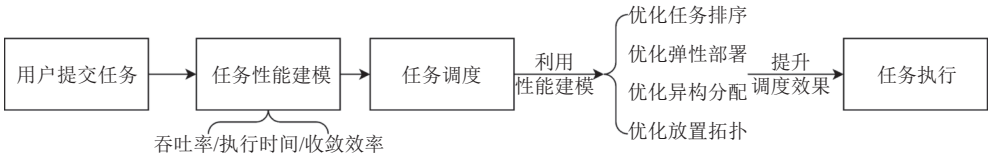


图 1 基于性能建模的任务调度流程

任务性能模型建立后, 进入任务调度阶段. 调度的本质是根据不同性能指标约束进行任务与资源的优化匹配决策. 在这一阶段, 调度模块会充分利用从性能建模阶段获取的任务性能模型, 通过不同调度优化策略, 来制定高效的调度算法, 从而为每个任务确定最佳的资源配置. 本文总结出 4 种调度优化策略, 包括: 优化任务排序、优化弹性部署、优化异构分配以及优化放置拓扑. 通过运用这些策略, 可以实现更加高效的任务调度, 进而显著提升任务执行效率和集群利用率.

1.2 深度学习训练特性

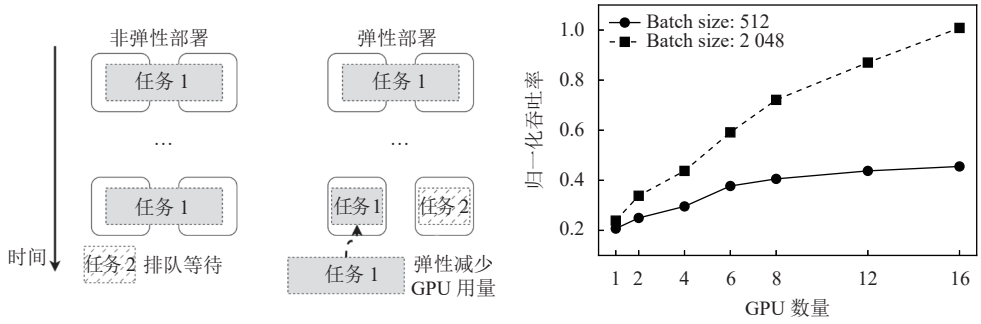
与大数据和高性能计算任务相比, 深度学习训练任务具有其独特的特性, 需要在任务性能建模和调度中特别

考虑并加以利用. 这些特性可划分为运行特性与资源使用特性两类. 其中, 任务运行特性为任务周期迭代性, 而任务资源使用特性则包括以下 3 类: 任务性能异构性、任务弹性部署性、任务放置拓扑敏感性. 特性的具体解释如下所述.

● 任务周期迭代性[20]. 深度学习训练表现为一个周期性的、迭代式的算子计算过程. 每次迭代都计算一个批次 (batch) 的训练数据, 计算量较为固定, 因此表现出较强的周期迭代性. 这一特性是深度学习训练任务性能建模的理论基础: 可通过对深度学习训练的单个迭代周期进行性能建模, 然后将其推广至整个训练过程. 同时, 只要资源配置不变, 任务吞吐率也具有稳定性. 因此, 调度过程无需担心任务性能模型随时间变化.

● 任务性能异构性[7,21]. 深度学习训练在不同的硬件上表现出明显的性能异构性. 由于不同代际 GPU 在架构、核心频率、核心数量上存在差异, 训练时可能会产生数十倍的性能差异. 此外, 不同任务的性能加速比也存在显著差异. 除了 GPU, CPU、内存等其他硬件也会影响数据预处理与传输速度, 从而对性能产生一定影响.

● 任务弹性部署性[22]. 深度学习训练任务在运行期间可以弹性地调控资源分配量以及任务超参数. 通过数据并行, 深度学习训练可利用多块 GPU 进行分布式训练, 并根据需要弹性地调整 GPU 使用数量. 如图 2(a) 所示, 在非弹性部署下, 任务 2 需要排队等待资源释放; 而在弹性部署下, 任务 1 可以通过弹性减少资源占用, 使任务 2 能立即运行. 图 2(b)[22]展示了 ResNet18[23]在不同 GPU 分配量和 batch size 下的吞吐率. 可以看出, 随着 GPU 数量的增加, 训练速度会加快, 但其吞吐率增长会逐渐放缓, 表现出边际效应. 而不同 batch size 的资源利用率不同, 从而影响性能. 因此, 调度方法需要根据任务性能模型和集群资源容量来智能地弹性调整 batch size 与 GPU 数量, 以提高集群的整体训练效率.



(a) 弹性部署示例 (b) ResNet18不同GPU数量的归一化吞吐率

图 2 任务弹性部署性

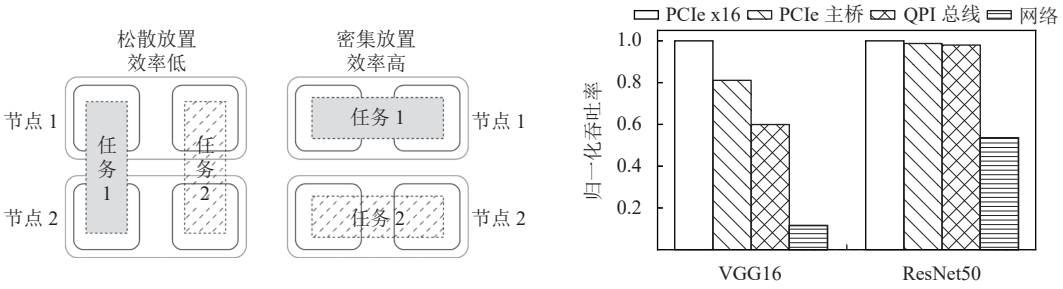
● 任务放置拓扑敏感性[24]. 不同的任务放置拓扑会对通信速度产生影响, 进而影响训练性能. 如图 3(a) 所示, 松散放置导致跨节点任务间的网络通信效率降低, 而密集放置则通过节点内高速总线提升效率. GPU 间通信方式多样, 包括 NVLink、PCIe x16、PCIe 主桥、QPI 总线或网络. 它们的带宽差异极大, 例如第 3 代 NVLink 带宽可达 600 GB/s[24], 而普通网络带宽仅有 1 GB/s. 同时, 训练模型对通信带宽的需求有差异. 如图 3(b)[25]所示, ResNet50[24]在跨 PCIe x16、PCIe 主桥、QPI 总线通信时的性能差距较小, 而 VGG16[26]则表现出显著的差距. 因此, VGG16 对放置拓扑敏感性更高. 当通过网络进行通信时, VGG16 与 ResNet50 都显示出较高的性能损失.

1.3 任务调度如何利用性能建模优化调度效果

性能建模在调度优化中具有至关重要的作用. 与对性能无感知的调度方法相比, 具备性能感知能力的调度方法能够预测任务执行后的性能, 从而有针对性地设计调度优化策略. 具体来说, 可通过优化任务排序、优化放置拓扑、优化异构分配和优化弹性部署的策略, 显著提升调度效果. 本节以优化任务排序为例, 探讨如何利用性能建模来增强任务调度的效果.

实验采用了 Tiresias[27]提出的 SRSF (shortest-remaining-service-first) 调度方法, 其核心目标是提高任务的完成效率. 该方法优先选择剩余服务时间最短的任务执行, 旨在避免队头阻塞, 从而加速任务完成. 为了计算剩余服务

时间, 此方法需要获取任务执行时间. 因此, 实验假设任务执行时间可通过性能建模来得到, 并对比了建模的平均相对误差 (mean relative error, MRE) 为 0、10%、25%、50% 和 75% 时的不同调度情况. 同时, 实验也与性能无感的最少获得服务优先 (LAS)[27]调度方法进行了对比.



(a) 放置拓扑敏感性示例　　(b) VGG16, ResNet50 不同放置拓扑的归一化吞吐率

图 3　任务放置拓扑敏感性

实验数据来自两个商业 GPU 集群: 一个是来自阿里巴巴集团的 GPU 集群 PAI[6] (包含 6 742 块 GPU), 持续跟踪了 2 个月的数据; 另一个是来自微软公司的 GPU 集群[7] (包含 2 490 块 GPU), 也持续跟踪了 2 个月的数据 (称为 Philly 数据集). 这两个数据集都涵盖了超过 100 000 个任务. 在数据处理阶段, 从每个数据集中抽取连续提交的 10 000 个任务, 并记录它们的任务提交和执行时间. 最后, 在一个包含 64 个同构 GPU 的仿真集群中, 按照跟踪数据中的提交时间来模拟这些任务的调度情况. 实验采用平均任务完成时间 ($\overline{JCT}$) 作为调度性能指标, 该值越小表示任务完成效率越高. 调度结果如图 4 所示, 其中, 0、10%、25%、50% 和 75% 是当 MRE 设置为对应值时的表现, LAS 则对应最少获得服务优先情况下的表现. 当建模的执行时间与实际情况误差在 10% 以内时, 调度效果明显优于其他情况. 当误差超过 25% 时, 调度效果显著下降. 完全不使用性能建模的调度方法表现最差. 这些结果清晰地表明, 准确的性能建模对于优化任务排序和提升整体调度效果至关重要.

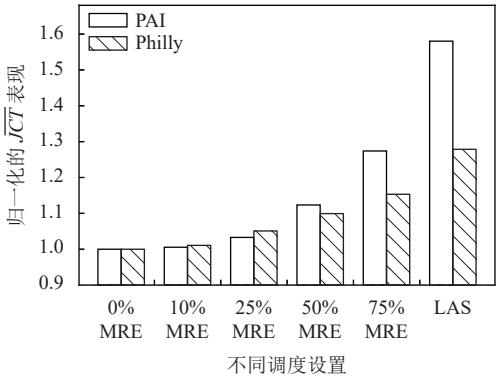


图 4　不同调度设置对 $\overline{JCT}$ 的影响

本实验中, Tiresias[27]调度方法通过建模任务执行时间合理规划任务优先级, 与性能无感的调度方法相比, 显著提升训练效率. 其他的调度方法可能利用性能建模采取其他的优化策略来进行调度优化. 第 3 节将详细介绍这些方法如何利用性能建模进行优化.

1.4 挑　战

本节旨在深入剖析性能建模与调度所面临的挑战. 从宏观视角出发, 性能建模的核心问题在于如何精准映射资源与任务性能之间的关系, 而调度则需要解决任务与资源之间的最优匹配问题. 这两者共同关注的两大关键要素是任务和资源. 然而, 这两大要素均展现出极强的多样性, 从而为性能建模与调度带来了诸多挑战, 具体表现在以下几个方面.

● 任务多样性. 深度学习训练任务的种类繁多. 从任务内视角看, 深度学习模型具有较强的多样性. 组成模型的算子种类多达上千种, 同时模型间算子数量也存在巨大差异. 例如, 简单的多层感知机模型可能只包含几十个串行连接的算子, 而复杂的 BERT-large[28]则包含高达 18 758 个具有复杂依赖关系的算子, 算子数量及其连接结构的复杂程度差距极大. 这种多样性使得性能建模难以统一化、标准化. 从任务外视角看, 任务的执行时间与资源需求同样具有较强的多样性. 例如, 在 Philly[7]数据集中, 占用 4 块及以上 GPU 的任务数仅占 11.09%, 但其执行时间占比却高达 36.6%. 此外, 数据集中短任务 (<30 min) 占 55.98%, 长任务 (>12 h) 占 8.85%, 甚至有少量任务超过一天. 如何在时长与资源需求差异巨大的任务间实现平衡调度具有挑战性.

