分　数：	
评卷人：	

華中科技大學

研究生（高级计算机体系结构）
课程论文（报告）

题　目：Pollux Co-adaptive Cluster Scheduling for Goodput-Optimized Deep Learning

学　　号 _____

姓　　名 _____

专　　业　计算机系统结构

类　　别　非定向硕士

课程指导教师　谢长生、曹强、肖亮

院（系）、年级　计科 2021 级

2022 年 1 月 9 日

一、论文写作大纲和整体逻辑结构

Pollux Co-adaptive Cluster Scheduling for Goodput-Optimized Deep Learning

Abstract

Pollux 通过在每个作业级别和整个集群级别自适应地协同优化相互依赖的因素，提高了深度学习（DL）集群中的调度性能。大多数现有的调度程序都希望用户指定每个作业的资源数量，这通常会导致资源使用效率低下。近期出现的一些调度器为用户选择作业资源，但这样做时，并不知道如何重新优化 DL 训练以更好地利用提供的资源。

Pollux 同时考虑了这两个方面。通过在训练期间监控每个作业的状态，Pollux 模拟了他们的 goodput（作者引入的一个指标，将系统吞吐量与统计效率结合起来）将如何通过添加或删除资源而改变。Pollux 动态（重新）分配资源以提高集群范围的吞吐量，同时尊重公平性并不断优化每个 DL 作业以更好地利用这些资源。

在实际 DL 作业和跟踪驱动模拟的实验中，Pollux 与最先进的 DL 调度器相比，即使为每个作业提供了理想的资源和训练配置，平均作业完成时间也减少了 37–50%。Pollux 基于对有用作业进度的更有意义的度量，促进 DL 作业之间竞争资源的公平性，并揭示了在云环境中降低 DL 成本的新机会。Pollux 作为开源项目的一部分发布在 https://github.com/petuum/adaptdl.

1. Introduction

深度学习（DL）训练已迅速成为许多共享资源环境（如数据中心和云）的主要工作负载。DL 作业是资源密集型且长期运行的，通常要求使用昂贵的硬件设备（如 GPU 或 TPU）进行分布式执行，以便在合理的时间内完成。为了满足这一资源需求，通常为深度学习提供专用集群，并配备一个调度程序，在许多相互竞争的 DL 作业之间协调资源共享。现有调度程序要求用户手动配置其作业，如果操作不当，可能会大大降低训练性能和资源效率。

在共享集群环境中，最佳选择是动态的，并且取决于作业运行时的集群负载。尽管最近的弹性调度器可以为每个作业自动选择适当数量的资源，但它们会盲目地选择与训练相关的相互依赖的配置，这些配置同样重要。如果没有关于集群硬件性能和 DL 模型体系结构的专家知识，资源量、batch 大小和学习率很难进行

适当配置。

集群调度器自动配置用户提交的 DL 作业的根本问题是在两个相反的期望之间取得了平衡：

（1）系统吞吐量，即每壁时钟时间处理的训练考试次数；

（2）统计效率，即每个训练示例处理的进度。

此外，统计效率下降的速度相对于批大小取决于当前的训练进度。与早期的训练相比，后期训练阶段的工作可以承受 10 倍或更大的批量，而不会降低统计效率。在这些观点的指导下，本文提出了一种混合资源调度器 Pollux，它可以共同自适应地分配资源，并为共享集群中的所有 DL 作业调整批处理大小和学习率。Pollux 通过联合管理几个系统级别和训练相关的参数实现这一点，包括 GPU 数量、workers 的共同位置、每个 gpu 批大小、梯度积累和学习率缩放。本文的贡献主要体现在：

（1） 提出了 DL 工作的 goodput 公式，这是一个整体的训练衡量，考虑到系统吞吐量和统计效率。

（2） 通过观察训练过程中作业的吞吐量和统计行为，作者证明了一个 DL 作业的 goodput 模型可以被学习，并用于预测给定不同资源分配和批处理大小的性能。

（3） 设计和实现了一个调度体系结构，它使用这些模型来为 DL 作业配置正确的资源分配和训练参数组合。并通过本地和全局的组件之间的交互来最大化 goodput。

2. Background: Distributed DL Training

2.1 System Throughput

DL 训练的系统吞吐量可以定义为单位时钟时间内处理的训练样本数。当一个 DL 作业分布在几个节点上时，它的系统吞吐量由几个因素决定，包括(1)分配给该作业的资源(如 GPU)的分配和分配，(2)分布式执行和同步的方法，以及(3)批处理大小。

