浙江大學
ZHEJIANG UNIVERSITY

《云原生技术及应用》
针对深度学习负载的 GPU 集群调度系统
文献综述

姓 名 刘京宗

学 号 22451040

学 院 软件学院

实验日期 2024 年 10 月 29 日

授课教师 才振功

针对深度学习负载的 GPU 集群调度系统文献综述

刘京宗 22451040

摘 要 近年来，随着深度学习在多个领域的应用不断扩大，对高性能计算资源的需求日益增加，GPU 集群成为支撑深度学习任务的主要平台。然而，多租户环境中 GPU 资源的管理效率成为瓶颈。本文系统性地综述了云原生环境下 GPU 集群调度的最新研究进展，包括 Pollux、DeepBoot、Aryl 等系统在调度策略上的创新。本文分析了各系统在 Goodput 优化、弹性资源共享、联合调度、检查点与任务恢复等方面的核心技术，探讨了这些技术在提升集群资源利用率和任务完成效率方面的作用。最后，本文还提出了未来研究方向，包括任务切换开销优化、跨集群资源协调和异构计算的调度，以进一步提高 GPU 集群调度系统的灵活性和资源利用效率。

关键词 GPU 集群调度，深度学习，任务切换开销，弹性资源共享，异构计算，多租户环境，Goodput 优化，负载预测，系统弹性与容错性

Comprehensive Review on GPU Cluster Scheduling Systems for Distributed Deep Learning: Challenges and Future Directions

Abstract In recent years, as deep learning applications expand across multiple domains, the demand for high-performance computing resources has intensified, with GPU clusters becoming the primary platform supporting deep learning tasks. However, the efficiency of GPU resource management in multi-tenant environments remains a bottleneck. This paper provides a systematic review of the latest advancements in GPU cluster scheduling in cloud-native environments, focusing on innovations in scheduling strategies across systems like Pollux, DeepBoot, and Aryl. We analyze the core technologies in these systems, including Goodput optimization, elastic resource sharing, unified scheduling, and checkpoint and task recovery mechanisms, highlighting their roles in improving resource utilization and task completion efficiency in clusters. Finally, we propose future research directions, such as optimizing task-switching overhead, cross-cluster resource coordination, and scheduling for heterogeneous computing, to further enhance the flexibility and resource efficiency of GPU cluster scheduling systems.

Keywords GPU Cluster Scheduling, Deep Learning, Task Switching Overhead, Elastic Resource Sharing, Heterogeneous Computing, Multi-Tenant Environment, Goodput Optimization, Load Prediction, System Elasticity and Fault Tolerance

一 引言

近年来，随着深度学习技术在计算机视觉、自然语言处理、语音识别和推荐系统等领域的广泛应用，GPU 集群逐渐成为高性能计算的核心支柱。然而，随着深度学习模型的复杂度增加和数据量的爆炸式增长，对计算资源的需求愈加迫切，多租户环境中的 GPU 资源管理效率问题逐渐凸显。为了满足多用户共享环境下的资源需求并提升集群资源的利用效率，云原生环境下的 GPU 集群调度成为当前研究的重点。尤其是对于深度学习任务，训练任务和推理任务对资源需求各有不同，如何在高效管理资源的同时平衡两类任务的需求，是一项具有挑战性的课题。

本文对当前 GPU 集群调度领域的主要研究成果进行了系统综述，着重分析了在云原生环境中应对深度学习负载的调度策略。文中讨论了 Pollux、DeepBoot、AFS、CoDDL、Aryl 等先进系统的创新机制，包括 Goodput 优化调度、弹性资源共享、训练与推理资源联合调度、以及检查点与任务恢复等核心技术。这些系统通过优化资源配置、动态调整调度策略、减小任务切换开销等手段，有效提升了系统的资源利用率和任务完成效率。本文的目的在于为后续研究提供参考，并探讨未来 GPU 集群调度系统在多租户环境下提升弹性、降低切换开销以及支持异构计算方面的潜在方向。

二 GPU 集群调度的挑战

1 训练与推理任务的资源竞争

在大规模深度学习集群中，训练任务和推理任务通常会被部署在不同的集群中。训练任务需要高计算能力和大 GPU 内存，而推理任务则更加关注低延迟和高吞吐量。

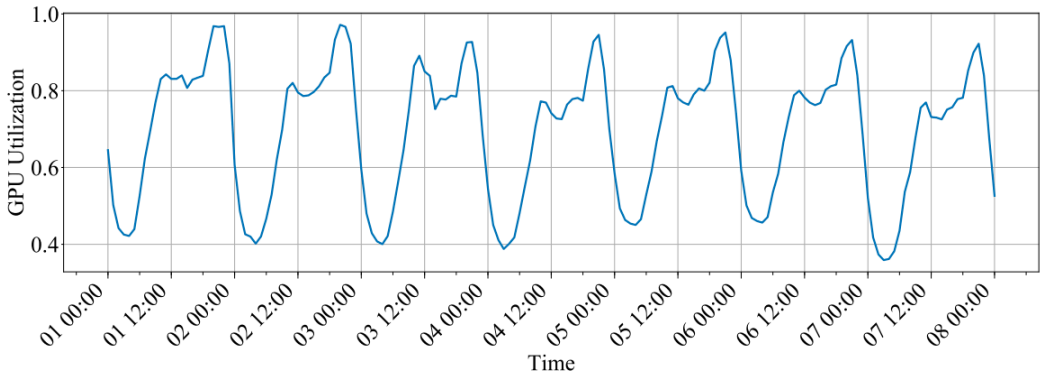


Fig. 1 推理任务的请求量

GPU 资源的低利用率：推理任务的请求量通常呈现周期性波动（如白天低需求，夜间高峰期），导致推理集群的闲置资源浪费严重（利用率 <40%）图 1展示了 Aryl 的推理集群的 GPU 利用率变化，显示出明显的日夜交替模式。

训练任务的长时间排队：训练集群因资源紧张，经常出现长时间的排队现象。例如，Aryl 文中的数据表明，在其 15 天的真实集群记录中，训练任务的平均排队时间超过 3000 秒，部分任务甚至需要等待 10000 秒以上（见图 2）。