● 资源配置多样性. 如第 1.2 节任务性能异构性、任务弹性部署性、任务放置拓扑敏感性所述, 不同的异构资源、弹性部署以及放置拓扑会对性能产生影响. 这些资源配置的组合可能性极多. 对于性能建模而言, 如何提升模型的泛化能力以适应如此广泛的资源配置是一大难题. 对于调度而言, 这一挑战可进一步细化为以下 3 个方面.

(1) 异构亲和度多样性. 如任务性能异构性所述, 尽管更快的 GPU 对所有任务都更有利, 但不同任务的异构加速比存在差异. 如何有效地平衡和分配不同数量和不同性能的异构资源, 以最大化全局的训练效率是一个严峻的挑战.

(2) 弹性部署配置多样性. 如任务弹性部署性所述, 深度学习训练任务的弹性部署涉及任务资源量 (如 GPU 数量) 和训练超参数配置 (如 batch size、学习率) 的弹性变化. 这些部署配置组合多, 增加了调度算法的搜索空间, 提高了调度难度.

(3) 放置拓扑多样性. 如任务放置拓扑敏感性所述, 即使任务资源配置与参数保持不变, 不同的放置拓扑也会通过影响通信速度间接影响任务性能. 同时, 任务间的干扰会进一步加剧性能的不确定性. 如何在放置拓扑敏感性不同的任务间进行合理组合, 并在多样性的影响因素 (如 PCIe、网络等) 间寻求最优放置是调度面临的又一大挑战.

2 任务性能建模

2.1 方法分类模型

本节将对深度学习训练任务的性能建模方法进行系统介绍. 如图 5 所示, 这些方法可以分为 4 大类: (1) 基于实测剖析的方法; (2) 基于任务元信息的方法; (3) 基于计算图结构的方法; (4) 基于可组合算子的方法.

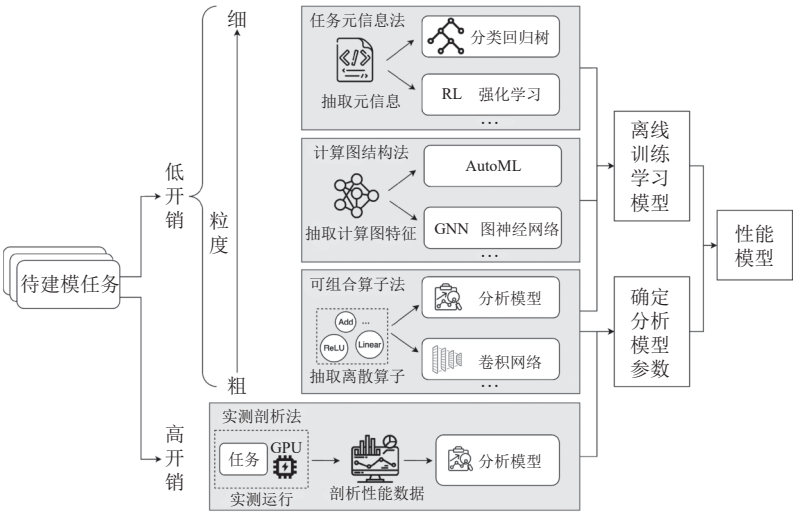


图 5 任务性能建模方法分类模型

从建模开销的角度来看, 基于实测剖析的方法需要实际运行任务以进行建模, 这不可避免地会产生较高的资源分配与任务运行开销. 而其他方法则属于低开销类别, 可以直接利用任务信息进行建模. 根据建模所需的任务信

息粒度, 这些低开销的方法可以进一步细分为 3 类: 基于任务元信息的方法、基于计算图结构的方法和基于可组合算子的方法. 其中, 基于任务元信息的方法仅需要任务的高层描述信息进行建模, 基于计算图结构的方法需要任务的计算图结构信息, 而基于可组合算子的方法则需要最细粒度的算子级别信息.

从性能模型的建模方法来看, 实测剖析法主要采用了基于分析的建模, 而基于计算图结构与任务元信息的方法则主要基于数据驱动的思路, 通过离线训练的方式进行建模; 基于可组合算子的方法则对以上两类建模方法均有所使用.

接下来, 将详细介绍这 4 大分类中的方法. 在后续表格中, “建模选型”列展示该性能建模方法使用的模型类型. “分布式训练”列表示该方法是否支持对分布式训练任务的性能建模. “建模指标”列表示该方法支持建模的性能指标. “任务/数据集”列简要展示实验验证其建模效果时所选取的建模对象. “效果”列简要展示该方法的建模性能, 其中误差指的是 MRE 误差.

2.2 基于实测剖析的方法

表 1 展示了具有代表性的基于实测剖析的任务性能建模方法. 这类方法遵循以下流程: (1) 通过分析离线任务的实测性能数据, 建立一个带参数的任务性能分析模型; (2) 使待建模任务在一种或多种实际硬件上运行多次迭代, 以收集稳定的运行时特征; (3) 剖析实测的运行时特征以确定参数, 从而得到针对该任务的定制化性能模型.

表 1 基于实测剖析的任务性能建模方法

成果	建模选型	分布式训练	建模指标			任务/数据集	效果
			执行时间	吞吐率	收敛效率		
Chronus[20] Hydra[29] Allox[30]	分析模型	—	√	—	—	14种CNN、RNN模型	执行时间误差低 (约5%)
Cynthia[31]	分析模型	√	√	√	—	ResNet32[23]、VGG-19[26]	吞吐率误差低 (约1.6%–6.3%)
Synergy[32]	分析模型	—	—	√	—	ResNet18[23]	吞吐率误差低 (约3%)
Optimus[33]	回归模型	√	√	—	√	InceptionBN[34]、Seq2Seq[35]等5种模型	收敛效率误差一般 (约9%)
Shockwave[36]	回归模型	√	√	√	—	Accordion[37]、SimiGrad[38]	执行时间误差偏高 (约13%)
Daydream[39]	仿真模型	√	—	√	—	AMP[40]、FusedAdam[41]	吞吐率误差低 (约2.8%)
Proteus[42]	仿真模型	√	—	√	—	9种CNN、RNN模型	吞吐率误差低 (约4.3%)

基于实测剖析的方法主要针对任务多样性提出解决方案. 面对内部结构复杂多样的任务, 可通过设计统一的方法来提取与任务性能相关的运行时特征, 从而消除任务在结构层面的差异性, 有效地解决这一挑战. 实测剖析法即采取该思路, 并结合回归模型、仿真模型以及基于规则的分析模型, 根据实测运行时特征对任务的性能进行建模. 主要工作具体内容如下.

一些工作通过分析任务实测运行时特征与性能的关系, 设计基于规则的分析模型进行建模. 其中一些工作直接利用深度学习训练的周期迭代性进行简单的实测建模, 如 Chronus[20]、Hydra[29]及 Allox[30]等方法. 它们假设任务的总运行迭代次数已知为 n, 然后通过实测多次迭代, 获取迭代的平均时间 $\overline{t_i}$, 从而预估最终的执行时间为 $n\times\overline{t_i}$. 然而, 这些简单实测方法开销较大, 不能有效支持分布式训练、大量异构资源等资源配置种类较多的场景. 为了对分布式训练提供支持, Cynthia[31]针对 PS 架构, 对 BSP (bulk synchronous parallel) 与 ASP (asynchronous parallel) 的分布式训练数据同步方式进行了详细的底层分析, 最后分别基于超参数、计算量、通信量以及硬件资源分配量构建分析模型, 通过实测补全模型参数来建模各类分布式配置的性能. 针对大量异构配置的情况, Synergy[32]关注如何降低实测开销. 该工作观察到, 当 GPU 类型不变时, CPU 与 GPU 的分配比例与训练性能之间存在可预测的变化趋势, 仅需实测少量不同 CPU/GPU 分配比例下的性能即可快速分析其他情况的性能, 减少了实测开销. 这些基于规则的分析模型能够较准确地应对特定场景 (如 PS 架构) 的性能建模. 然而, 由于规则与场景存在强耦合关系, 当场景发生变化时, 这些方法需经过人工重新设计才能适配, 场景适应能力较差.

另一些工作通过设计回归模型, 并利用实测剖析拟合参数的方式进行建模. 例如 Optimus[33]根据大量实测模型的收敛曲线人工设计回归模型, 随后在线上阶段通过实测任务早期收敛状态拟合回归模型参数, 从而建模收敛效率. Shockwave[36]针对 batch size 弹性调控任务进行性能建模. 该方法将任务迭代过程中的 batch size 变化视为轨迹, 并结合 batch size 调控优化算法 (Accordion[37]、SimiGrad[38]) 的领域知识, 使用贝叶斯概率回归模型, 在线上运行阶段持续实测来更新概率回归模型, 从而拟合轨迹并建模任务执行时间. 这些基于回归模型的方法需要实测较多迭代, 才可获得较好的回归效果, 因此实测开销较大.

部分工作通过仿真执行的方式进行建模. 例如, Daydream[39]关注训练优化机制 (如 AMP[40]、算子融合[41]) 对性能的影响. 它利用底层的核函数剖析工具 CUPTI 建立 CPU、GPU、网络通信的调用依赖路径分析模型, 并通过对优化机制的仿真执行, 建模优化后的吞吐率. Proteus[42]针对分布式训练的复杂并行通讯策略, 建立了名为“Strategy Tree”的数据结构, 能够统一建模训练时的分布式通信决策空间, 并建立一个分层拓扑感知的仿真执行器 (hierarchical topo-aware executor). 通过实测执行路径并补足参数后, 对计算与通信进行仿真来建模任务吞吐率. 基于仿真执行的方法在建模准确度方面具有优势, 这是由于这类方法以实测获取的执行路径为基础进行仿真, 更贴近真实执行环境. 其弱点则在于无法应对硬件资源的变化. 当 GPU 不同时, 必须重新实测以获取新的执行路径后再进行仿真.

虽然实测剖析法能够较好地面对任务多样性挑战, 但在应对资源配置多样性挑战时表现较差. 这是因为实测得到的性能数据无法在不同资源配置下进行迁移和泛化, 导致这类方法需要在多种资源配置下分别进行实测, 从而产生较高开销. 调度器在使用实测剖析法时需要考虑实际场景是否能够接受这样的开销.

2.3 基于任务元信息的方法

表 2 列举了一些基于任务元信息的性能建模方法. 这些方法聚焦于任务的元信息, 如模型名称、超参数设置 (batch size、学习率等)、资源申请量/分配量、放置拓扑以及任务提交时间等, 但不深入探究模型的内部细节. 这类方法通常采取数据驱动模型, 并遵循以下流程: (1) 首先, 收集大量任务执行历史数据, 提取任务元信息作为特征; (2) 随后, 利用这些特征数据来训练学习模型; (3) 最后, 利用该模型直接获取其他任务的性能模型.