数据并行执行。同步数据并行是分布式执行 DL 训练的一种流行方法。模型参数被复制到一组分布式的 GPU 上，每个小的批处理被划分为每个节点等大小的分区，每个 GPU 用自己的分区计算一个局部的梯度估计，然后再所有 GPU 上求这个局部梯度估计的平均值，得到需要的梯度估计。最后每个节点采用相同的梯度估计来得到新的参数模型。

由批大小造成的延迟。根据 Amdahl 定律，无论使用多少 GPU，每次训练迭代的运行时间都以 T_{sync} 为下限。为了克服这种可伸缩性限制，一种常见的策略是增

加批处理大小，不过这样做会导致在更多的训练中计算局部梯度估计。因此，当在同步数据并行设置中扩展到更多的 GPU 时，使用更大的批处理大小可以实现更高的系统吞吐量。

2.2 Statistical Efficiency

DL 训练的统计效率可以定义为每处理一个训练数据单元所取得的训练进展量，受诸如批次大小或学习率等参数的影响。例如，较大的批量通常会降低统计效率。预测统计效率的能力是提高统计效率的关键，因为可以使用预测来更好地适应批处理大小和学习率。

具体的影响因素有两个，一是梯度噪声的规模，DL 训练的统计效率与梯度噪声标度（GNS）相关，后者测量随机梯度的噪声信号比。更大的 GNS 意味着训练参数（如 batch 大小和学习率）可以增加到更高的值，而统计效率的降低相对较小。GNS 在不同的 DL 模型之间可能有很大差异，并且在训练期间会逐渐增加。因此，在训练后期，对于大批量的任务可以获得显著更好的统计效率。

第二个是学习率的增长。当使用增加的总批量进行训练时，学习率 η 也应增加，否则最终训练的模型质量/精度可能会显著降低。

2.3 Existing DL Schedulers

作者将现有的 DL 调度器大致分为两类：

首先是非规模自适应调度程序，非规模自适应调度程序不知道 DL 作业相对于分配资源量的性能可伸缩性。例如，Tire-sias 要求用户在提交作业时指定 GPU 的数量，该数量在作业的生命周期内是固定的。Gandiva 还要求用户指定 GPU 的数量，但通过细粒度的时间共享和作业打包提高了资源利用率。

其次是缩放自适应调度程序，它会根据每个作业的资源利用率自动决定分配给每个作业的资源量，以加快作业的速度。例如，Optimus 学习给定各种资源量的每个作业的系统吞吐量的预测模型，并优化集群范围的资源分配，以最小化平均作业完成时间。SLAQ 没有在 DL 上进行评估，它使用类似的技术来最小化训练一般 ML 模型的平均损失值。Gavel 更进一步，基于吞吐量指标进行调度，该指标在不同加速器类型之间具有可比性。AntMan 使用动态扩展和细粒度 GPU 共享来提高集群利用率、资源公平性和作业完成时间。Themis 引入了完成时间公平性的概念，并通过两级调度架构促进了多个 DL 应用程序之间的公平性。

关键的是，现有的调度程序对 DL 训练的统计效率以及资源决策和训练参数的相互依赖性是不可知的。Pollux 显式地共同调整这些相互依赖的值，以提高 DL 作业的 goodput。

3. The Goodput of DL Training and Pollux

在本节中定义了 DL 作业的 goodput，它是考虑系统吞吐量和统计效率的训练性能的度量。然后作者描述了如何在训练期间度量 goodput，并将其用作预测模型，该模型由 Pollux 共同优化集群范围的资源分配和批处理大小。

作者给出了 goodput 的定义如下：

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

也就是一个 DL 训练任务在第 t 轮迭代时的收益是它在第 t 轮迭代时的系统吞吐量和统计效率之间的乘积，因此可以说 goodput 综合考虑了吞吐和系统效率。

为了最优化 goodput，需要对吞吐 Throughput 和效率 Efficiency 两个指标进行建模并作实时的评估。下面简单介绍本文对吞吐 Throughput 建模的方式。