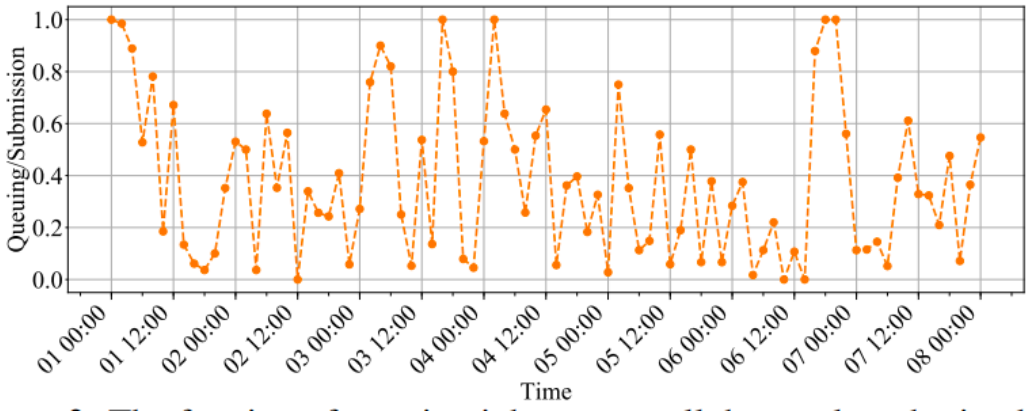
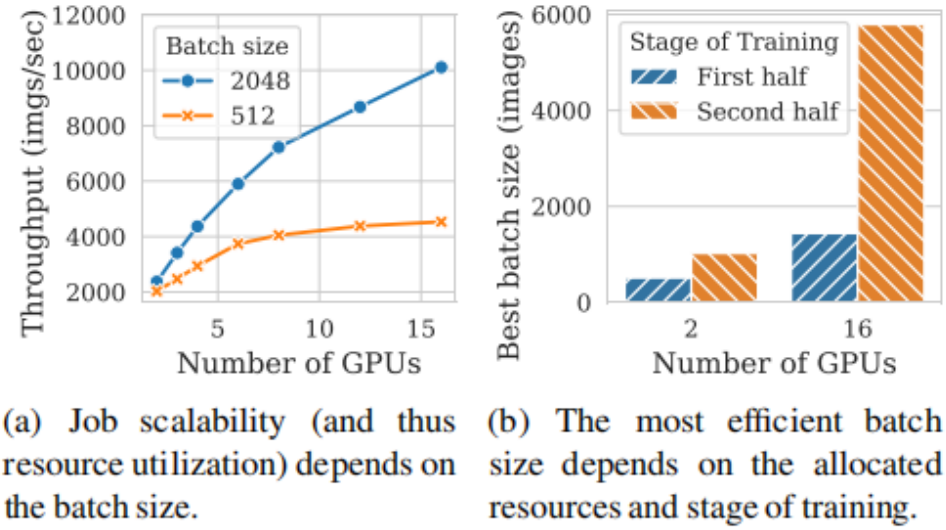


Fig. 2 训练任务的长时间排队



(a) Job scalability (and thus resource utilization) depends on the batch size.

(b) The most efficient batch size depends on the allocated resources and stage of training.

Fig. 3 批次大小与训练阶段关系

2 工作负载的动态性与不确定性

深度学习任务的训练时间往往难以预测，且模型训练过程中资源需求会随训练阶段发生动态变化。例如，模型训练的早期阶段需要更小的批次（batch size）以获得更高的统计效率，而在训练后期可以使用更大的批次以提高系统吞吐量（见图 3Pollux 中的批次大小与训练阶段关系）。

此外，推理请求的高峰期不可预测，需要调度系统能够快速响应资源需求的变化。现有系统在任务预占和资源切换时的开销较高，需要开发更高效的调度算法来解决这些问题。

三 当前研究的主要方法与关键技术

1 Goodput 优化调度

Goodput 优化调度的核心思想是通过优化深度学习任务在资源分配和训练参数配置方面的协同，使得任务在给定时间内能够尽可能高效地利用资源，从而加速模型的收敛。传统的深度学习调度器通常主要关

注系统吞吐量，即单位时间内处理的样本数，但这种单一的指标忽略了每个样本对模型训练进展的实际贡献。因此，Pollux 调度系统引入了 goodput 这一概念，定义为系统吞吐量与统计效率的乘积，它综合考虑了吞吐量与每个样本的有效性，能够更真实地反映训练任务在特定配置下的效率。

在 Pollux 系统中，goodput 的定义如下：

$$\text{GOODPUT}_t(*) = \text{THROUGHPUT}(*) \times \text{EFFICIENCY}_t(M(*))$$

其中，THROUGHPUT 表示系统吞吐量，即每秒处理的样本数；EFFICIENCY 表示统计效率，即每个样本对模型训练的有效贡献度；$M(*)$ 是特定配置下的总批大小。通过这种定义，goodput 能够较为全面地评价任务的训练效果，因为它不仅考虑了处理速度，还考虑了模型在训练数据上的收敛情况。

为了计算系统吞吐量和统计效率,需要构建一套复杂的模型来动态预测它们的变化。系统吞吐量 THROUGHPUT 受多种因素影响，包括 GPU 分配、批大小、网络同步时间等。对于分布式深度学习任务，数据并行同步（data-parallel execution）的模式使得系统吞吐量的瓶颈常常是通信开销，即 GPU 之间的参数同步延迟。因此，当 GPU 数量增加时，每个 GPU 处理的数据量会减少，使得计算时间（Tgrad）缩短，但同步时间（Tsync）保持不变。因此，为了更好地利用更多的 GPU，通常会增大批大小，从而提高计算时间与同步时间的比例（见图 3(a)）。

在 Pollux 中，系统吞吐量可以通过以下公式来描述：

$$\text{THROUGHPUT}(a, m, s) = \frac{M(a, m, s)}{T_{\text{iter}}(a, m, s)}$$

其中 $M(a, m, s)$ 是总批大小，它是 GPU 分配数（a）、单 GPU 批大小（m）和梯度累积步数（s）的乘积，而 Titer 则表示每次迭代的总时间。Titer 的计算公式则通过 Tgrad（梯度计算时间）和 Tsync（同步时间）来获得，它们的计算方式如下：

$$T_{\text{grad}}(m) = \alpha_{\text{grad}} + \beta_{\text{grad}} \cdot m$$

$$T_{\text{sync}}(a, m) = \begin{cases} 0 & \text{if } K = 1 \\ \alpha_{\text{local sync}} + \beta_{\text{local sync}} \cdot (K - 2) & \text{if GPUs are co-located} \\ \alpha_{\text{node sync}} + \beta_{\text{node sync}} \cdot (K - 2) & \text{if GPUs are across nodes} \end{cases}$$

其中 α_{grad} 和 β_{grad} 是梯度计算时间的模型参数，而 $\alpha_{\text{local sync}}$、$\beta_{\text{local sync}}$ 和 $\alpha_{\text{node sync}}$、$\beta_{\text{node sync}}$ 则分别对应同节点和跨节点 GPU 间的通信时间参数。

除了系统吞吐量，Pollux 还动态地评估统计效率 EFFICIENCY，它表示的是每个样本对模型收敛的贡献。统计效率的计算依据梯度噪声比例（Gradient Noise Scale, GNS），它衡量的是随机梯度的噪声信号比。随着批大小增加，统计效率通常会下降，但在训练的不同阶段（如早期和后期）和不同的深度学习模型上，统计效率下降的速度可能不同。为此，Pollux 采用预处理后的梯度噪声比例（PGNS）来计算统计效率。统计效率的计算公式为：

$$\text{EFFICIENCY}_t(M) = \frac{\phi_t + M_0}{\phi_t + M}$$

其中 ϕ_t 是训练时刻的 PGNS，而 M_0 是初始批大小。这样定义的统计效率表示，在相同的训练进展下，不同批大小的模型训练效果可以通过此公式进行预测。

Pollux 系统的架构包含两个核心模块：PolluxAgent 和 PolluxSched。PolluxAgent 是任务级优化模块，与每个深度学习任务一起运行，负责不断监测系统吞吐量、梯度噪声比例等参数，并动态调整批大小和学习率，使得在当前分配的资源下，任务能够达到最佳 goodput。PolluxAgent 根据资源配置计算当前任务的 goodput，并将结果汇报给 PolluxSched。PolluxSched 是集群级的调度器，负责综合考虑整个集群中的所有任务的 goodput，并根据资源利用情况进行动态调整，使得集群资源能够得到最佳利用，同时保证任务之间的公平性。