表 2 基于任务元信息的任务性能建模方法

成果	建模选型	分布式训练	建模指标			任务/数据集	效果
			执行时间	吞吐率	收敛效率		
MLaaS[6]	分类回归树	√	√	—	√	PAI[6] 400 000+任务	执行时间误差一般 (约8.9%)
Helios[43]	分类回归树	√	√	—	—	Helios[43] 158万+任务	执行时间误差偏高 (约10.8%)
Harmony[44]	强化学习+MLP	√	√	√	—	Seq2Seq[35]、CTC[45]	吞吐率误差一般 (约9.8%)
GENIE[46]	回归模型	√	√	√	—	6种CNN、RNN模型	执行时间误差偏高 (约12%)

基于任务元信息的方法在应对资源配置多样性挑战方面表现出色. 面对多样化的资源配置, 这类方法能够通过学习大量不同资源配置下的任务性能数据, 有效地训练出适应性强的数据驱动模型. 具体来说, 它们利用分类回归树模型、强化学习模型、回归模型等模型, 专注于将任务的高层元信息及各类资源配置作为特征进行训练, 从而在变化的资源配置中展现出优越的性能.

在具体实践中, 一些方法倾向于利用任务提交的时序规律元信息进行建模. 这些方法针对大规模 GPU 集群进行研究, 任务时序规律较强. 例如, MLaaS[6]基于阿里巴巴的 PAI 大规模集群 (6 742 个 GPU), 利用任务提交时间的周期性与反复性, 将任务提交时间、资源请求量以及用户分组信息等作为特征, 建立基于 CART[47]分类回归树的机器学习模型来建模任务的执行时间信息. 类似地, Helios[43]专注于对 SenseTime 的 6 416 块 GPU 集群上的任务进行性能建模. Helios 着重于时间特征的分析, 利用提交时间在日、周、月中潜在的时序规律进行特征建模, 并结合资源需求量、放置拓扑等其他元信息, 建立基于 GBDT 的机器学习模型来建模任务执行时间. 以上基于时序规律的性能模型在大规模的真实任务和商业集群上经过验证, 能够取得较好建模效果. 然而由于时序规律不在任意集群间具有普遍性, 这种模型仅能够针对特定集群进行预测, 适用范围较窄.

相较之下, 中小规模集群的任务在时序方面规律较弱, 因此针对这类集群的工作更关注从任务资源使用特征挖掘元信息. Harmony[44]针对 PS 架构下分布式任务之间的通信干扰, 基于强化学习方法建模. 该方法利用历史数据拟合多层感知机模型, 将任务的参数服务器数量、计算节点数量作为元信息特征, 将吞吐率作为预测目标. 随后, 将吞吐率预测模型作为任务放置的奖励函数, 通过在线上运行收集数据, 不断训练强化学习策略, 反馈式地调节吞吐率预测模型. 与前述方法不同, GENIE[46]针对不同放置拓扑的通信开销进行细化分析, 将分布式任务训练拆解为计算与同步通讯的两个过程, 随后分别为每个过程建立回归模型, 通过收集大量离线任务的放置拓扑, 以及 GPU 占用量, 通信资源占用量等元信息, 来拟合不定系数, 从而在线上为其他任务进行建模. 以上这些不依赖时序规律的方法对适用的集群没有要求, 然而由于仅从资源视角提取特征, 难以对任务内部的变化进行感知, 因此面对多样化的任务时处于劣势.

尽管这类方法对解决资源配置多样性存在优势, 但这类方法主要从元信息视角提取任务特征, 并不关心深度学习模型的内部结构. 这导致在面对未见过的新模型结构时泛化性能较差, 因此在应对任务多样性挑战时存在困难.

2.4 基于计算图结构的方法

表 3 展示了基于计算图结构的任务性能建模方法. 深度学习模型实质上是由算子构成的有向无环计算图, 因此这类方法端到端地从计算图整体结构中提取特征, 并直接利用这些特征构建学习模型来预测性能指标.

表 3 基于计算图结构的任务性能建模方法

成果	建模选型	分布式训练	建模指标			任务/数据集	效果
			执行时间	吞吐率	收敛效率		
Horus[48]、文献[49]	XGBoost	—	—	√	—	19种CNN模型	吞吐率误差偏高 (约11.3%)
DNNAbacus[50]	AutoML[51]	—	—	√	—	29种DNN模型	吞吐率误差一般 (约7.1%)
DNNPerf[52]	GCN	—	—	√	—	5种CNN模型	吞吐率误差偏高 (约12.8%)
Driple[53]	GCN	√	√	√	√	10种DNN模型	执行时间误差一般 (约8%) 迁移时间较低 (比重新训练降低7.3倍)

为应对任务多样性的挑战, 可通过深入理解任务内部结构特征, 设计统一的特征提取方法, 提高模型对多样化任务的适用性. 为应对资源配置多样性, 可通过提升性能模型在不同资源配置之间的迁移适配能力, 提升对多样资源配置的适应性. 基于计算图结构的方法采取以上思路来应对这两类挑战. 这类方法能够利用图特征编码、图特征压缩等通用性强的技术, 有效地从计算图结构中提取特征, 避免在面对多样性任务时出现方法不适用的情况. 面对资源配置多样性时, 这类方法通常采取迁移学习来进行模型的适配. 具体的, 基于计算图结构的方法通常采取图神经网络、决策树、AutoML[51]等模型进行特征提取与性能建模, 包括如下方面.

一些研究采用人工设计特征工程, 并辅以机器学习模型的方法进行建模. 比如, Horus[48]和 Yeung 等人[49]对 GPU 共享任务的吞吐率进行预测. 这些方法将计算图结构特征压缩提取为一维向量, 包括卷积层数量、线性层数量、待训练权重量、浮点计算量、batch size 等. 随后使用 XGBoost 决策树模型预测 GPU 共享时的利用率波动, 进而预测吞吐率. 而 DNNAbacus[50]则设计了创新的网络结构化矩阵 (network structural matrix) 数据结构来表示计算图特征, 该数据结构大幅压缩了特征空间. 随后, 该方法利用 AutoML[51]自动搜索合适的模型结构, 训练能够同时建模吞吐率与显存占用量的模型. 以上方法需要人工设计特征工程, 以便进行计算图的特征提取, 其建模在特定数据集下可能具有较优的效果. 但是当出现新类型的算子、计算图结构时, 可能需花费人力对特征工程进行更新、细化, 具有较高设计成本.

另一些研究则探索使用图神经网络 (GNN) 直接对计算图进行特征编码和建模. 比如, DNNPerf[52]利用图卷积神经网络 (GCN), 分别使用算子类型与算子执行依赖的特征端到端地预测吞吐率. 该方法提出了基于注意力机制的节点-边编码器, 对算子间数据传递进行细化特征提取, 从而提高了建模准确度. 然而, DNNPerf 的限制在于其仅支持在单一种类资源配置下的建模. 针对此问题, Driple [53]考虑到弹性部署、放置拓扑的多样性可能导致泛化性差, 因此通过图分组、聚合的技术压缩特征量, 并利用迁移学习的方法使预训练模型能够快速适应各类不同种类

的资源配置. 以上基于 GNN 的方法能够利用 GNN 自身强大的关系表征能力, 对特征进行自动提取, 有效降低人工特征工程的繁琐工作量.

尽管基于计算图的方法在应对任务多样性及资源配置多样性挑战时都存在一定的解决方案. 但这些方案也存在一定的弊端. 对于任务多样性挑战, 当面对庞大的计算图时, 基于计算图结构的特征提取过程会造成信息损失, 从而对建模准确度产生影响. 对于资源配置多样性挑战, 虽然使用迁移学习可以保障在不同资源配置下的建模准确度, 但该过程需要在新的资源配置下采集大量数据, 因此存在较高的数据采集开销. 不过, 此类开销仅在集群资源发生变化时出现 (例如采购新型 GPU 等情况), 因此当集群资源变化少时, 基于计算图结构的方法仍然存在优势.

2.5 基于可组合算子的方法

表 4 展示了基于可组合算子的任务性能建模方法. 这些方法的流程通常包括以下步骤: (1) 将深度学习模型的计算图分解为独立可组合的细粒度算子; (2) 根据离线获取的算子性能数据, 建立针对算子的性能模型, 并对待建模任务的每个算子获取其独立的性能指标; (3) 将其独立的算子性能指标有效整合, 以获取任务整体性能.

表 4　基于可组合算子的任务性能建模方法

成果	建模选型	分布式训练	建模指标			任务/数据集	效果
			执行时间	吞吐率	收敛效率		
Habitat[54]	分析模型	—	—	√	—	DCGAN[55]等5种CNN模型	吞吐率误差偏高 (约10.8%)
SEER[56]	分析模型	√	—	√	—	10种CNN模型	吞吐率误差偏高 (约12.1%)
PerfNet[57] PerfNetV2[58]	一维卷积模型+MLP	—	—	√	—	8种CNN模型	吞吐率误差偏高 (约13.1%)

基于可组合算子的方法主要致力于解决任务多样性挑战. 在面对多样化的任务时, 这类方法将其视为有限算子的组合. 通过对少量算子进行建模, 再将这些算子灵活组合, 即可为各种任务提供有效的性能模型, 从而解决多样性挑战. 这类方法的认识基础在于, 目前深度学习模型的常用算子种类是有限的. 通过对这些常用算子进行建模, 可将算子的知识复用于不同任务, 显著降低建模难度. 在具体实施上, 这类方法可采用基于规则的分析模型, 或基于轻量的深度学习模型进行建模, 主要有以下内容.

一些工作采用基于规则的分析模型直接建模算子的性能. 例如, Habitat[54]发现一些算子在同架构 GPU 上的核函数实现不变, 其性能变化趋势相似. 因此, Habitat 假设已知某一类 GPU 上的任务吞吐率, 并通过不同代际 GPU 之间的各项参数 (如计算频率、显存带宽等) 比例来直接建模其他 GPU 上的吞吐率. SEER[56]也着眼于卷积算子的建模, 从 GPU 核函数执行的角度, 从底层对卷积算子计算流程进行详尽分析. 根据输入数据量及参数的不同, SEER 将卷积算子分为计算受限、显存受限和利用率低下 3 种类型, 并分别建立对应的分析模型, 建模算子执行时间. 以上基于规则分析的方法具有无数据采集开销、预测延迟极低的优势, 但当其面临新硬件、新算子时, 需重新调优、更新分析模型, 对人力要求较高.

另一些工作采取了数据驱动的思路, 设计轻量的深度学习模型进行建模. 例如 PerfNet[56]与 PerfNetV2[58], 它们提取了模型和硬件特征. 软件特征主要包括算子的输入输出数据维度、激活函数、优化器类型、batch size 等. 硬件特征包括使用 GPU 的 CUDA 核心数、基础频率、内存频率、带宽以及峰值 FLOPs 等. 在考虑了所有可能的特征因素后, 它们建立了 1 维卷积与多层感知机相结合的轻量深度学习模型, 收集算子性能数据来拟合每个算子的吞吐率数据. 类似地, Habitat[54]对于无法利用分析模型建模的核函数变化算子, 建立多层感知机来建模算子运行时间. 尽管以上基于数据驱动的方法将 GPU 规格信息 (CUDA 核心数, 频率, FLOPs 等) 作为特征训练, 然而实际影响性能的因素还包括 GPU 架构、功耗调度等复杂因素. 因此, 面临未训练过的 GPU 时, 建模准确度较差, 需采取迁移学习或重新训练的方式才能获取更准确的建模结果.