3.1 Modeling Statistical Efficiency

本节主要介绍 Pollux 对 Efficiency 的建模。

Pollux 将 Efficiency 建模为 $\text{EFFICIENCY}_t(M_0)$的相对值，并在调整批大小的过程中只考虑了 $M > M_0$，因此 Efficiency 的取值范围为(0,1]，例如如果某个时刻采用批大小为 M 时的 Efficiency 是 E，表达的含义是需要用 1/E 的样本来训练才能够达到初始批大小 M_0 用一个样本能够达到的 progress。

下面的公式描述了这种关系：

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

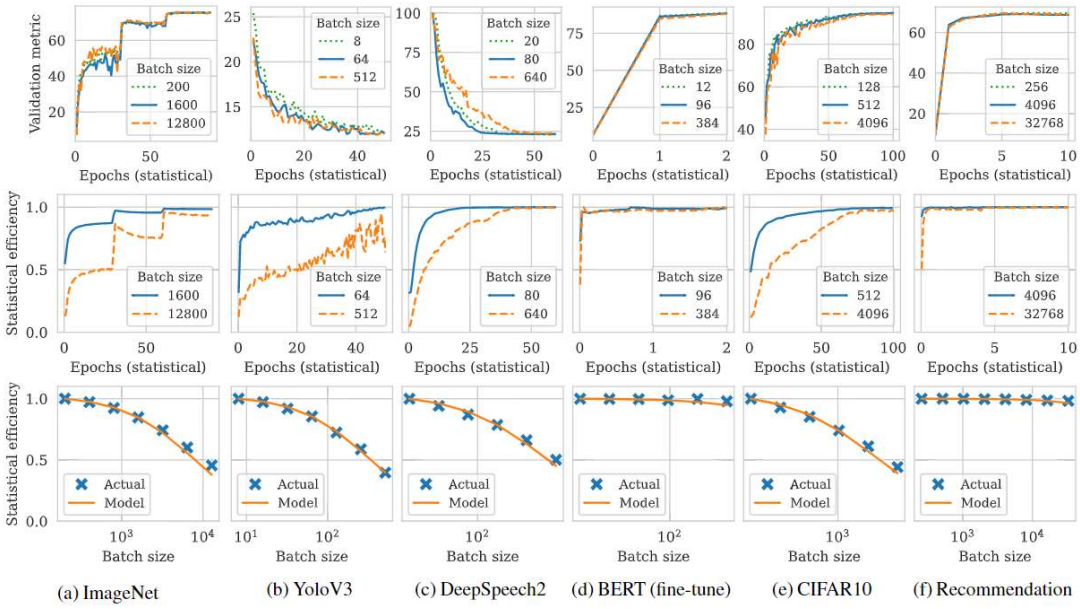


图 2 各 DL 训练任务的训练进度

正如作者的模型所预测的那样，每个 epoch 在理论上使不同批量的训练进度相同。因此，不同批量的验证曲线之间的相似度是效率准确性的指标，作为实际训练进度的预测指标。尽管几个 DL 任务的验证曲线存在差异（尤其是在早期），但它们在作者评估的不同批量中获得了相似的最佳值（除 DeepSpeech2 外，所有任务的相对差异为±1%，为±4%）。

图 2（中间和底部）显示了训练期间以及不同批量范围内的测量和预测效率。一般来说，较大的批量在训练初期效率较低，但与“规模不变迭代”概念类似，缩小了差距。例外情况是 BERT，它是从已经预先训练好的模型开始的微调任务，使用比其他模型更小、更浅的模型体系结构。在训练过程中，效率的变化因任务而异，这取决于特定的属性，如学习率计划。例如，ImageNet 使用基于步长的学习率退火，每当学习率退火时，其效率都会急剧提高。

3.2 Modeling System Throughput

对于吞吐 Throughput，根据定义，可以通过计算批的大小与每个 iteration 的时间的比值来计算：

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

而其中的 T_{iter} 可以通过以下方式进行计算：

$$T_{iter}(a,m,s)=s\times T_{grad}(a,m)+\left(T_{grad}(a,m)^{\gamma}+T_{sync}(a)^{\gamma}\right)^{1/\gamma}.$$