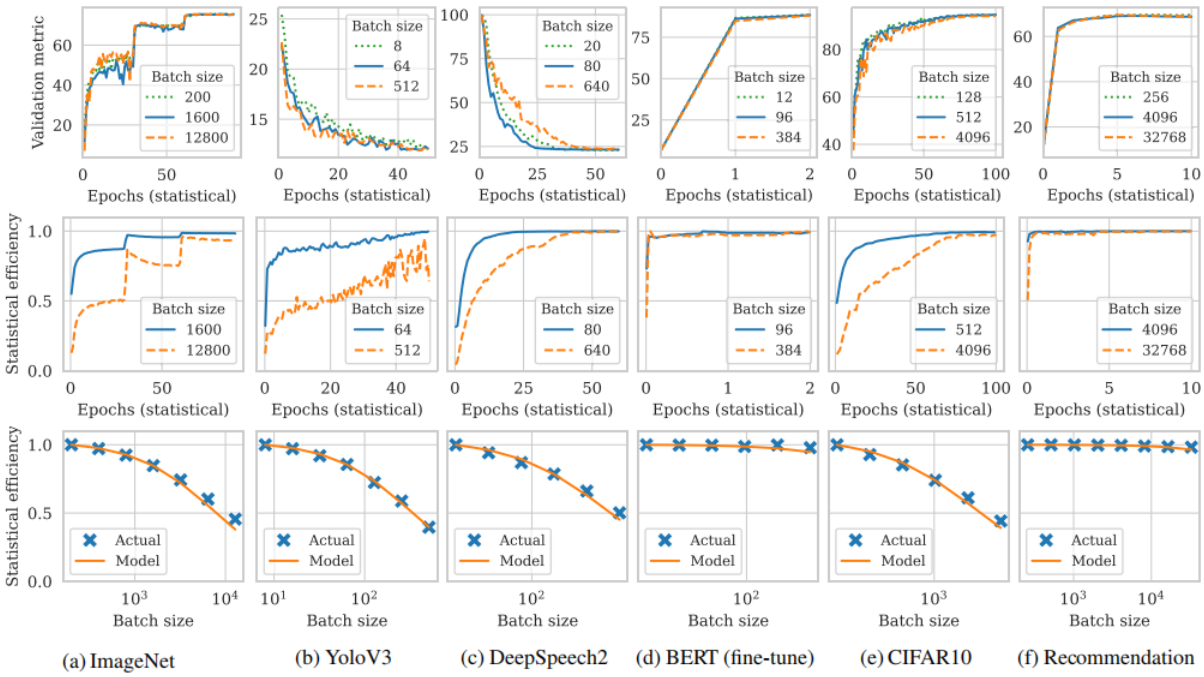


Fig. 4 模型的统计效率

在实验中，Pollux 展示了显著的性能提升，在真实的 DL 任务和基于追踪的模拟中，Pollux 相对于最先进的 DL 调度器，平均任务完成时间减少了 37% 到 50%。这种优势主要体现在 Pollux 能够根据任务的统计效率和系统吞吐量，在高吞吐低效率和低吞吐高效率两者之间进行动态调整。例如，在资源空闲时，Pollux 可以分配更多的 GPU，增大批大小，从而提高系统吞吐量（见图 4），而在资源紧张时，则通过减少批大小来提高统计效率。

Pollux 的 goodput 优化调度在云环境中也展现了新的自动扩展潜力。通过基于 goodput 的自动扩展，Pollux 可以在训练的不同阶段动态调整资源数量，从而在保证模型质量的同时降低成本。在实验中，通过 goodput 驱动的自动扩展相较传统方法，训练大型模型的成本降低了 25%。

综上所述，Pollux 通过 goodput 这一创新性指标，将系统吞吐量与统计效率结合，从而在复杂的深度学习任务中实现高效调度。通过资源和参数的动态优化，Pollux 显著提高了模型训练效率和集群资源利用率。

2 弹性资源共享

AFS（Apathetic Future Share）提出了一种基于未来负载假设的弹性资源调度算法。AFS 方法结合了资源效率和短作业优先的策略，并且考虑到未来资源竞争的不可预见性，设计了一种保守的资源分配假设，避免了传统贪婪调度算法对未来资源可用性的过度乐观预期。

AFS 算法会根据当前资源竞争情况进行动态资源调整：当未来竞争激烈时，优先为更高效的作业分配资源，否则为短作业优先。通过这种方式，AFS 在每次资源配置变化（称为 “churn 事件”）时重新调整各个作业的资源份额，确保系统在大多数情况下能高效分配资源以优化平均 JCT。

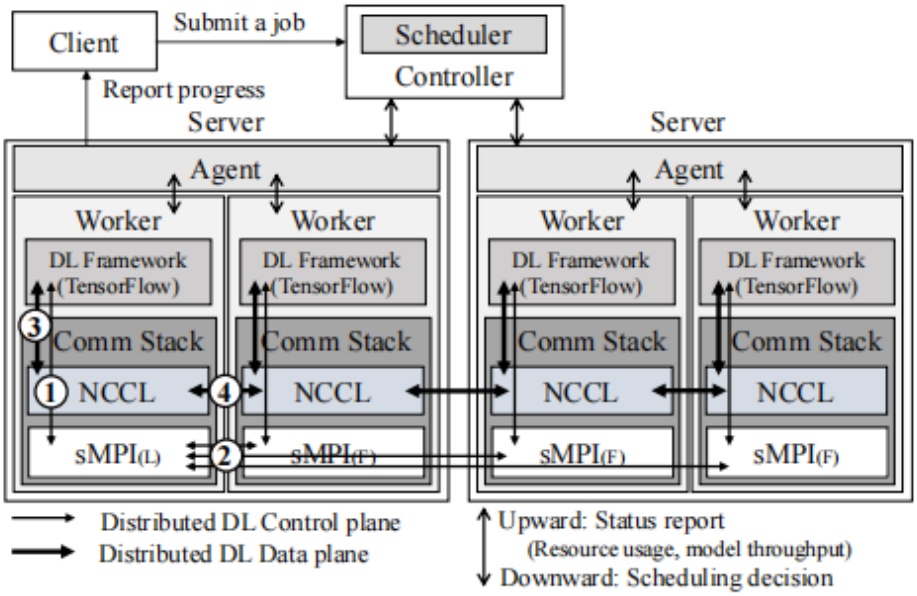


Figure 4: Overview of CoDDL system architecture. There are two servers for the cluster and two GPUs for each server. sMPI (L) and (F) refer to the leader and the follower stacks, respectively.