在应对任务多样性挑战方面, 基于可组合算子的方法通过知识复用展现优势; 然而这类方法主要从算子角度建模, 忽略了算子执行之外的其他开销 (例如框架开销、内存拷贝开销等), 这对建模准确度存在一定影响. 在应对资源配置多样性挑战方面, 基于数据驱动的可组合算子方法通常具有较为简单和轻量的模型, 因此可基于迁移学

习, 快速地在不同资源配置间迁移使用. 同时, 与基于计算图结构的方法相比, 其所需的训练数据也较少 (仅需采集算子数据, 无需在大量模型上采集数据), 因此数据采集开销也相对较低, 面对多样化的资源配置具有较强适应能力. 相比之下, 基于规则的分析模型在面对新的资源类型时, 可能需花费人力重新设计/调优算子的性能分析规则, 因此适应能力较弱. 但值得注意的是, 相较于实测剖析法, 可组合算子法在人工设计负担上有所降低, 因为它主要针对算子性能进行分析, 而大量算子在同架构/同世代 GPU 之间存在共性 (具有相同核函数). 分析规则可在这些 GPU 之间复用, 从而降低了人工设计的复杂性.

2.6 对比与小结

表 5 展示了对本节所详述的任务性能建模方法在多个关键维度上的细致对比. 在建模思路上, 实测剖析法主要依赖于人工设计的分析模型, 而任务元信息与计算图结构法则更倾向于采用数据驱动的模型. 可组合算子法在两种思路上均有所涉猎, 显示出其方法的灵活性.

表 5 任务性能建模方法对比

性能建模方法分类	模型使用开销	任务信息粒度	建模思路	建模误差	任务多样性挑战	资源配置多样性挑战
实测剖析法	高	一	分析模型	低	优势	劣势
任务元信息法	低	粗	数据驱动模型	一般	劣势	优势
计算图结构法	低	中等	数据驱动模型	一般	优势	较优
可组合算子法	低	细	分析模型 数据驱动模型	偏高	优势	优势 (数据驱动模型)/较劣 (分析模型)

在建模准确度方面, 实测剖析法由于紧密依托任务的实测运行时特征, 更贴近真实运行环境, 因此其建模误差相对较低. 其他三类方法在建模准确度上表现相当, 其中可组合算子法稍显逊色, 这主要归因于它在建模时未能充分考虑算子以外的运行时开销. 尽管如此, 实测剖析法的高准确度是以更高的性能建模使用开销为代价的, 尤其是对于新任务而言, 其实测开销是不可避免的.

针对任务多样性挑战, 任务元信息法显现出明显的短板. 这主要是因为其关注的任务信息层次较为粗放, 难以深入挖掘多样化任务内部的细微特征. 在面对资源配置多样性挑战时, 依赖分析模型的方法由于受限于人工制定的规则, 其适应能力明显受限. 在采用数据驱动模型的方法中, 可组合算子法凭借其轻量级的数据需求实现了快速迁移, 从而更轻松地适应了新的资源配置环境. 相比之下, 计算图结构法则涉及大量的数据采集开销, 在与能够直接利用丰富任务性能数据进行驱动的任务元信息法相比时, 其在面对多样资源配置时稍显不足. 然而值得一提的是, 计算图结构法相较于需要人工重新设计分析模型的实测剖析法仍具有较大的优势.

综上所述, 各种任务性能建模方法在不同挑战和场景下均展现出了独特的优劣势, 为实际应用提供了多样化的选择与思考角度.

3 基于性能建模的任务调度

3.1 方法分类模型

基于性能建模的调度方法能够利用性能模型, 通过不同调度优化策略来提升调度效果. 这些策略主要包括: (1) 优化任务排序, 以应对任务多样性挑战; (2) 优化异构分配, 以应对异构亲和度多样性挑战; (3) 优化弹性部署, 以应对弹性部署配置多样性挑战; (4) 优化放置拓扑, 以应对放置拓扑多样性挑战. 这些方法借助性能模型, 通过这 4 类策略对调度过程进行优化, 从而有效应对调度挑战. 相较于性能无感的调度方法, 可显著提升调度效果. 本节根据这 4 类策略, 分类探讨了这些调度方法如何利用性能建模来增强调度效果.

本节的表格中, “调度目标”列明确了每种调度方法旨在提升的特定调度效果, 包括完成效率、截止时间、收敛效率、成本效率以及公平性. “性能建模方法”列指方法采用的性能建模途径, 对应于第 2 节所述的 4 类方法. “性能建模需求”列则展示了调度算法所需的性能指标. “性能建模利用方式”列展示了该方法利用性能建模设计的调度算法类型. “实验设置”列则简要展示了该调度方法实验验证的集群与任务规模. “效果”列则简要展示该方法

的调度效果.

3.2 优化任务排序

表 6 展示了利用性能建模来优化任务排序的调度方法. 这类调度方法主要针对任务多样性挑战进行解决. 面对多样化的任务时, 通过对任务执行时间与资源需求进行精确感知, 可设计出更优化的任务执行顺序, 从而权衡不同类型任务需求, 有效解决挑战. 这类方法采取以上思路, 借助性能建模, 设计了基于优先级、线性规划、图匹配的算法来优化任务排序, 进而提升调度效果. 具体而言, 有以下方法.

表 6 基于性能建模优化任务排序的调度方法

成果	调度目标	性能建模方法	性能建模需求			性能建模利用方式	实验设置		效果
			执行时间	吞吐率	收敛效率		集群规模	任务规模	
Tiresias[27]	完成效率	假设已知	√	—	—	设计优先级算法	60	480	降低平均JCT (比Apache Yarn[59]降低78%)
Themis[60]	公平性	实测剖析	√	—	—	设计优先级算法	256	15 000+	提升FTF公平性 (比Tiresias[27]提升2.25倍)
Allox[30]	完成效率	实测剖析	√	√	—	设计图匹配算法	40	10 000+	降低平均JCT (比SRTF降低95%)
Chronus[20]	截止时间	实测剖析	√	—	—	设计线性规划算法	120/96	30 000+	提升截止时间满足率 (比EDF[61]提高75%)

一些方法通过利用性能建模设计优先级算法的方式, 优化任务排序. 如: (1) Tiresias[27]. 该方法假设任务执行时间是已知的, 随后根据任务执行时间设计了服务时间 (service time) 指标, 该指标由任务占用 GPU 数量乘以其剩余执行时间进行计算. 相比于直接使用任务执行时间排序, 服务时间能够同时捕获到执行时间与资源占用量的因素. Tiresias 通过优先调度剩余服务时间短的任务, 能够有效在时空资源需求差异大的任务之间进行权衡. 这种调度策略, 即 SRSF (shortest-remaining-service-first), 成为后续面向深度学习训练调度方法的常见基准方法. (2) Themis[60]针对公平性问题提出. 该方法针对同构、非弹性场景, 因此采取简单的实测剖析法获取执行时间. 随后, Themis 设计了适用于深度学习训练任务的公平性指标: FTF (finish-time-fairness). 该指标使用独占资源时的执行时间除以共享资源时的执行时间 (由建模得出) 计算. 该指标适配放置拓扑敏感、执行时间较长的特性, 使调度器倾向于优先执行那些在较长时间内获得服务较少的任务. 随后 Themis 设计了基于投标拍卖的优先级算法, 以在集群层面达到较高的 FTF 指标. FTF 指标成为后续面向公平性调度方法的常用指标. 相比于 DRF[62]等传统的性能无感公平性调度, 基于 FTF 指标的调度能够有效捕获深度学习训练任务在不同放置方式下的性能差异, 更准确地描绘资源分配的公平程度. 以上方法, 通过利用性能建模设计排序标准进行排序. 虽然它们具有较高的任务排序效率, 然而这种排序过程未考察任务在全局的排序可能性, 因此仅能取得次优排序结果.

Allox[30]通过利用性能建模设计最优图匹配算法, 对任务排序进行优化. 该方法将单 GPU 任务排序问题映射为最小成本二部图匹配问题. 其中, 性能建模得到的任务执行时间被映射为二部图匹配的“边”成本, 因此性能建模成为该调度算法设计的必要前提. 由于仅关注单 GPU 训练任务, 因此实测剖析法产生的开销较低, 适用于该场景. 这类图匹配的方法能够在多项式时间内取得理论最优的任务排序, 然而目前的图匹配算法不能将分布式任务纳入问题的映射, 因此适用场景有限.

Chronus[20]通过性能建模设计了线性规划算法进行优先级求解. 该方法针对截止时间目标进行优化. 该方法面向同构 GPU、任务资源配置固定的场景, 因此可使用实测剖析法以较低开销准确建模任务执行时间. 获得执行时间建模后, Chronus 可据此判断任务的截止时间可满足性, 并可将不可满足的任务延迟调度, 避免队头阻塞. 该方法将最小化截止时间违约率的优先级排序转化为带执行时间约束的混合整数线性规划问题, 利用求解器快速获得最优任务排序. 相比于 EDF[61]等传统的性能无感算法, Chronus 能有效避免截止时间无法满足的任务抢占集群资源. 然而, Chronus 的线性规划建模要求任务支持抢占式调度, 这引入了一定的抢占开销, 对任务完成效率具有一

定影响.

这些对任务排序进行优化的调度方法大多采用实测剖析的方法进行性能建模. 这是由于优化任务排序不涉及对任务的弹性资源分配、放置拓扑等进行优化. 这使得任务可使用的资源配置种类整体不多, 实测开销较低. 因此这些方法对于更为准确、实现便捷的实测剖析法具有倾向性.

3.3 优化异构分配

表 7 展示了利用性能建模来优化异构分配的调度方法. 在面对异构亲和多样化的任务时, 通过对任务异构加速比的感知, 可以避免在亲和性较差的任务与异构资源之间进行匹配尝试, 从而缩减调度决策空间, 有效解决异构挑战. 基于这一思路, 这类方法设计了统一量化算法、线性规划算法、迭代交错算法, 旨在在具有不同异构资源亲和度的任务、不同性能的异构资源之间进行更优化的匹配, 从而提升调度效果. 具体而言, 有如下方法.

表 7 基于性能建模优化异构分配的调度方法

成果	调度目标	性能建模方法	性能建模需求			性能建模利用方式	实验设置		效果
			执行时间	吞吐率	收敛效率		集群规模	任务规模	
Hydra[29]	完成时间 截止时间	实测剖析	√	—	—	设计统一量化算法	45	400	提升截止时间满足率 (比Allox[30]提升86%)
Hare[63]	完成效率	实测剖析	—	√	—	设计迭代交错算法	256	2 000+	降低平均JCT (比Tiresias[27]降低80%)
Gavel[21]	公平性	任务元信息	—	√	—	设计线性规划算法	36	5 000+	提升FTF公平性 (比Themis[60]提升2.8倍)

一些方法为应对异构挑战, 通过利用性能建模设计统一量化的算法, 抹除异构差异性, 从而降低调度难度. 例如 Hydra[29], 该方法针对非弹性场景, 且其面向的异构 GPU 种类较少 (≤3), 因此采取与 Allox[30]、Chronus[20]类似的实测剖析法对性能进行建模. Hydra 面向同时优化截止时间满足与完成时间的目标, 利用建模得到的执行时间设计了成本公式, 可在异构 GPU 上统一量化地表示截止时间的违约量以及任务完成效率, 减弱了异构 GPU 所带来的差异性. 随后, Hydra 设计了多轮最小成本搜索的启发式方法, 在全局内搜索具有最小成本的调度结果. 尽管统一量化方法减弱了异构性带来的挑战, 但目前仅能针对单 GPU 训练, 无法对分布式任务使用的异构通信资源进行统一量化, 适用范围较小.