下图给出了验证上述模型准确性的测试，可以看出这个模型基本上很准确地对吞吐 Throughput 进行了评估。

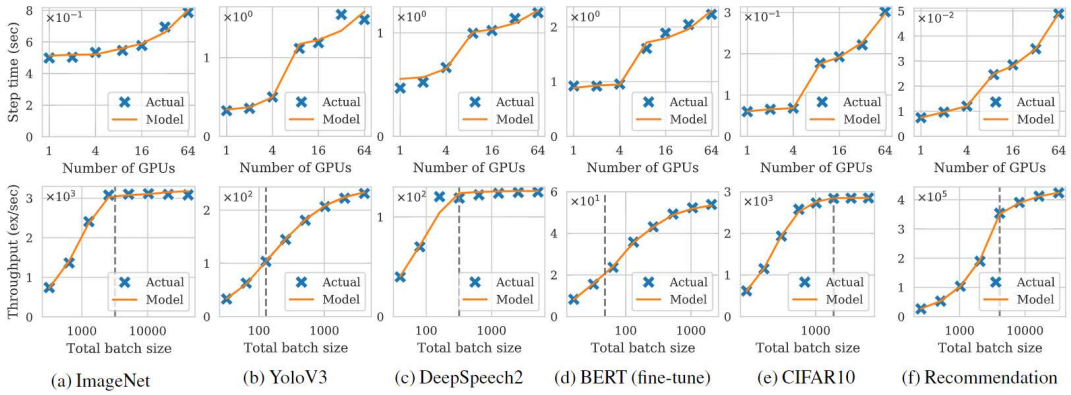


图 3 各 DL 训练任务的吞吐

4. Pollux Design and Architecture

Pollux 以两种不同的粒度调整 DL 作业执行。首先，在作业级别的粒度上，Pollux 动态调整批大小和学习率，以实现所分配资源的最佳利用率。其次，在集

群范围内的粒度上，Pollux 动态（重新）分配资源，这是由共享集群的所有作业的 goodput 以及集群级别的目标（包括公平性和作业完成时间）驱动的。为了以可伸缩的方式实现这种协同自适应，Pollux 的设计包括两个主要组件，如图 4 所示。

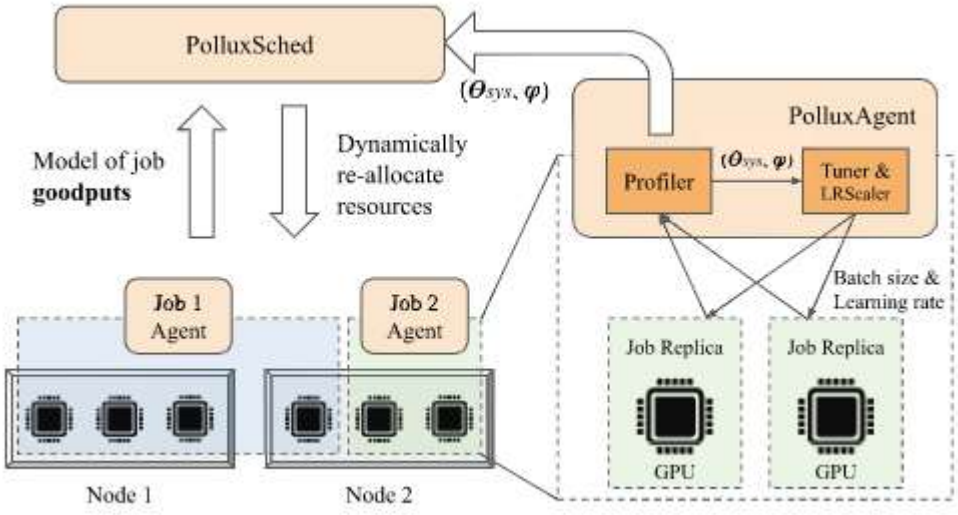


图 4 Pollux 的自适应调度架构

首先，PolluxAgent 与每个作业一起运行。它适合该作业的效率和吞吐量函数，并调整其批处理大小和学习率，以有效利用其当前分配的资源。PolluxAgent 定期向 PolluxSched 报告其作业的 goodput。

其次，PolluxSched 会定期优化集群中所有作业的资源分配，考虑到每个作业当前的 goodput 函数和集群范围内的资源争用。PolluxSched 做出的调度决策还考虑了与资源重新分配相关的开销、由于多个作业之间的网络干扰而导致的速度放缓，以及资源公平。

PolluxAgent 和 PolluxSched 相互适应。PolluxAgent 适应每个训练工作，使其有效地使用其分配的资源，而 PolluxSched 动态地重新分配每个工作的资源，考虑到 PolluxAgent 调整其工作的能力。