Fig. 5 CoDDL 的系统框架

为了支持 AFS 的弹性资源分配，作者设计了一个称为 CoDDL 的系统框架。CoDDL 可以自动并行化作业，并高效执行频繁的资源调整。用户在提交模型时无需考虑多 GPU 的并行执行，CoDDL 会自动将单 GPU 模型扩展到多个 GPU 上运行。CoDDL 的系统架构包括一个控制器和多个服务器代理及 GPU 工作节点，通过控制器分配和调整每个作业的资源份额，并通过服务器代理和工作节点来执行具体的训练任务。

在自动化并行训练与资源重调整方面，CoDDL 系统提供了高效的支持。系统会为每个作业设置一个与当前 GPU 份额相匹配的批量大小，并通过累积梯度更新来处理单 GPU 内存不足的问题。CoDDL 还实现了自定义的通信栈，使得工作节点可以在不需要停止和重新启动的情况下动态加入或退出任务，从而避免了常规的 DLT（深度学习训练）自动扩展开销。CoDDL 还设计了快速的扩展和收缩机制，使得在资源配置发生频繁变化时，GPU 的闲置时间最小化。在资源扩展时，作业可以在旧份额上继续执行，新的 GPU 配置完成后再加入作业；在资源收缩时，仅需调整通信栈而无需中断执行。

在面对多个突发的资源重配置命令时，CoDDL 提供了两种处理方法。一种是合并多个连续的配置命令

以简化管理，另一种是取消当前的重配置并快速接受新的配置命令。CoDDL 使用后一种方法以避免阻塞可能导致的死锁。

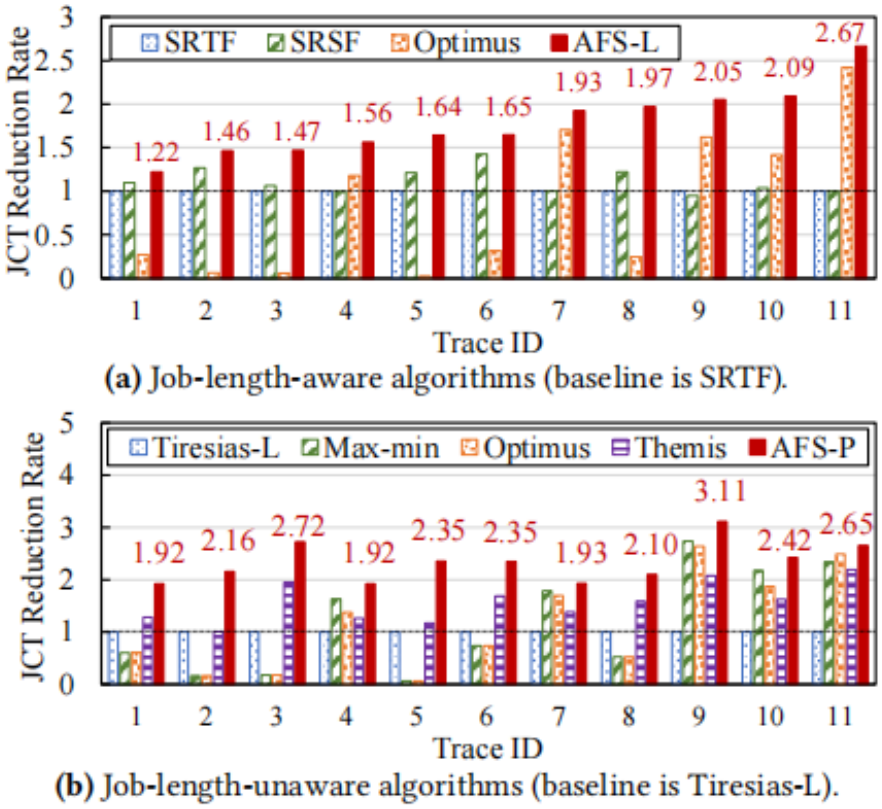


Fig. 6 AFS 相较于其他调度算法对比

AFS 相较于其他调度算法（如 Shortest-Remaining-Time-First (SRTF)）在处理深度学习任务时表现更优，主要因为它考虑了作业的资源效率及未来的竞争情况，使得其在较高的资源竞争环境中能有效地减少作业阻塞问题，从而降低整体的平均 JCT。实验表明，AFS 在平均 JCT 方面比现有的调度算法（如 Themis、SRTF 和 Tiresias-L）有显著的提升，最高能减少 3 倍以上。此外，CoDDL 的弹性资源共享功能大大降低了资源调整的开销，确保了系统的资源利用效率。

3 训练与推理资源的联合调度

DeepBoot 是一种面向深度学习任务（DLT）在 GPU 集群中实现训练和推理资源联合调度的动态系统，旨在解决训练任务的高作业完成时间（JCT）以及推理任务的低 GPU 利用率问题。当前的多租户系统通常会为训练和推理分别配置独立的 GPU 集群，这种模式虽然可以实现二者的隔离，但往往由于推理工作负载的周期性波动导致推理 GPU 的利用率偏低，而训练任务则由于 GPU 资源不足造成长时间排队等待。为了应对这种资源利用不平衡的现状，DeepBoot 提出了一种动态调度系统，通过在推理集群闲置时将其 GPU 资源分配给训练任务，从而提升资源利用效率并降低训练任务的 JCT。

DeepBoot 的核心在于设计了自适应任务缩放（Adaptive Task Scaling, ATS）和自动快速弹性（Auto-Fast Elastic, AFE）两大关键算法。ATS 主要负责在训练和推理任务之间动态分配 GPU 资源，而 AFE 则在推理任务需要优先使用 GPU 资源时，能高效地处理训练任务的缩减与恢复。DeepBoot 的整体设计考虑

了训练和推理任务不同的资源需求及其调度复杂性，保证在训练任务的高 JCT 和推理任务的低 GPU 利用率之间实现有效的资源平衡。

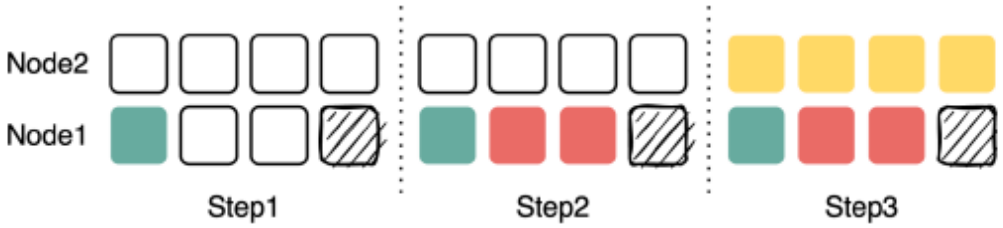


Fig. 7 一个副本分配的例子，包括 3 个具有 1、2 和 4 个 gpu 的任务。黑色绘图 GPU 被不变分配占据

在 DeepBoot 系统中，ATS 分为 ATS-Training（ATS-T）和 ATS-Inference（ATS-I），分别对应训练任务的资源分配和推理任务的资源管理。ATS-T 利用优化算法将训练任务的资源分配问题建模为“背包问题”，从而求解出在训练集群和推理集群内的最优分配策略，以此提升训练任务的效率。具体来说，ATS-T 通过分析任务的 GPU 需求和性能提升比率，以训练速度增益（SPEEDUP）作为优化目标，优先分配可用 GPU 给那些能够大幅提升训练速度的任务（图 7）。通过这种优化，DeepBoot 能够有效减少训练任务在高负载下的等待时间，使其可以利用推理集群中的闲置 GPU 资源，从而缩短训练任务的整体完成时间。为了