部分方法通过设计在异构 GPU 上的迭代交错执行算法, 在任务部署后的训练迭代层面设计调度, 减弱资源匹配阶段由异构性所带来的调度难度. 如 Hare[63], 该工作发现, 分布式任务的不同工作节点在异构 GPU 上的吞吐率存在差异, 执行较快的节点需要等待另外的节点计算, 产生低利用率的空档. Hare 针对这一空档, 利用建模得到在异构 GPU 上的任务吞吐率, 设计打包算法以最大化任务组合下的空档利用可能性, 并设法在训练迭代层面设计交错执行策略, 从而进行时分复用, 有效利用空档. 相比于异构性能无感的方法 (shortest-remaing-time-first, SRTF), 大幅降低任务完成时间. 以上这类迭代交错的方法的缺陷在于, 其仅支持通过时分复用提升效率. 然而, 存在大量轻型深度学习训练任务, 可利用空间复用提升资源利用率. 这类方法需考虑如何拓展以支持该场景.

还有方法通过利用性能建模设计线性规划算法解决异构调度问题, 如 Gavel[21]. 该方法面向 GPU 可空间共享的调度场景. 由于不同任务共享 GPU 运行的干扰差异性较大, 任务组合数量众多, 因此使用基于任务元信息的方法对任务吞吐率性能进行建模, 避免大量实测开销. Gavel 是面向异构 GPU 的公平性调度方法. 针对异构挑战, 该方法设计了基于吞吐率矩阵的数据结构, 将待调度任务及其组合在异构 GPU 上的吞吐率统一涵盖表示. 随后, Gavel 利用该矩阵作为输入, 设计了通用性强的线性规划建模方法, 通过更换目标函数可对各个类型的公平性指标进行优化, 包括 max-min 公平性、makespan 最小化, 以及 FTF 公平性. 以上这种面向异构场景的线性规划方法求解复杂度较高, 当待调度任务、异构 GPU 较多时求解缓慢.

以上针对异构分配优化的方法中, 当异构 GPU 种类较少时, 实测剖析的开销是可接受的. 因此, 如 Hydra[29]、Hare[63]等方法仍采用了准确、方便的实测剖析法进行性能建模. 而涉及对大量异构任务组合放置的 Gavel[21]则采

取了能够降低建模开销的任务元信息法.

3.4 优化弹性部署

表 8 展示了利用性能建模来优化弹性部署的调度方法. 这类调度方法主要应对弹性部署配置多样性挑战. 面对多样化的弹性部署配置, 可依靠对不同弹性资源量下的任务性能感知, 挖掘性能变化规律, 并依此精简弹性配置选择空间, 从而有效应对挑战. 基于这一思路, 这类方法设计了贪心算法、图匹配算法, 或构建了强化学习模型, 旨在以最优的方式弹性分配资源, 进而提升全局资源利用率和训练效率.

表 8 基于性能建模优化弹性部署的调度方法

成果	调度目标	性能建模方法	性能建模需求			性能建模利用方式	实验设置		效果
			执行时间	吞吐率	收敛效率		集群规模	任务规模	
ElasticFlow[64]	截止时间	实测剖析	√	—	—	设计贪心算法	128	15 000+	提升截止时间满足率 (比Chronus[20]提升50%)
Pollux[22]	收敛效率	任务元信息	—	—	√	设计贪心算法	64	160	降低模型收敛时间 (比Optimus[33]降低37%)
MuxFlow[65]	完成效率	计算图结构	—	√	—	设计图匹配算法	1 000	7 300+	提升GPU利用率 (比Gandiva[24]提升45%)
MLFS[66]	完成效率 收敛效率	可组合算子	—	√	√	设计强化学习模型	96	2 000+	降低模型收敛时间 (比HyperSched[67]降低64%)

一些方法利用性能建模设计贪心启发式的弹性资源分配规则, 从而避免较高的弹性资源配置搜索空间. 如: (1) ElasticFlow[64]专注于截止时间满足的调度目标. 它提出了一种基于弹性无服务器架构的训练平台, 用户只需提交模型参数与截止时间需求, 而 GPU 数量由平台自动配置. 该方法需要对分布式训练任务的性能在不同 GPU 数量下的性能进行建模, 以判断哪种资源需求能够满足截止时间要求. 由于 ElasticFlow 仅针对长任务调度, 因此采取实测剖析法对执行时间进行建模的开销是可接受的. 该研究发现, 尽管弹性部署配置多样, 但由于通讯开销的存在, 使用单个 GPU 时资源性价比较高. 然而, 仅分配单个 GPU 会导致训练效率低下, 易造成截止时间违约. 因此, ElasticFlow 设计了贪心的弹性调度算法, 为任务分配能够满足其截止时间需求的最少 GPU, 从而以较少资源代价提升截止时间满足率. (2) Pollux[22]利用基于机器学习的任务元信息法对任务在不同参数、资源下的收敛效率进行建模. 它针对任务的收敛效率提出了一个新的指标“goodput”. 该指标能够综合性地衡量每次迭代对于收敛的贡献程度. 随后, Pollux 通过设计分层的联合调度架构来最大化“goodput”. 在任务级别上, Pollux 利用建模得到的收敛率, 设计贪心算法调整任务的 batch size 与学习率, 使其“goodput”最大化. 在集群级别上, 它根据全部任务的“goodput”进行资源重分配, 优先将资源分配给收敛效率低的任务. 以上基于贪心启发式的方法, 能够以较低的调度延迟获取弹性部署结果, 但其普遍不存在理论最优性, 存在优化空间.

MuxFlow[65]将性能模型融入至图匹配算法的设计中, 从而在多项式时间内选取最优弹性配置. 它利用基于 MLP 的计算图结构法对共享 GPU 的任务性能进行建模. 该调度方法能够在底层对不同任务共享 GPU 时使用的 SM 单元进行弹性调控. 根据建模得到的吞吐率信息, 该方法将吞吐率最优的共享 SM 单元分配问题转化为最大权重二部图匹配问题, 从而快速求解出最优弹性部署结果. MuxFlow 的问题在于, 仅适配单 GPU 内多任务的资源弹性部署, 无法面向集群整体的任务打包组合进行全局优化.

还有一些方法将性能建模融入强化学习的模型设计中, 利用强化学习的策略学习能力, 解决弹性调度的挑战. 如 MLFS[66]同时优化完成效率与收敛效率, 使用可组合算子方法对吞吐率、收敛效率进行建模. 该方法将性能模型与深度强化学习模型共同训练, 将 GPU 增加和减少设置为动作, 将建模得到的吞吐率、收敛效率作为回报, 在线上不断根据实时任务进度反馈, 更新策略网络以及性能模型的参数, 从而同时训练出较优的性能模型与调度策略. 基于强化学习模型的方法面临的问题在于其模型的稳定性较差、收敛缓慢. 在模型收敛前, 强化学习模型通常会采取探索-利用权衡的策略, 尝试各类弹性资源分配情况以补充经验回放池, 这对线上任务效率具有较大的负面影响.

以上针对弹性部署进行优化的方法, 大多采取了除实测剖析法以外的低开销方法. 原因在于对弹性部署的支持增加了资源配置种类, 使得开销更低的计算图结构、可组合算子以及任务元信息法更受青睐. 由于这些调度方法的调度算法机制与其使用的性能建模是解耦合的, 因此调度方法可在这 3 类低开销性能建模方法之间进行灵活的选择和替换.

3.5 优化放置拓扑

表 9 展示了利用性能建模优化弹性部署的调度方法. 这类方法主要应对挑战放置拓扑多样性. 面对多样化的放置拓扑选择, 通过对任务的放置拓扑敏感性的深入感知, 可以筛选出高性能的任务组合及其放置拓扑结构, 避免在大量放置拓扑间盲目搜索, 从而解决挑战. 基于这一思路, 这类方法设计了试错迁移算法、规划算法以及贪心算法, 旨在为任务规划合理的放置拓扑, 进而减少因通信延迟和任务间干扰导致的性能瓶颈.

表 9 基于性能建模优化放置拓扑的调度方法

成果	调度目标	性能建模方法	性能建模需求			性能建模利用方式	实验设置		效果
			执行时间	吞吐率	收敛效率		集群规模	任务规模	
Gandiva[24]	完成效率	实测剖析	—	√	—	设计试错迁移算法	180	2 000+	提升GPU利用率 (比Apache Yarn[59]提升26%)
SMD[68]	完成效率	可组合算子	√	√	—	设计规划算法	24	200	提升GPU利用率 (比Optimus[33]提升30%)
CoGNN[69]	完成效率	可组合算子	—	√	—	设计贪心算法	8	200+	降低平均JCT (比MPS[70], MIG[71]降低75%)
PowerFlow[72]	完成效率 成本效率	计算图结构	—	√	—	设计贪心算法	8/16/24	1 901	降低集群能耗 (比Tiresias[27]降低1.57倍)

Gandiva[24]通过利用性能建模设计试错、迁移的算法, 对任务放置拓扑进行优化. 该工作是早期经典的深度学习训练调度工作, 首次针对深度学习训练任务的放置拓扑进行特殊优化. 虽然在首次调度前该方法对不同放置拓扑的任务性能无感知, 但它通过在线实测分析当前放置拓扑下任务的通信、性能干扰情况, 基于经验在线地建立性能模型, 并设计 grow-shrink 机制对任务进行不停机在线迁移, 使得新的放置拓扑具有更低的通信延迟与性能干扰, 从而通过试错、反馈、调优的机制提高集群整体利用率与效率. 该方法的缺陷在于, 这种试错、迁移的过程对于每个新来任务是必需的, 因此会使任务前期的效率较差.

一些方法利用性能建模, 设计规划算法对最优放置拓扑的问题进行求解, 如 SMD[68]. 该方法使用基于可组合算子的方法进行性能建模, 并将其与调度算法进行深度结合. 它针对 PS 架构分布式训练任务进行调度, 并设计分析模型对基于 ASP 与 BSP 的 PS 架构算子级别通信延时进行精细化建模. 利用该建模, SMD 能够从算子层面预测各种放置拓扑下的任务间通信干扰, 并将最小化干扰的调度问题建模为一个带有装箱约束的非凸整数非线性规划问题. 随后, 开发了一种名为 sum-of-ratio multi-dimensional-knapsack decomposition 的 ϵ 近似算法来快速求解该问题, 并通过理论和实验验证了算法的有效性. 但是该方法将其规划模型设计与 PS 分布式训练架构的性能模型深度耦合, 需要考虑如何扩展方法, 以适配其他分布式训练架构, 如 ring-allreduce.