4.1 PolluxAgent: Job-level Optimization

PolluxAgent 的一个实例将从每个训练工作开始。在训练过程中，它不断地测量作业的梯度噪声规模和系统吞吐量，并以固定的间隔向 PolluxSched 报告。它还使用这些信息来确定给定其当前资源分配的最有效的批处理大小，并使用适当的插件 LR 缩放规则(例如，SGD 的 AdaScale 或 Adam 的平方根缩放)调整其任务的学习率以适应这个批处理大小。

一旦找到新的配置，作业将使用它进行后续的训练迭代，使用插件 LR 缩放

规则来适当地调整其学习率。由于作业的效率功能随时间变化，PolluxAgent 将定期重新评估最有效的配置。

4.2 PolluxSched: Cluster-wide Optimization

PolluxSched 负责为每个作业任务动态分配 GPU 资源，PolluxSched 的调度目标是最大化 fitness。

$$\text{FITNESS}_p(A) = \left(\frac{1}{J} \sum_{j=1}^{J} \text{SPEEDUP}_j(A_j)^p \right)^{1/p}.$$

A 为分配矩阵，A_j 为作业 j 的每一行分配向量，因此 A_{jn} 为节点 n 上分配给作业 j 的 GPU 个数，j 为集群中运行和等待的作业总数。作者将每项工作的 SPEEDUP 定义为使用给定资源分配下 goodput 的提升和使用公平资源分配下的比值，可根据下式进行计算。

$$\text{SPEEDUP}_j(A_j) = \frac{\max_{m,s} \text{GOODPUT}_j(A_j, m, s)}{\max_{m,s} \text{GOODPUT}_j(a_f, m, s)},$$

其中 GOODPUT_j 是作业 j 在其当前训练迭代中的 goodput，a_f 是作业的公平资源分配，定义为集群独占的 1/j 的共享。

4.3 Implementation

PolluxAgent 是作为一个 Python 库实现的，它被导入到 DL 训练代码中。其中集成了 PolluxAgent 和 PyTorch， PyTorch 使用 all-reduce 作为梯度同步算法。PolluxAgent 插入了性能分析代码，用于度量每次训练迭代所花费的时间以及计算梯度噪声尺度。

PolluxAgent 以一个固定的时间间隔将系统吞吐量模型拟合到目前为止收集的概要指标，并向 PolluxSched 报告拟合的系统吞吐量参数以及最新的梯度统计数据。在向 PolluxSched 报告后，PolluxAgent 通过使用当前分配的资源优化其最新的 goodput 函数来更新作业的每个 GPU 的批处理大小和梯度累加步骤。

PolluxSched 在 Kubernetes 中作为服务实现。在固定的时间间隔内，PolluxSched 运行其搜索算法，然后通过创建和终止运行 Job Workers 的 Kubernetes 容器来应用生成的分配矩阵。

为了找到一个好的分配矩阵，PolluxSched 使用了一种基于种群的搜索算法，该算法扰动并组合候选分配矩阵以生成更高值的分配矩阵，并最终修改它们以满足节点资源约束和干扰避免。具有最高适应度得分的分配矩阵应用于集

群中运行的作业。PolluxAgent 和 PolluxSched 都需要一个子过程，该子过程用于优化 GOODPUTt(a,m,s)。作者首先对总批量 M 的一系列候选值进行采样，然后找到最小的 s，根据用户定义的上限放入 GPU 内存，并最终采用产生最高 GOODPUT 值的配置。

5. Evaluation

作者使用带有 64 个 GPU 的测试台集群将 Pollux 与两个最先进的 DL 调度程序进行了比较。尽管 Pollux 的一个主要优势是自动为每个作业选择配置，但作者发现，即使参与对比的基准调度器提供了经过良好调优的作业配置，Pollux 仍将平均作业完成时间减少了 37–50%。

Pollux 能够根据当前集群状态和训练进度，在高吞吐量/低效率和低吞吐量/高效率训练模式之间进行权衡，从而动态调整每个作业。使用集群模拟器，作者评估了特定设置对 Pollux 的影响，包括总工作负载强度、事先驱动的探索、调度间隔和干扰避免。作者还展示了一个在云中自动扩缩的例子，基于 Pollux 的自动缩放器可以将大型模型（如 ImageNet）的训练成本降低 25%。