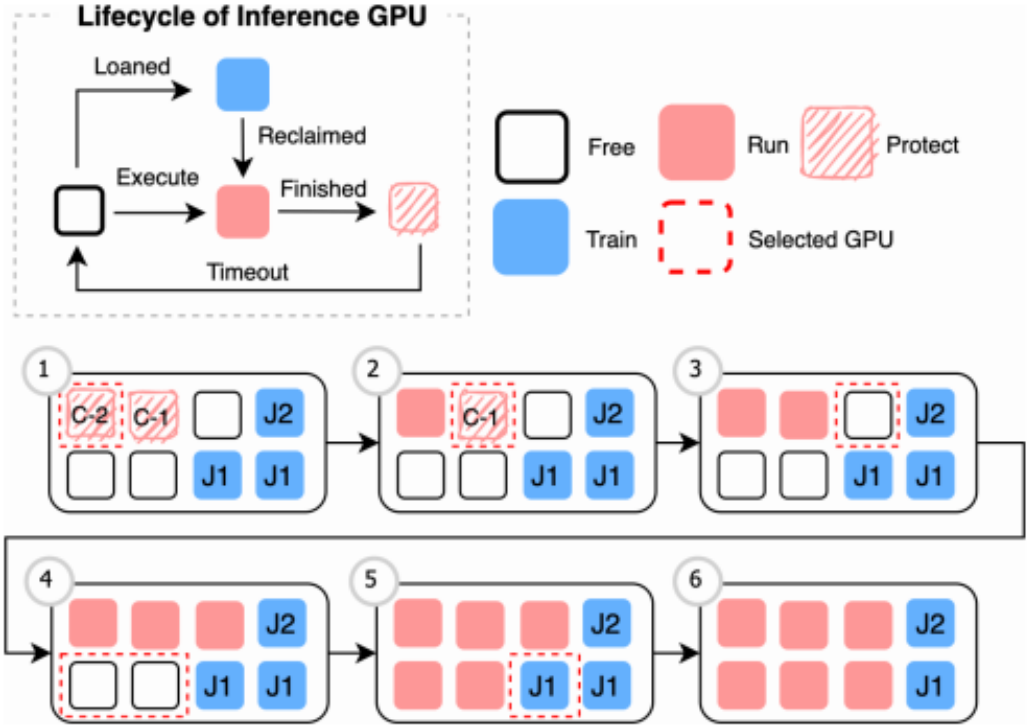


Fig. 8 推理 GPU 的生命周期和推理任务分配

在推理任务高峰期到来时避免 GPU 资源冲突，ATS-I 进一步设计了推理任务的保护机制。在 GPU 处于空闲状态时，系统会将其状态设为“FREE”；当推理任务执行时，GPU 状态将转换为“RUN”；任务完成后进入“PROTECT”状态，防止训练任务在短时间内再次占用该资源（图 8）。这种保护机制能够有效减少

推理任务的频繁抢占行为，从而保证推理任务的实时性需求。对于借用推理资源的训练任务，DeepBoot 还会将其资源调整频率限制在每 60 秒一次，避免训练任务的频繁中断，从而减少重启的开销。

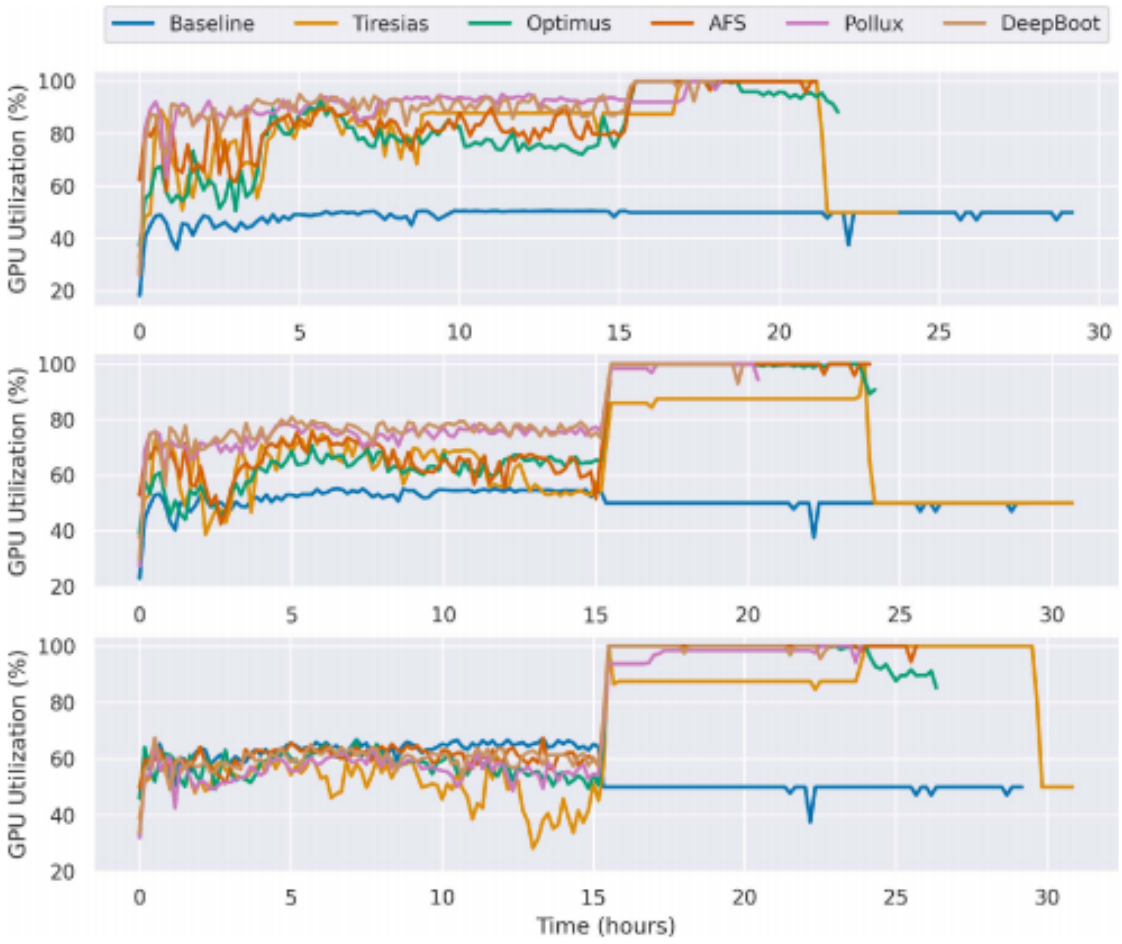


Fig. 9 不同负载条件下各种调度算法的 GPU 使用情况

在调度训练任务的过程中，ATS-T 将任务的资源分配建模为多重选择背包问题，通过动态规划求解出使得 SPEEDUP 最大化的资源分配方案。图 9显示了在不同负载条件下，各种调度算法的 GPU 使用情况。通过这种资源分配方法，DeepBoot 能够在推理集群中借用更多空闲资源，实现较高的 GPU 使用率和较低的任务延迟，从而在不同的负载规模下表现出优越的资源利用效率。在训练任务的资源管理方面，DeepBoot 通过 AFE 实现了弹性训练的自动化管理。当推理任务需要优先占用 GPU 资源时，AFE 能够通过快速缩减训练任务的规模，释放必要的资源给推理任务。AFE 基于 Pollux 进行实现，能够在推理任务完成后快速恢复训练任务的状态，从而减少重启造成的资源浪费。在弹性训练中，DeepBoot 为每个任务分配一定数量的 GPU 资源，并依据任务的负载和资源需求自动调整其分配，以提升任务的整体执行效率。在实验中，AFE 显著减少了由于推理任务抢占带来的训练任务重启开销，使得 GPU 的实际利用率大幅提升（图 10）。

在实际测试和模拟实验中，DeepBoot 在 Microsoft 深度学习工作负载模拟环境中表现出优越的性能。与其他调度系统（如 Optimus、Tiresias、AFS 等）相比，DeepBoot 的 JCT 降低了 32%-38%。通过分析不同算法的调度效率可以看出，DeepBoot 在中等和大负载环境下相较其他算法显著提升了资源利用率（图