还有部分方法利用性能建模设计贪心的任务放置规则, 通过降低任务干扰或提升通信效率, 增强调度效果, 如: (1) CoGNN[69]针对打包并发任务设计贪心算法, 通过降低任务放置干扰, 提升效率. 该方法专注于图神经网络 (GNN) 的并发训练, 使用基于可组合算子的方法进行性能建模. 该方法观察到 GNN 中算子的 GPU 利用率低, 存在通过并发放置提升效率的机会. 因此, 它基于分析模型建立了面向 GNN 算子的内存冲突模型及其性能干扰模型, 并据此设计了 BMC (balanced-memory-consumption) 贪心打包调度策略, 以平衡打包后任务的并发显存访问延时, 从而最小化并发 GNN 放置产生的性能干扰. 相比于 MPS、MIG 等性能无感的并发训练机制, 大幅降低任务完成时间. (2) PowerFlow[72]基于计算图结构法建模性能, 针对集群能耗问题进行优化. 首先, 该方法从底层对训练不同阶段 (前向计算、反向传播、参数更新) 进行分析, 并分别设计回归模型建立不同通信速率下吞吐率与能耗的关系模型. 随后, 基于该模型以及不同放置拓扑下的通信能力分析, 设计了启发式贪心算法, 为任务分配具有最优

能耗性价比的 GPU 资源量以及放置拓扑, 并与网络打包/伙伴分配[73]的集中放置策略相结合, 进一步优化全局任务拓扑, 避免因集群碎片造成的额外能耗开销.

与弹性部署优化情况类似, 支持优化放置拓扑的调度可在较多的资源配置种类之间选择, 因此上述方法大多采取低开销性能建模方法. Gandiva[24]采取了高开销的实测剖析法, 则因此造成了任务执行初期的低效情况. 在上述方法中, SMD[68]较为特别, 原因在于它的调度算法与它设计的基于可组合算子的性能建模方法深度耦合, 这使得在扩展此方法的适用场景时面临一定的挑战.

3.6 对比与小结

表 10 展示了对本节综述的调度方法在多个维度上的对比. 这些调度方法利用性能建模, 分别针对任务排序、异构分配、弹性部署、放置拓扑进行优化, 依次对任务多样性挑战、异构亲和度多样性挑战、弹性部署配置多样性挑战和放置拓扑多样性挑战进行解决, 进而从不同维度优化调度效果.

表 10 基于性能建模的调度方法对比

利用性能建模的调度优化策略	应对挑战	性能建模方法倾向性	性能建模利用方式 (调度算法设计依据)				
			优先级算法	规划算法	图匹配算法	贪心算法	其他算法
优化任务排序	任务多样性挑战	实测剖析法	√	√	√	—	—
优化异构分配	异构亲和度多样性挑战	实测剖析法 (少量异构) 任务元信息法 (大量异构)	—	√	—	—	统一量化/交错迭代
优化弹性部署	弹性部署配置多样性挑战	低开销方法	—	—	√	√	强化学习
优化放置拓扑	放置拓扑多样性挑战	低开销方法	—	√	—	√	迁移试错

在性能建模方法的选择上, 当调度涉及的资源配置种类相对较少时, 建模相对准确、设计和实现较为便捷、且更贴近实际执行环境的实测剖析法往往更受青睐. 而在需要优化弹性部署或放置拓扑的场景中, 低开销建模方法则因其高效性而更具优势.

在性能建模信息的利用方式 (即调度算法设计依据) 方面, 规划模型因其问题映射难度相对较低、求解理论丰富完备以及拥有众多现成的求解器和算法而备受推崇, 多种方法通过精心设计规划模型来充分利用性能建模信息. 相比之下, 图匹配算法虽然在快速求解方面也有一定优势, 但由于问题映射上的难度较高, 因此仅被少数方法采用. 贪心算法虽然在理论上不能保证得到最优解, 但在处理大规模调度问题时却能展现出快速求解的能力, 因此也受到了一些研究的关注. 优先级算法则主要应用于针对任务排序的方法中, 专注于设定合理的优先级来优化任务排序. 此外, 还有一些其他的性能建模利用方式, 如统一量化、强化学习等. 这些方法的设计思路与其对应的调度优化策略紧密相关, 对特定策略的优化效果十分显著, 但难以在不同策略间直接进行迁移和适配.

4 展 望

第 2、3 节对深度学习训练的性能建模及调度进行了全面概述. 然而, 现有方法在新型应用场景上仍存在一定局限性. 如在对延时敏感的场景下, 现有实测剖析法的显著开销使其难以有效应用. 在为最近涌现的新型任务 (如大模型任务、推理训练混合任务) 进行性能建模与调度时, 其特殊的任务性能特性将导致现有方法难以适配. 此外, 现有方法在用户隐私方面不具备保障. 针对这些局限性, 本节将展望可能产生突破的研究方法与方向, 以期为未来深度学习训练任务的性能建模与调度提供更加有效和灵活的解决方案.

● 面向低延时需求的性能建模方法. 随着人工智能的发展, 面向广大用户群体的廉价云 GPU 租赁服务大量涌现. 此类集群中, 由学生和研究者提交的短期训练任务 (5–10 min) 居多. 这些任务主要用于调测、测试, 对任务调度的延时和响应时间有较高的要求. 然而, 现有基于实测剖析的性能建模方法由于涉及资源分配和任务实测过程, 开销较高, 难以实现低延时调度. 为降低开销, 未来研究可考虑与任务快速启停迁移技术 (如 Singularity[74]、Non-Intrusive[75]) 结合, 通过降低任务启停开销至毫秒级别, 降低资源分配开销. 同时, 结合 GPU 虚拟化技术 (如

KubeShare[76]、GaiaGPU[77]、TGS[78]、cGPU[79]) 也为可行方向, 以避免任务多次迁移. 此外, 还可对深度学习训练和 GPU 的底层运行机制的深入探索, 设计无需实测的仿真运行器, 从根本上消除建模开销.

● 面向新型任务的性能建模与调度. 随着各类新型深度学习模型及其资源利用方式的涌现, 具有新型性能特征的任务为现有性能建模与调度方法带来了新的挑战. 如大模型的出现使得支持流水线并行的调度成为必需; 而推理训练混合的新型任务部署形式, 使调度需满足高响应、高吞吐的混合需求. 为扩展现有方法以支撑新型任务, 具体而言, 有如下分析: (1) 大模型任务. 随着基于 Transformer[80], BERT[28]的大型预训练模型以及 ChatGPT[81]、文心一言[82]等大语言模型的出现, 单 GPU 显存已无法满足需求. 将模型切分为异构子任务, 并采取流水线并行训练的方式已成为趋势. 然而, 现有方法仅支持同构子任务, 对计算和通信量具有差异的异构子任务难以适配. 在性能建模方面, 现有方法需要提升灵活性, 支撑对各种切分方式下的异构子任务性能进行建模. 调度方面, 需探索大模型的合理切分点, 并结合考虑流水线式的通信需求, 以避免因通信干扰导致的流水线气泡. (2) 推理训练混合任务. 近期工作[6]指出深度学习推理与训练任务混合部署的方案更加经济. 与训练时关注完成时间和吞吐率不同, 推理更关注请求延迟、SLA (服务等级协议) 等指标. 因此, 在性能建模方面, 需探索如何预测不同用户负载下的推理延时以及训练吞吐率. 在调度方面需考虑如何满足推理动态与训练静态的资源需求. 现有方法可考虑结合 GPU 弹性资源分配机制, 探索混合部署下 GPU 配额的动态分配算法, 避免资源抢占造成的推理延迟上升.

● 关注隐私保障的性能建模与调度. 现有方法假设用户源代码与训练数据可获取. 然而, 用户的隐私需求已成为如今人工智能服务提供商不可忽略的问题. 这为现有方法带来挑战. 下面在数据中心场景与云边端协同场景, 分别分析以上挑战: (1) 源代码隐私保障 (数据中心场景). 在为向数据中心提交的任务进行性能建模时, 现有方法需获取用户源代码, 以提取建模所需的特征. 为在保障隐私的前提下有效应用这些建模方法, 需要探索如何在避免接触代码的情况下提取必要特征. CUPTI、Nsight 等工具支持剖析算子的执行跟踪. 因此, 现有方法可探索如何从执行跟踪进行分析, 从而还原所需的算子特征. (2) 数据隐私保障 (云边端联邦学习场景). 云边端协同的联邦学习为保护用户数据隐私提供了一种解决方案. 该场景中, 用户在本地执行训练, 而云端仅负责模型整合. 因此, 性能建模与调度需要在无法实际接触用户数据和设备的情况下进行. 性能建模方面可以考虑扩展现有任务元信息方法, 将用户侧非隐私信息融入任务元信息, 利用海量数据驱动来提升建模准确度. 在调度方面, 需探索可试错、可自适应的调度方法, 避免因用户数据未知造成的用户间计算量不均衡, 从而提升训练效率.

5 总 结

本文全面概述了基于性能建模的深度学习训练任务调度的研究现状. 首先, 系统介绍了面向深度学习训练任务的性能建模方法, 并对现有工作进行了分类整理, 涵盖了基于实测剖析、任务元信息、可组合算子以及计算图结构的多种方法. 其次, 根据利用性能建模优化调度效果的策略, 对现有调度工作进行了分类整理, 将其划分为优化任务排序、异构分配、弹性部署以及放置拓扑这 4 大类别, 深入阐述了性能建模在任务调度中的核心意义与作用. 最后, 对基于性能建模的任务调度未来研究方向进行了展望. 通过本文的综述, 读者将对面向深度学习训练的任务性能建模与调度有更全面的了解, 为相关研究和应用提供参考和借鉴, 也为未来本领域的研究方向提供了有益的展望和启示.

References:

[1] Litjens G, Kooi T, Bejnordi BE, Setio AAA, Ciompi F, Ghafoorian M, van der laak JAWM, van Ginneken B, Sánchez CI. A survey on deep learning in medical image analysis. Medical Image Analysis, 2017, 42: 60–88. [doi: 10.1016/j.media.2017.07.005]

[2] Parekh D, Poddar N, Rajpurkar A, Chahal M, Kumar N, Joshi GP, Cho W. A review on autonomous vehicles: Progress, methods and challenges. Electronics, 2022, 11(14): 2162. [doi: 10.3390/electronics11142162]

[3] Kortli Y, Jridi M, Al Falou A, Atri M. Face recognition systems: A survey. Sensors, 2020, 20(2): 342. [doi: 10.3390/s20020342]

[4] Liu YC, Zong CQ. End-to-end speech translation by integrating cross-modal information. Ruan Jian Xue Bao/Journal of Software, 2023, 34(4): 1837–1849 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6413.htm [doi: 10.13328/j.cnki.jos.006413]

[5] Khurana D, Koli A, Khatter K, Singh S. Natural language processing: State of the art, current trends and challenges. Multimedia Tools

and Applications, 2023, 82(3): 3713–3744. [doi: 10.1007/s11042-022-13428-4]

[6] Weng QZ, Xiao WC, Yu YH, Wang W, Wang C, He J, Li Y, Zhang LP, Lin W, Ding Y. MLaaS in the wild: Workload analysis and scheduling in large-scale heterogeneous GPU clusters. In: Proc. of the 19th USENIX Symp. on Networked Systems Design and Implementation. Renton: USENIX, 2022. 945–960.

[7] Jeon M, Venkataraman S, Phanishayee A, Qian JJ, Xiao WC, Yang F. Analysis of large-scale multi-tenant GPU clusters for DNN training workloads. In: Proc. of the 2019 USENIX Annual Technical Conf. Renton: USENIX, 2019. 947–960.