5.1 Experimental Setup

实验配置。作者使用由 16 个节点和 64 个 GPU 组成的集群进行实验。每个节点都是 AWS EC2 g4dn。12X 大型实例，带有 4 个 NVIDIA T4 GPU、48 个 VCPU、192GB 内存和一个 900GB SSD。作者在这个集群上部署了 Kubernetes 1.18.2，以及 CephFS 14.2.8 来存储检查点，以便重新启动检查点。

表 1 评估工作负载中使用的模型和数据集

Task	Dataset	Model	Optimizer	LR Scaler	M_0	Validation	Size	Frac. Jobs
Image Classification	ImageNet [12]	ResNet-50 [24]	SGD	AdaScale	200 imgs	75% top1 acc.	XL	2%
Object Detection	PASCAL-VOC [16]	YOLOv3 [55]	SGD	AdaScale	8 imgs	84% mAP	L	6%
Speech Recognition	CMU-ARCTIC [38]	DeepSpeech2 [3]	SGD	AdaScale	20 seqs	25% word err.	M	10%
Question Answering	SQuAD [54]	BERT (finetune) [14]	AdamW	Square-Root	12 seqs	88% F1 score	M	10%
Image Classification	Cifar10 [39]	ResNet18 [24]	SGD	AdaScale	128 imgs	94% top1 acc.	S	36%
Recommendation	MovieLens [23]	NeuMF [25]	Adam	Square-Root	256 pairs	69% hit rate	S	36%

构建负载。在微软发布的深度学习集群跟踪数据中，作者从最繁忙的 8 小时范围（3-10 小时）中随机抽取了 160 个 Job。原始跟踪数据中的每个作业都有关于其提交时间、GPU 数量和持续时间的信息。但是，没有提供有关正在训练的模型体系结构或数据集特征的信息。作者对每个作业进行了分类：小作业（0-1 GPU 小时）、中作业（1-10 GPU 小时）、大作业（10-100 GPU 小时）和 XLarge 作业（100-1000 GPU 小时）。对于跟踪中的每个作业，作者从表 1 中选择了一个属于同一类别的训练作业。

为基准调度器手动调优。作者手动调整了合成工作负载中每个作业的 GPU 数

量和批处理大小。作者的配置中，假设用户非常理性，并且对他们所训练的模型的可伸缩性非常了解。低于 50%的理想可伸缩性将导致资源利用率不足，超过80%的理想可伸缩性意味着作业仍然可以有效地利用更多的 GPU。

DL 调度器的比较。作者将 Pollux 与 2.3 中描述的两种近期的深度学习调度算法 Tiresias 和 Optimus 进行了比较。然而，Pollux 动态地适应 GPU 的数量和 DL 训练任务的 batch 大小，Optimus 只适应 GPU 的数量，而 Tiresias 两者都不适应。为了建立一个公平的对比方式，对于所有三个调度器，作者使用 AdaScale 来扩展 SGD 的学习率，使用平方根扩展规则来扩展 Adam 和 AdamW。对于每个作业，Tiresias 使用作者合成工作负载中指定的 GPU 数量和批处理大小。Optimus+Oracle 使用指定的批处理大小，但动态地决定 GPU 的数量。如果分配给作业的 GPU 太少，无法支持指定的批处理大小，则每个作业使用梯度累加。

5.2 Testbed Macrobenchmark Experiments

表 2 总结了作者在 7 种配置下的实验结果：

需要注意的是，首先，Pollux 与使用经过良好调优的作业配置的基线调度器进行了比较；其次，基线调度器使用真实的作业配置；第三，Pollux 使用两个可选值作为其公平性参数。

表 2 实验结果总结

Policy	Job Completion Time		Makespan
	Average	99%tile	
Pollux ($p=-1$)	**0.76h**	11h	**16h**
Optimus+Oracle+TunedJobs	1.5h	15h	20h
Tiresias+TunedJobs	1.2h	15h	24h
Optimus+Oracle	2.7h	22h	28h
Tiresias	2.8h	25h	31h
Pollux ($p=+1$)	0.83h	**10h**	**16h**
Pollux ($p=-10$)	0.84h	12h	18h

和优化过的作业配置进行比较。从表 2 中可以看到，即使 Optimus+Oracle 和 Tiresias 获得了 5.1 节中所述的优化作业配置，它们仍然显著落后于 Pollux。在此设置中，Pollux（p=−1） 与 Optimus+Oracle+TunedJobs 和 Tiresias+TunedJobs 相比，平均 JCT 缩短了 50%和 37%，尾（99th）延迟缩短了 27%和 27%，完成时间缩短了 20%和 33%。如前所述，此设置非常有利于基线调度器，基本上模仿拥有系统吞吐量、统计效率以及其值如何随资源分配和 batch 大小变化的专家知识的用户。