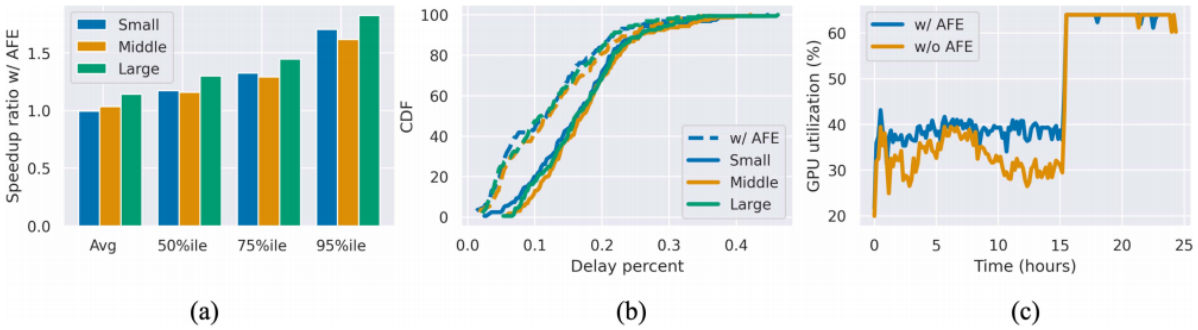


Fig. 10 (a) 平均加速 50%，75% 和 95%。(b) 在不同规模工作负载下执行时间中等待时间百分比的 CDF。虚线是 AFE 的结果。(c) 在大规模工作量下的 GPU 利用率比较

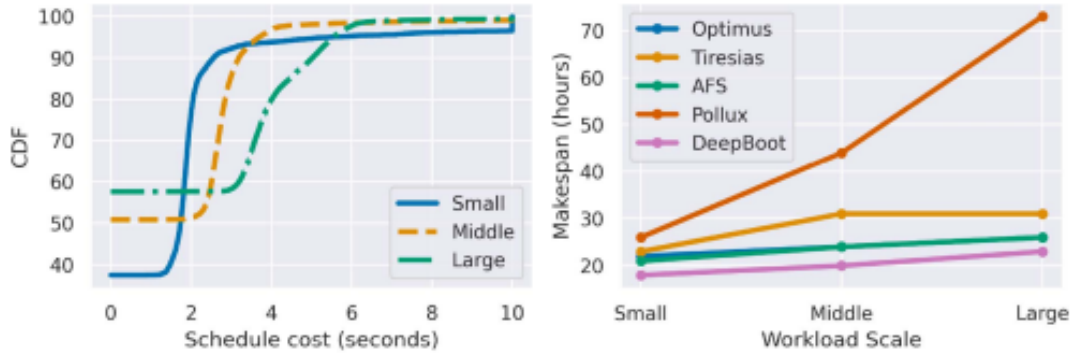


Fig. 11 进度计划的成本分析 (a) CDF(b) 小、中、大规模的调度算法

11)，并在大规模训练集群上表现出更高的任务处理效率。此外，与基于遗传算法的 Pollux 相比，DeepBoot 通过采用更高效的动态规划算法，解决了 Pollux 在大规模任务调度上的迭代时间瓶颈问题，在任务规模增大时实现了更加稳定的性能表现。

与 DeepBoot 类似，Aryl 系统也试图通过利用推理集群中的闲置 GPU 资源来降低训练任务的排队时间，同时提升推理集群的资源利用效率。Aryl 提出了一种容量借用和弹性缩放的方案，通过将低负载推理 GPU 资源借给训练任务以改善训练集群的资源不足问题，同时在推理需求增大时自动收回资源。Aryl 的设计灵感来自于基于容量借用的动态集群管理，其系统架构如图 12所示。推理集群通过容量借用机制，动态决定闲置资源的借出数量；训练调度器接收到借用资源指令后，将任务分配到这些临时资源上，以提高训练集群的整体资源利用率。当推理任务需要优先使用 GPU 时，Aryl 会优先停止那些运行在借用资源上的弹性任务，从而降低因借用资源收回所导致的任务中断次数。

Aryl 的调度系统还支持训练任务的弹性缩放。当推理资源空闲时，Aryl 会通过增加弹性任务的 GPU 资源来加速训练任务，而在资源紧张时则缩减任务规模，释放资源以供更高优先级的任务使用。Aryl 将调度问题建模为多重选择背包问题，任务的 GPU 需求被分成多个弹性选项，通过优先分配提升 JCT 降低值最大的任务，来实现更好的资源利用。Aryl 在测试中实现了 1.50 倍的平均 JCT 提升，并在调度训练任务时，通过更高效的背包算法解决了传统调度方法的资源浪费问题。

与 DeepBoot 不同的是，Aryl 在资源回收策略上采用了基于成本的最小化策略，即在回收资源时优先选

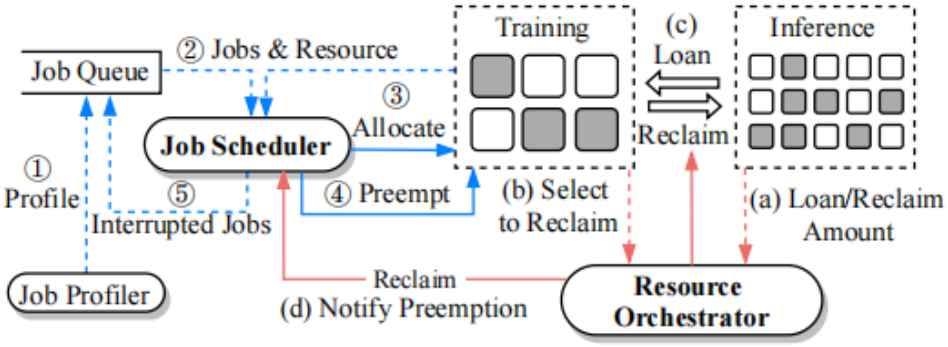


Fig. 12 Aryl 系统架构

择对训练任务影响最小的服务器，从而最大限度地减少因推理任务需要而中断的训练任务数量。此外，Aryl 在任务调度方面侧重于将任务按资源需求进行分组分配，保证任务在集群内的最佳放置，以避免资源碎片化。

总体而言，DeepBoot 和 Aryl 都展示了在深度学习任务调度中的优势，前者注重在推理任务高峰期的资源保护与弹性训练的恢复效率，后者则通过容量借用和基于成本的回收策略，在生产环境下有效提升了集群资源利用率和任务完成效率。

4 检查点与任务恢复机制

在《DeepBoot: Dynamic Scheduling System for Training and Inference Deep Learning Tasks in GPU Cluster》和《Tiresias: A GPU Cluster Manager for Distributed Deep Learning》两篇文献中，检查点和任务恢复机制的核心目标是确保任务在多租户 GPU 集群中的高效管理。GPU 资源有限且共享的情况下，训练任务和推理任务常常会产生冲突，需要设计合理的机制来减少任务中断时的性能损失。在这两篇论文中，检查点机制被用来记录任务的执行状态，确保在任务暂停和恢复时能够从之前的状态继续执行，而不是重新开始。两篇论文针对不同场景提出了不同的解决方案，分别应对训练和推理任务之间的动态切换，以及大规模分布式任务的抢占式调度。