[8] Rico-Gallego JA, Díaz-Martín JC, Manumachu RR, Lastovetsky AL. A survey of communication performance models for high-performance computing. ACM Computing Surveys, 2019, 51(6): 126. [doi: 10.1145/3284358]

[9] Reuther A, Byun C, Arcand W, Bestor D, Bergeron B, Hubbell M, Jones M, Michaleas P, Prout A, Rosa A, Kepner J. Scalable system scheduling for HPC and big data. Journal of Parallel and Distributed Computing, 2018, 111: 76–92. [doi: 10.1016/j.jpdc.2017.06.009]

[10] Netto MAS, Calheiros RN, Rodrigues ER, Cunha RLF, Buyya R. HPC cloud for scientific and business applications: Taxonomy, vision, and research challenges. ACM Computing Surveys, 2019, 51(1): 8. [doi: 10.1145/3150224]

[11] Song J, Sun ZZ, Mao KM, Bao YB, Yu G. Research advance on mapreduce based big data processing platforms and algorithms. Ruan Jian Xue Bao/Journal of Software, 2017, 28(3): 514–543 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5169.htm [doi: 10.13328/j.cnki.jos.005169]

[12] Yu FX, Wang D, Shangguan LF, Zhang MJ, Tang XL, Liu CC, Chen X. A survey of large-scale deep learning serving system optimization: Challenges and opportunities. arXiv:2111.14247, 2021.

[13] Yu FX, Wang D, Shangguan LF, Zhang MJ, Liu CC, Chen X. A survey of multi-tenant deep learning inference on GPU. arXiv:2203.09040, 2022.

[14] Ren J, Gao L, Yu JL, Yuan L. Energy-efficient deep learning task scheduling strategy for edge device. Chinese Journal of Computers, 2020, 43(3): 440–452 (in Chinese with English abstract). [doi: 10.11897/SP.J.1016.2020.00440]

[15] Mittal S, Vaishay S. A survey of techniques for optimizing deep learning on GPUs. Journal of Systems Architecture, 2019, 99: 101635. [doi: 10.1016/j.sysarc.2019.101635]

[16] Rasley J, Rajbhandari S, Ruwase O, He YX. DeepSpeed: System optimizations enable training deep learning models with over 100 billion parameters. In: Proc. of the 26th ACM SIGKDD Int'l Conf. on Knowledge Discovery & Data Mining. ACM, 2020. 3505–3506. [doi: 10.1145/3394486.3406703]

[17] Gao HR, Wu H, Xu YJ, Li XH, Wang T, Zhang WB. Survey on memory swapping mechanism for deep learning training. Ruan Jian Xue Bao/Journal of Software, 2023, 34(12): 5862–5886 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6800.htm [doi: 10.13328/j.cnki.jos.006800]

[18] Gao W, Hu QH, Ye ZS, Sun P, Wang XL, Luo YW, Zhang TW, Wen YG. Deep learning workload scheduling in GPU datacenters: Taxonomy, challenges and vision. arXiv:2205.11913, 2022.

[19] Mayer R, Jacobsen HA. Scalable deep learning on distributed infrastructures: Challenges, techniques, and tools. ACM Computing Surveys, 2021, 53(1): 3. [doi: 10.1145/3363554]

[20] Gao W, Ye ZS, Sun P, Wen YG, Zhang TW. Chronus: A novel deadline-aware scheduler for deep learning training jobs. In: Proc. of the 2021 ACM Symp. on Cloud Computing. Seattle: ACM, 2021. 609–623. [doi: 10.1145/3472883.3486978]

[21] Narayanan D, Santhanam K, Kazhamiaka F, Phanishayee A, Zaharia M. Heterogeneity-aware cluster scheduling policies for deep learning workloads. In: Proc. of the 14th USENIX Symp. on Operating Systems Design and Implementation. USENIX, 2020. 481–498.

[22] Qiao A, Choe SK, Subramanya SJ, Neiswanger W, Ho Q, Zhang H, Ganger GR, Xing EP. Pollux: Co-adaptive cluster scheduling for goodput-optimized deep learning. In: Proc. of the 15th USENIX Symp. on Operating Systems Design and Implementation. USENIX, 2021.

[23] He KM, Zhang XY, Ren SQ, Sun J. Deep residual learning for image recognition. In: Proc. of the 2016 IEEE Conf. on Computer Vision and Pattern Recognition. Las Vegas: IEEE, 2016. 770–778. [doi: 10.1109/CVPR.2016.90]

[24] Xiao WC, Bhardwaj R, Ramjee R, Sivathanu M, Kwatra N, Han ZH, Patel P, Peng X, Zhao HY, Zhang QL, Yang F, Zhou LD. Gandiva: Introspective cluster scheduling for deep learning. In: Proc. of the 13th USENIX Symp. on Operating Systems Design and Implementation. Carlsbad: USENIX, 2018. 595–610.

[25] Han ZH, Tan HS, Jiang SHC, Fu XM, Cao WL, Lau FCM. Scheduling placement-sensitive BSP jobs with inaccurate execution time estimation. In: Proc. of the 2020 IEEE Conf. on Computer Communications. Toronto: IEEE, 2020. 1053–1062. [doi: 10.1109/INFOCOM41043.2020.9155445]

[26] Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition. In: Proc. of the 3rd Int'l Conf. on Learning Representations. San Diego: ICLR, 2015. [doi: 10.48550/arXiv.1409.1556]

[27] Gu JC, Chowdhury M, Shin KG, Zhu YB, Jeon M, Qian JJ, Liu HH, Guo CX. Tiresias: A GPU cluster manager for distributed deep

learning. In: Proc. of the 16th USENIX Symp. on Networked Systems Design and Implementation. Boston: USENIX, 2019. 485–500.

[28] Devlin J, Chang MW, Lee K, Toutanova K. BERT: Pre-training of deep bidirectionals for language understanding. In: Proc. of the 2019 Conf. of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. Minneapolis: ACL, 2018. 4171–4186. [doi: 10.18653/v1/N19-1423]

[29] Yang ZC, Wu H, Xu YJ, Wu YW, Zhong H, Zhang WB. Hydra: Deadline-aware and efficiency-oriented scheduling for deep learning jobs on heterogeneous GPUs. IEEE Trans. on Computers, 2023, 72(8): 2224–2236. [doi: 10.1109/TC.2023.3242200]

[30] Le TN, Sun X, Chowdhury M, Liu ZH. AlloX: Compute allocation in hybrid clusters. In: Proc. of the 15th European Conf. on Computer Systems. Heraklion: ACM, 2020. 31. [doi: 10.1145/3342195.3387547]

[31] Zheng HY, Xu F, Chen L, Zhou Z, Liu FM. Cynthia: Cost-efficient cloud resource provisioning for predictable distributed deep neural network training. In: Proc. of the 48th Int'l Conf. on Parallel Processing. Kyoto: ACM, 2019. 86. [doi: 10.1145/3337821.3337873]

[32] Mohan J, Phanishayee A, Kulkarni J, Chidambaram V. Looking beyond GPUs for DNN scheduling on multi-tenant clusters. In: Proc. of the 16th USENIX Symp. on Operating Systems Design and Implementation. Carlsbad: USENIX, 2022. 579–596.

[33] Peng YH, Bao YX, Chen YR, Wu C, Guo CX. Optimus: An efficient dynamic resource scheduler for deep learning clusters. In: Proc. of the 13th EuroSys Conf. Porto: ACM, 2018. 3. [doi: 10.1145/3190508.3190517]

[34] Szegedy C, Vanhoucke V, Ioffe S, Shlens J, Wojna Z. Rethinking the inception architecture for computer vision. In: Proc. of the 2016 IEEE Conf. on Computer Vision and Pattern Recognition. Las Vegas: IEEE, 2016. 2818–2826. [doi: 10.1109/CVPR.2016.308]

[35] Sutskever I, Vinyals O, Le QV. Sequence to sequence learning with neural networks. In: Proc. of the 27th Int'l Conf. on Neural Information Processing Systems. Montreal: MIT Press, 2014. 3104–3112.

[36] Zheng PF, Pan R, Khan T, Venkataraman S, Akella A. Shockwave: Fair and efficient cluster scheduling for dynamic adaptation in machine learning. In: Proc. of the 20th USENIX Symp. on Networked Systems Design and Implementation. Boston: USENIX, 2023. 703–723.

[37] Agarwal S, Wang HY, Lee K, Venkataraman S, Papailiopoulos D. Adaptive gradient communication via critical learning regime identification. In: Proc. of the 4th Machine Learning and Systems. MLSys, 2021. 55–80.

[38] Qin HY, Rajbhandari S, Ruwase O, Yan F, Yang L, He YX. SimiGrad: Fine-grained adaptive batching for large scale training using gradient similarity measurement. In: Proc. of the 34th Int'l Conf. on Neural Information Processing Systems. NeurIPS, 2021. 20531–20544.

[39] Zhu HY, Phanishayee A, Pekhimenko G. Daydream: Accurately estimating the efficacy of optimizations for DNN training. In: Proc. of the 2020 USENIX Annual Technical Conf. USENIX, 2020. 337–352.

[40] Lam MO, Hollingsworth JK, De Supinski BR, Legendre MP. Automatically adapting programs for mixed-precision floating-point computation. In: Proc. of the 27th Int'l ACM Conf. on Int'l Conf. on Supercomputing. Eugene: ACM, 2013. 369–378. [doi: 10.1145/2464996.2465018]

[41] Niu W, Guan JX, Wang YZ, Agrawal G, Ren B. DNNFusion: Accelerating deep neural networks execution with advanced operator fusion. In: Proc. of the 42nd ACM SIGPLAN Int'l Conf. on Programming Language Design and Implementation. ACM, 2021. 883–898. [doi: 10.1145/3453483.3454083]

[42] Duan JF, Li XH, Xu P, Zhang XC, Yan SG, Liang Y, Lin DH. Proteus: Simulating the performance of distributed DNN training. arXiv:2306.02267, 2023.

[43] Hu QH, Sun P, Yan SG, Wen YG, Zhang TW. Characterization and prediction of deep learning workloads in large-scale GPU datacenters. In: Proc. of the 2021 Int'l Conf. for High Performance Computing, Networking, Storage and Analysis. St. Louis: ACM, 2021. 104. [doi: 10.1145/3458817.3476223]

[44] Bao YX, Peng YH, Wu C. Deep learning-based job placement in distributed machine learning clusters. In: Proc. of the 2019 IEEE Conf. on Computer Communications. Paris: IEEE, 2019. 505–513. [doi: 10.1109/INFOCOM.2019.8737460]

[45] Graves A, Fernández S, Gomez F, Schmidhuber J. Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks. In: Proc. of the 23rd Int'l Conf. on Machine Learning. Pittsburgh: ACM, 2006. 369–376. [doi: 10.1145/1143844.1143891]

[46] Chen ZY, Quan W, Wen M, Fang JB, Yu J, Zhang CY, Luo L. Deep learning research and development platform: Characterizing and scheduling with QoS guarantees on GPU clusters. IEEE Trans. on Parallel and Distributed Systems, 2020, 31(1): 34–50. [doi: 10.1109/TPDS.2019.2931558]

[47] Steinberg D, Colla P. CART: Classification and regression trees. The Top Ten Algorithms in Data Mining, 2009, 9: 179.