Pollux 的一个关键改进来源是其在训练期间权衡高吞吐量/低效率和低吞吐量/高效率模式的能力。图 5 显示了执行合成工作负载期间分配的 GPU 总数和平均效率。在集群争用较低的时期，Pollux 可以分配更多的 GPU（由（A）表

示），并使用更大的批量来提高训练吞吐量，即使是以较低的统计效率为代价，因为这样做会导致总体上更高的吞吐量。另一方面，在高集群争用期间，Pollux 可以使用较小的批大小来提高统计效率（由（B）表示）。

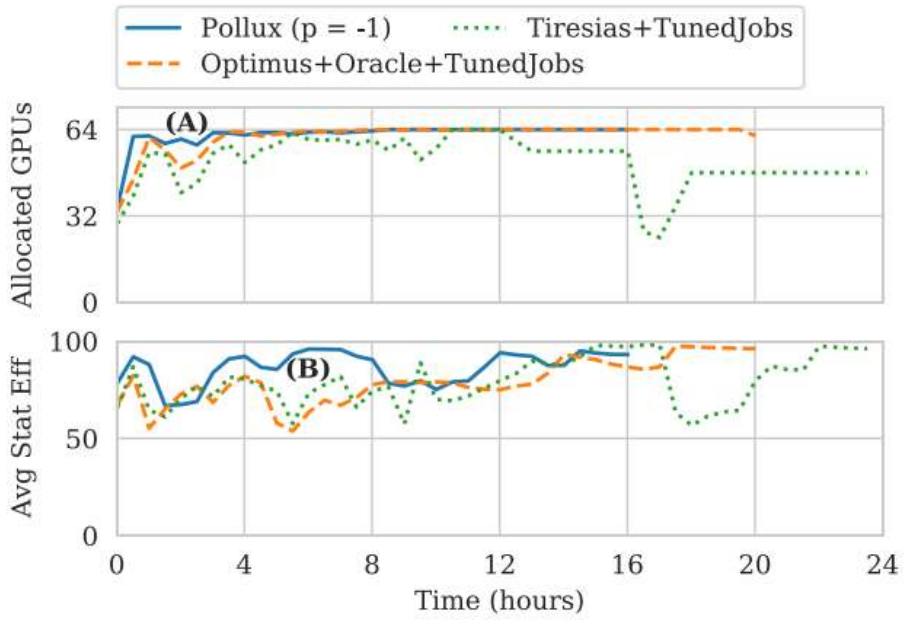


图 5 Pollux 与 Optimus 和 Tiresias 的比较

使用实际作业配置进行比较。这里作者找到了实际（大多数用户最初的配置）的一种作业配置，它和经过调优的配置相比，通常使用了更少的 GPU 和更小的 batch 大小。后续作者根据这个负载和配置进行了 Pollux 与 Baseline 的 DL 调度器的一些测试。最后作者发现在这个负载下，Pollux 相比其他 Baseline 有了大幅度的提升。