在 DeepBoot 的设计中，推理任务的优先级较高，当推理任务到来时，GPU 资源必须及时转交给推理任务，从而导致训练任务暂停。为了减少频繁切换所带来的性能损失，DeepBoot 引入了 Auto-Fast Elastic (AFE) 机制。在 AFE 机制中，每当训练任务被暂停时，系统会将模型的当前状态、参数以及数据访问进度保存为检查点。这样，在推理任务结束后，当训练任务恢复时，它可以迅速加载之前保存的状态，避免重新初始化模型和数据。在论文中通过图 1 展示了不同类型的模型在被中断和重启时的时间成本。较轻量级的模型，如 ResNet 在处理小型数据集时，主要的时间开销集中在容器的创建与销毁上。而 BERT 等模型由于处理的是大型数据集，其主要的时间消耗则体现在数据加载阶段。这表明在不同规模和类型的任务中，任务中断与恢复的时间开销存在显著差异。

为了进一步减少 GPU 频繁切换的开销，DeepBoot 在推理任务结束后设置了 GPU 保护状态。当 GPU 资源进入保护状态时，尽管推理任务已经完成，但 GPU 资源不会立即释放给训练任务，而是保留一段时间以应对潜在的推理任务请求。这个设计避免了推理任务高峰期频繁切换 GPU 的情况，从而减少了训练任务因频繁暂停而产生的开销。在图 8中可以看到 GPU 的生命周期管理示意图，展示了 GPU 在训练任务和

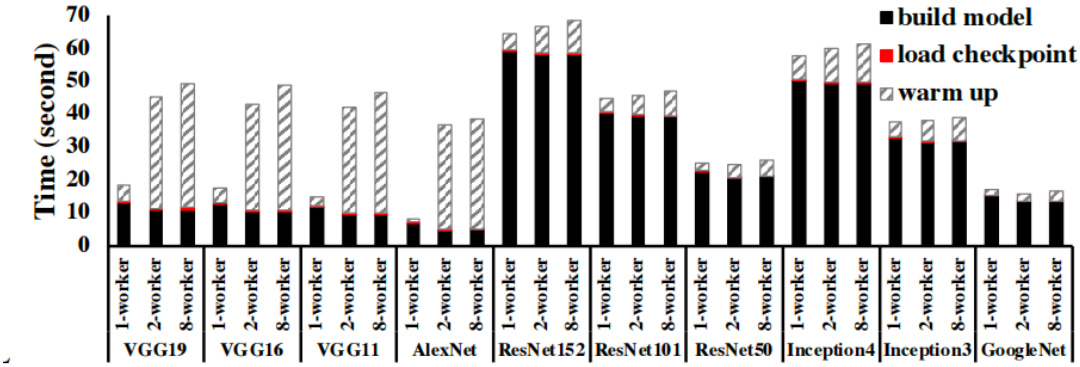


Fig. 13 不同模型在恢复过程中的时间成本

推理任务之间的切换过程中如何进入和退出保护状态。这种机制确保了资源的稳定性，并通过减少不必要的任务恢复操作来提高系统的总体性能。

Tiresias 的研究侧重于如何在大规模集群中高效管理分布式训练任务。在分布式训练环境中，由于任务规模大且执行时间长，资源的抢占和恢复需要更加精细的管理。Tiresias 提出了抢占式调度策略，使得高优先级任务可以随时中断低优先级任务以获取资源。在任务被抢占时，当前任务的模型状态会以检查点的形式保存到共享存储中，并在资源可用时从检查点恢复。这一过程不仅包括模型状态的保存与加载，还涉及到任务的重新初始化和预热步骤。在图 13中展示了不同模型在恢复过程中的时间成本，从构建模型到加载检查点，再到完成预热，各个步骤的时间开销差异明显。

为了避免因抢占导致的频繁恢复带来过高的系统开销，Tiresias 引入了优先级离散化机制。系统将任务按照已消耗的服务时间划分到不同的优先级队列中，每个任务的优先级会随着其运行时间的增加而逐渐降低，从而减少高优先级任务频繁抢占资源的情况。多级队列的设计使得系统在确保高优先级任务及时获得资源的同时，也能保证低优先级任务不会被无限期地推迟。通过这种优先级管理，Tiresias 有效避免了传统抢占式调度中的资源浪费，并最大程度地减少了任务恢复的频率。文中的图 8 展示了多级队列的结构和任务在队列之间的流转过程，每个任务在不同优先级队列中的移动都基于其已经获取的服务时间。当一个任务在某个队列中等待过久时，Tiresias 还会将其提升到更高优先级的队列中，以避免任务因为资源不足而陷入“饥饿”状态。这种设计确保了所有任务最终都能获得执行机会，同时最大化了 GPU 资源的利用率。

Tiresias 在抢占和恢复机制上还考虑了通信延迟的影响。在大规模分布式任务中，每次抢占和恢复不仅会涉及 GPU 上的计算状态保存和加载，还需要处理数据在 GPU 与内存之间的传输。尤其是在任务规模较大的情况下，这些传输时间可能成为系统性能的瓶颈。为了减少传输延迟，Tiresias 采用了两阶段优先级调度算法（2DAS），该算法在计算任务优先级时，综合考虑了任务占用的 GPU 数量（空间维度）和任务已运行的时间（时间维度）。这使得系统能够更智能地决定哪些任务需要暂停，哪些任务应该优先恢复。图 14中的时间序列展示了不同调度算法下的任务调度效果，表明 2DAS 在减少等待时间和提高资源利用率方面优于其他单维度的调度算法。

与 DeepBoot 的设计相比，Tiresias 更注重任务抢占过程中的细节管理，如检查点的频率、存储的位置以及如何减少通信开销。这使得 Tiresias 在面对大规模分布式任务时表现更为出色。然而，与 Tiresias 注重分布式训练任务的抢占不同，DeepBoot 更倾向于解决训练和推理任务之间的资源共享问题。DeepBoot

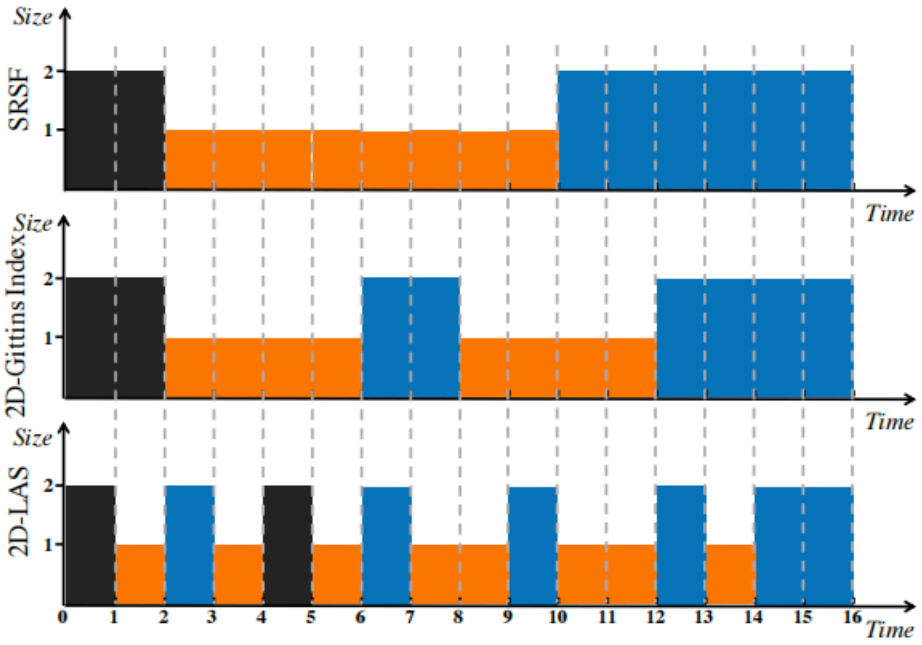


Fig. 14 不同调度算法下的任务调度效果

通过 GPU 保护状态和弹性训练机制，使得训练任务能够在推理任务高峰期灵活调整其资源需求，并在资源充足时快速恢复。这种灵活性在需要频繁切换任务类型的场景中尤为重要。