[48] Yeung G, Borowiec D, Yang RY, Friday A, Harper R, Garraghan P. Horus: Interference-aware and prediction-based scheduling in deep learning systems. IEEE Trans. on Parallel and Distributed Systems, 2022, 33(1): 88–100. [doi: 10.1109/TPDS.2021.3079202]

[49] Yeung G, Borowiec D, Friday A, Harper R, Garraghan P. Towards GPU utilization prediction for cloud deep learning. In: Proc. of the 12th USENIX Workshop on Hot Topics in Cloud Computing. USENIX, 2020.

[50] Bai L, Ji WX, Li QY, Yao XL, Xin W, Zhu WY. DNNAbacus: Toward accurate computational cost prediction for deep neural networks. arXiv:2205.12095, 2022.

[51] He X, Zhao KY, Chu XW. AutoML: A survey of the state-of-the-art. Knowledge-based Systems, 2021, 212: 106622. [doi: 10.1016/j.knosys.2020.106622]

[52] Gao YJ, Gu XY, Zhang HY, Lin HX, Yang M. Runtime performance prediction for deep learning models with graph neural network. In: Proc. of the 45th IEEE/ACM Int'l Conf. on Software Engineering: Software Engineering in Practice. Melbourne: IEEE, 2023. 368–380. [doi: 10.1109/ICSE-SEIP58684.2023.00039]

[53] Yang G, Shin C, Lee J, Yoo Y, Yoo C. Prediction of the resource consumption of distributed deep learning systems. Proc. of the ACM on Measurement and Analysis of Computing Systems, 2022, 6(2): 29. [doi: 10.1145/3530895]

[54] Yu GX, Gao YB, Golikov P, Pekhimenko G. Habitat: A runtime-based computational performance predictor for deep neural network training. In: Proc. of the 2021 USENIX Annual Technical Conf. USENIX, 2021. 503–521.

[55] Radford A, Metz L, Chintala S. Unsupervised representation learning with deep convolutional generative adversarial networks. In: Proc. of the 4th Int'l Conf. on Learning Representations. San Juan: ICLR, 2016. [doi: 10.48550/arXiv.1511.06434]

[56] Liu GD, Wang S, Bao YG. SEER: A time prediction model for CNNs from GPU kernel's view. In: Proc. of the 30th Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT). Atlanta: IEEE, 2021. 173–185. [doi: 10.1109/PACT52795.2021.00020]

[57] Wang CC, Liao YC, Kao MC, Liang WY, Hung SH. PerfNet: Platform-aware performance modeling for deep neural networks. In: Proc. of the 2020 Int'l Conf. on Research in Adaptive and Convergent Systems. Gwangju: ACM, 2020. 90–95. [doi: 10.1145/3400286.3418245]

[58] Wang CC, Liao YC, Kao MC, Liang WY, Hung SH. Toward accurate platform-aware performance modeling for deep neural networks. ACM SIGAPP Applied Computing Review, 2021, 21(1): 50–61. [doi: 10.1145/3477133.3477137]

[59] Vavilapalli VK, Murthy AC, Douglas C, Agarwal S, Konar M, Evans R, Graves T, Lowe J, Shah H, Seth S, Saha B, Curino C, O'Malley O, Radia S, Reed B, Baldeschwieler E. Apache hadoop YARN: Yet another resource negotiator. In: Proc. of the 4th Annual Symp. on Cloud Computing. Santa Clara: ACM, 2013. 5. [doi: 10.1145/2523616.2523633]

[60] Mahajan K, Balasubramanian A, Singhvi A, Venkataraman S, Akella A, Phanishayee A, Chawla S. THEMIS: Fair and efficient GPU cluster scheduling. In: Proc. of the 17th USENIX Conf. on Networked Systems Design and Implementation. Santa Clara: USENIX, 2020. 289–304.

[61] Kargahi M, Movaghar A. A method for performance analysis of earliest-deadline-first scheduling policy. The Journal of Supercomputing, 2006, 37(2): 197–222. [doi: 10.1007/s11227-006-5944-2]

[62] Ghodsi A, Zaharia M, Hindman B, Konwinski A, Shenker S, Stoica I. Dominant resource fairness: Fair allocation of multiple resource types. In: Proc. of the 8th USENIX Symp. on Networked Systems Design and Implementation. Boston: USENIX, 2011.

[63] Chen FH, Li P, Wu C, Guo S. Hare: Exploiting inter-job and intra-job parallelism of distributed machine learning on heterogeneous GPUs. In: Proc. of the 31st Int'l Symp. on High-performance Parallel and Distributed Computing. Minneapolis: ACM, 2022. 253–264. ACM: Minneapolis MN USA. [doi: 10.1145/3502181.3531462]

[64] Gu DD, Zhao YH, Zhong YM, Xiong YF, Han ZH, Cheng P, Yang F, Huang G, Jin X, Liu XZ. ElasticFlow: An elastic serverless training platform for distributed deep learning. In: Proc. of the 28th ACM Int'l Conf. on Architectural Support for Programming Languages and Operating Systems. Vancouver: ACM, 2023. 266–280. [doi: 10.1145/3575693.3575721]

[65] Zhao YH, Liu X, Liu SF, Li X, Zhu YB, Huang G, Liu XZ, Jin X. MuxFlow: Efficient and safe GPU sharing in large-scale production deep learning clusters. arXiv:2303.13803, 2023.

[66] Wang HY, Liu ZT, Shen HY. Job scheduling for large-scale machine learning clusters. In: Proc. of the 16th Int'l Conf. on Emerging Networking Experiments and Technologies. Barcelona: ACM, 2020. 108–120. [doi: 10.1145/3386367.3432588]

[67] Liaw R, Bhardwaj R, Dunlap L, Zou YT, Gonzalez JE, Stoica I, Tumanov A. HyperSched: Dynamic resource reallocation for model development on a deadline. In: Proc. of the 2019 ACM Symp. on Cloud Computing. Santa Cruz: ACM, 2019. 61–73. [doi: 10.1145/3357223.3362719]

[68] Yu ML, Wu C, Ji B, Liu J. A sum-of-ratios multi-dimensional-knapsack decomposition for DNN resource scheduling. In: Proc. of the 2021 IEEE Conf. on Computer Communications. Vancouver: IEEE, 2021. 1–10. [doi: 10.1109/INFOCOM42981.2021.9488916]

[69] Sun QX, Liu Y, Yang HL, Zhang RZ, Dun M, Li MZ, Liu XY, Xiao WC, Li Y, Luan ZZ, Qian DP. CoGNN: Efficient scheduling for concurrent GNN training on GPUs. In: Proc. of the 2022 Int'l Conf. for High Performance Computing, Networking, Storage and Analysis. Dallas: IEEE, 2022. 1–15. [doi: 10.1109/SC41404.2022.00044]

[70] NVIDIA. Multi-process service. 2023. https://docs.nvidia.com/deploy/mps/index.html

[71] NVIDIA. NVIDIA multi-instance GPU user guide. 2023. https://docs.nvidia.com/datacenter/tesla/mig-user-guide/index.html

[72] Gu DD, Xie XT, Huang G, Jin X, Liu XZ. Energy-efficient GPU clusters scheduling for deep learning. arXiv:2304.06381, 2023.

[73] Zhao HY, Han ZH, Yang Z, Zhang QL, Yang F, Zhou LD, Yang M, Lau FCM, Wang YQ, Xiong YF, Wang B. HiveD: Sharing a GPU cluster for deep learning with guarantees. In: Proc. of the 14th USENIX Symp. on Operating Systems Design and Implementation. USENIX, 2020. 515–532.

[74] Shukla D, Sivathanu M, Viswanatha S, *et al*. Singularity: Planet-scale, preemptive and elastic scheduling of AI workloads. arXiv:2202.07848, 2022.

[75] Wang SQ, Gonzalez OJ, Zhou XB, Williams T, Friedman BD, Havemann M, Woo T. An efficient and non-intrusive GPU scheduling framework for deep learning training systems. In: Proc. of the 2020 Int'l Conf. for High Performance Computing, Networking, Storage and Analysis. Atlanta: IEEE, 2020. 1–3. [doi: 10.1109/SC41405.2020.00094]

[76] Yeh TA, Chen HH, Chou J. KubeShare: A framework to manage GPUs as first-class and shared resources in container cloud. In: Proc. of the 29th Int'l Symp. on High-performance Parallel and Distributed Computing. Stockholm: ACM, 2020. 173–184. [doi: 10.1145/3369583.3392679]

[77] Gu J, Song SB, Li Y, Luo HM. GaiaGPU: Sharing GPUs in container clouds. In: Proc. of the 2018 IEEE Int'l Conf. on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications (ISPA/IUCC/BDCloud/SocialCom/SustainCom). Melbourne: IEEE, 2018. 469–476. [doi: 10.1109/BDCloud.2018.00077]

[78] Wu BY, Zhang ZL, Bai ZH, Liu XZ, Jin X. Transparent GPU sharing in container clouds for deep learning workloads. In: Proc. of the 20th USENIX Symp. on Networked Systems Design and Implementation. Boston: USENIX, 2023. 69–85.

[79] ALIBABA. Alibaba cloud elastic GPU service best practice. 2023. https://static-aliyun-doc.oss-cn-hangzhou.aliyuncs.com/download%2Fpdf%2F163835%2FBest_Practices_reseller_en-US.pdf

[80] Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, Kaiser Ł, Polosukhin I. Attention is all you need. In: Proc. of the 31st Int'l Conf. on Neural Information Processing Systems. Long Beach: Curran Associates Inc., 2017. 6000–6010.

[81] OpenAI. GPT-4 technical report. arXiv:2303.08774, 2023.

[82] Baidu. ERNIE bot. 2023. https://yiyan.baidu.com/

附中文参考文献:

[4] 刘宇宸, 宗成庆. 跨模态信息融合的端到端语音翻译. 软件学报, 2023, 34(4): 1837–1849. http://www.jos.org.cn/1000-9825/6413.htm [doi: 10.13328/j.cnki.jos.006413]

[11] 宋杰, 孙宗哲, 毛克明, 鲍玉斌, 于戈. MapReduce 大数据处理平台与算法研究进展. 软件学报, 2017, 28(3): 514–543. http://www.jos.org.cn/1000-9825/5169.htm [doi: 10.13328/j.cnki.jos.005169]

[14] 任杰, 高岭, 于佳龙, 袁璐. 面向边缘设备的高能效深度学习任务调度策略. 计算机学报, 2020, 43(3): 440–452. [doi: 10.11897/SP.J.1016.2020.00440]

[17] 高赫然, 吴恒, 许源佳, 李修和, 王焘, 张文博. 面向深度学习训练的内存交换机制综述. 软件学报, 2023, 34(12): 5862–5886. http://www.jos.org.cn/1000-9825/6800.htm [doi: 10.13328/j.cnki.jos.006800]

[82] 百度. 文心一言. 2023. https://yiyan.baidu.com/

杨紫超(1999－), 男, 博士生, 主要研究领域为资源调度, 分布式系统.

吴悦文(1990－), 男, 博士, CCF 专业会员, 主要研究领域为云计算, 容量规划.

吴恒(1983－), 男, 博士, 副研究员, 主要研究领域为容器虚拟化, 边缘计算.

张文博(1976－), 男, 博士, 研究员, 博士生导师, 主要研究领域为云计算, 服务计算.