总体来看，这两篇论文展示了在多租户 GPU 集群环境中，实现高效任务管理和检查点恢复的不同策略。DeepBoot 的设计适用于需要频繁在推理和训练任务之间切换的场景，其弹性训练和快速恢复机制能够有效减少因任务中断带来的开销。而 Tiresias 则更适合大规模分布式训练任务的管理，通过抢占式调度和优先级管理，确保资源能够合理分配，并减少任务抢占带来的恢复开销。这两种机制各有优劣，在实际应用中，可以根据系统的具体需求进行选择或结合使用，从而在 GPU 资源有限的情况下，实现更高效的任务管理与调度。

四　研究挑战与未来方向

当前的 GPU 集群调度系统在应对多任务环境中的表现已有显著提升，但仍然面临多方面的挑战，这些挑战为未来的研究指明了方向。首先，任务切换的开销与实时性优化仍是一个亟待解决的问题。在训练与推理任务共存的环境中，资源切换不可避免地会导致任务暂停与重启。DeepBoot 系统的研究表明，当训练任务因推理请求被中断时，任务的重启过程涉及容器的删除与重建、模型初始化以及数据加载等复杂步骤，这些过程可能耗费 20 至 100 秒的时间，严重影响系统的响应速度。此外，对于某些实时性要求较高的推理任务，如在线推荐和语音识别，毫秒级的响应时间尤为关键。然而，现有系统在频繁资源切换时仍面临性能损失的问题，需要进一步优化。这一领域未来的研究可以借鉴 PipeSwitch 系统的思路，通过 CUDA 上下文的预初始化实现毫秒级切换，以减少任务暂停带来的开销。

跨集群资源的协调与优化是另一个重要的研究方向。随着深度学习任务的复杂度不断提高，单一集群往往无法满足所有的计算需求。在多租户云环境中，不同用户对资源的需求存在显著差异，这导致了资源的竞争与不公平分配问题。例如，Aryl 系统的研究表明，推理集群的负载具有明显的周期性，而训练任务

的负载则更为随机。因此，如何在多个集群之间高效地共享资源是一个亟待解决的问题。未来的研究可以开发智能化的资源管理系统，通过分析历史数据和实时监控，提前预测负载的变化，并在不同集群之间动态调整资源分配。此外，Pollux 系统的 Goodput 优化策略尽管实现了高效的资源使用，但在多租户环境中仍需要进一步改进，以确保在资源紧张时不同用户之间的公平性。

新型硬件的集成与异构计算环境的调度同样是未来的重要研究领域。随着 TPU、FPGA 等新型硬件的逐渐普及，GPU 不再是唯一的计算选择。然而，这些不同类型的硬件适用于不同的计算任务。例如，TPU 在大型神经网络的矩阵计算中表现优异，而 FPGA 则更适合执行特定的定制化任务。如何在这些异构硬件之间高效调度任务，是当前系统的一大挑战。未来的研究可以采用深度强化学习等智能算法，根据任务的特性自动匹配最合适的硬件资源。此外，开发支持异构资源共享的新模型，使不同硬件能够协同工作，将进一步提升集群的整体性能。

负载预测与自适应调度是未来调度系统发展的关键方向之一。GPU 集群中的负载波动具有高度的动态性，不同时间段的资源需求可能会发生剧烈变化。Aryl 系统展示了推理任务在一天中的负载变化曲线，揭示了其明显的日夜波动特征。而训练任务的负载则更为不可预测，这为调度系统带来了巨大挑战。传统的 Shortest-Remaining-Time-First（SRTF）算法由于无法及时响应突发负载，在动态环境中的表现受到限制。因此，未来的调度系统需要采用机器学习算法来预测负载变化，并根据预测结果提前调整资源分配。AFS 系统在这一方面做出了一些探索，但未来还可以进一步开发更为智能的自适应调度算法，以实时调整任务的优先级和资源配额。

最后，多租户环境中的系统弹性与容错性是确保集群稳定运行的关键。共享资源环境中的硬件故障或网络波动可能导致模型训练的中断，这不仅会影响单个任务的完成，还可能影响整个集群的稳定性。Tiresias 系统提出了优先级离散化策略，以减少频繁任务中断带来的开销。然而，未来的研究需要进一步增强系统的容错能力，例如开发更加智能的检查点机制，使系统在故障发生时能够自动恢复，并在恢复过程中优化资源分配。此外，多租户环境中的租户隔离机制也需要改进，以确保一个租户的任务故障不会对其他租户产生影响。通过引入容错调度算法，系统可以优先保障关键任务的资源需求，减少服务中断对用户体验的影响。

综上所述，未来的 GPU 集群调度系统需要在多个方面进行改进与创新。任务切换的开销优化、跨集群资源的协调、新型硬件的集成、负载预测与自适应调度以及系统的弹性与容错能力，将是推动这一领域发展的重要方向。随着这些问题的逐步解决，GPU 集群将能够更高效地支持日益复杂的深度学习任务，为大规模 AI 模型的训练和推理提供更强大的计算能力。

五　总结

本文系统性地回顾了当前 GPU 集群调度系统在云原生环境下的主要技术进展。通过对 Pollux、DeepBoot、Aryl 等典型系统的分析，我们总结了各系统在资源利用率、任务完成效率以及多租户环境适应性方面的贡献。这些系统采用了 Goodput 优化、联合调度、弹性资源共享以及任务恢复等策略，通过动态调整资源分配和调度策略，成功缓解了深度学习任务中的资源竞争和动态负载波动带来的挑战。

尽管当前系统在资源调度上已取得显著进展，本文提出未来的研究应着重解决任务切换开销、跨集群资源协调、异构硬件调度以及负载预测与自适应调度等问题。通过进一步优化这些方面，GPU 集群调度系统将能更高效地支持日益复杂的深度学习任务，为大规模 AI 模型的训练与推理提供可靠的计算平台。

参 考 文 献

[1] CHEN Z, ZHAO X, ZHI C, et al. Deepboot: Dynamic scheduling system for training and inference deep learning tasks in gpu cluster[J]. IEEE Transactions on Parallel and Distributed Systems, 2023.

[2] LI J, XU H, ZHU Y, et al. Aryl: An elastic cluster scheduler for deep learning[A]. 2022.

[3] QIAO A, CHOE S K, SUBRAMANYA S J, et al. Pollux: Co-adaptive cluster scheduling for goodput-optimized deep learning[C]//15th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 21). 2021.

[4] HWANG C, KIM T, KIM S, et al. Elastic resource sharing for distributed deep learning[C]//18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21). 2021: 721-739.

[5] GU J, CHOWDHURY M, SHIN K G, et al. Tiresias: A {GPU} cluster manager for distributed deep learning[C]// 16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19). 2019: 485-500.