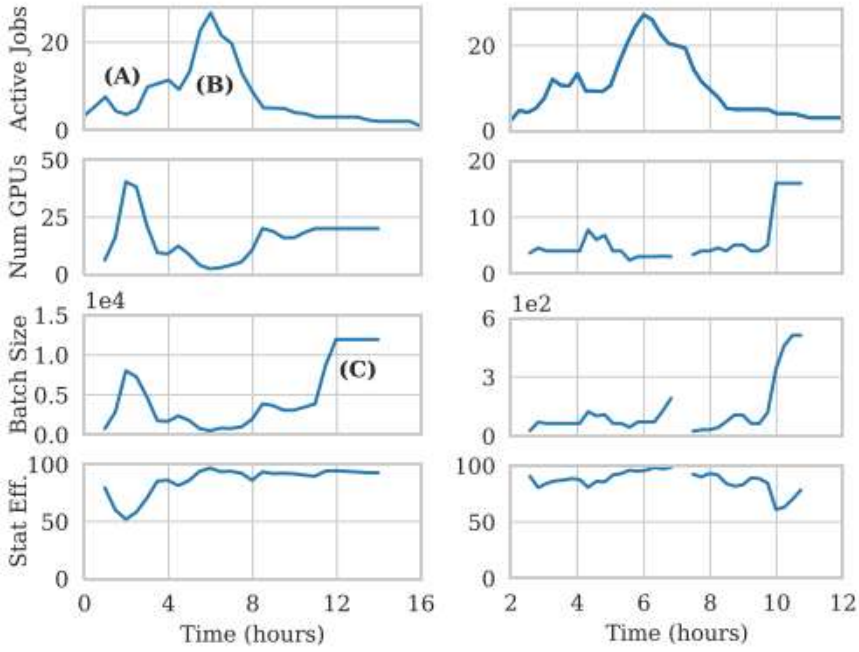


图 6 ImageNet 下观察共同自适应情况

对共同自适应的作业配置的深入探讨。图 6 左边展示了 Pollux 为一个 ImageNet 培训作业选择的配置。作者发现随着合成工作负载的压力增大，（A）初始阶段集群争用压力小，如果给 ImageNet 分配更多 GPU，导致更大的 batch 大小，会降低统计效率。（B）后续集群争用压力大，分配给 ImageNet 的 GPU 更少，导致 batch 大小较小，并提高统计效率。（C）集群争用恢复时，ImageNet 将继续分配更多 GPU 并使用更大的 batch 大小。

公平性参数造成的影响。作者使用公平性参数的三个值运行 Pollux，分别是 p=1、−1、−10。与不公平（p=1）相比，引入中等程度的公平（p=−1）虽然延长了平均作业完成时间（JCT），但降低了尾（99th）JCT。

系统开销的影响。平均来看，Pollux 对每个作业都每 7 分钟重新分配一次资源一次，导致由于检查点重新启动而产生平均 8%的运行时间开销。每 30 秒都会在其最新观察到的指标上拟合其吞吐量模型参数，每次平均耗时 0.2 秒。通过优化 GOODPUT_t 找到最佳的每个 GPU 的 batch 大小和梯度累积的步骤平均需要花费 0.4 毫秒。

5.3 Simulator Experiments

本节中作者构建了一个离散时间的模拟器，用来做负载重放。通过这个模拟器，作者对 Pollux 调度的公平性以及调度中的其他影响因素进行了评估。

调度公平性。作者将 Pollux 的完成时间公平性与 Optimus + Oracle + TunedJobs 的系统以及 Tirsias + TunedJobs 的系统进行比较。图 7 展示了测试结果。

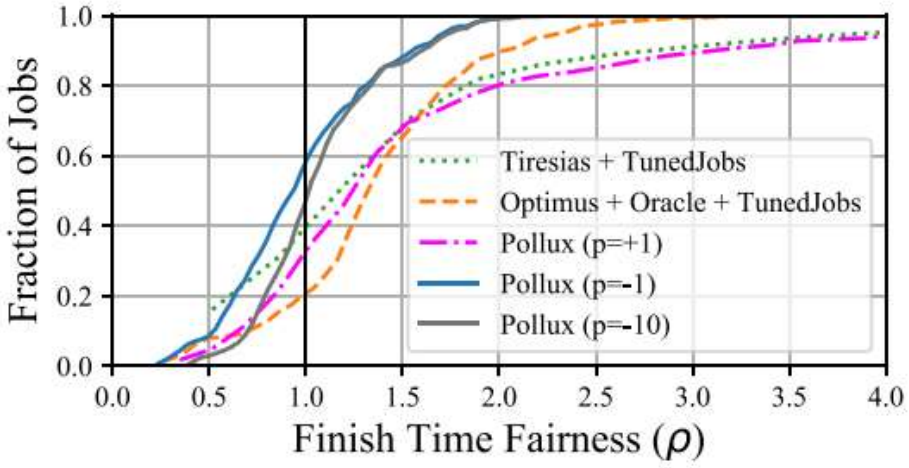


图 7 完成时间公平性的 CDF 曲线

调度中的其他影响因素。首先是对工作负载的敏感性，作者对比了 Pollux 和 Optimus + Oracle + TunedJobs 的系统以及 Tirsias + TunedJobs 的系统在负载不断加压的情况下的表现，如图 8(a)所示。和作者预期相同，三个调度策略都会

有不同程度的性能下降，但在所有负载下，Pollux 的表现都相对更好一些。

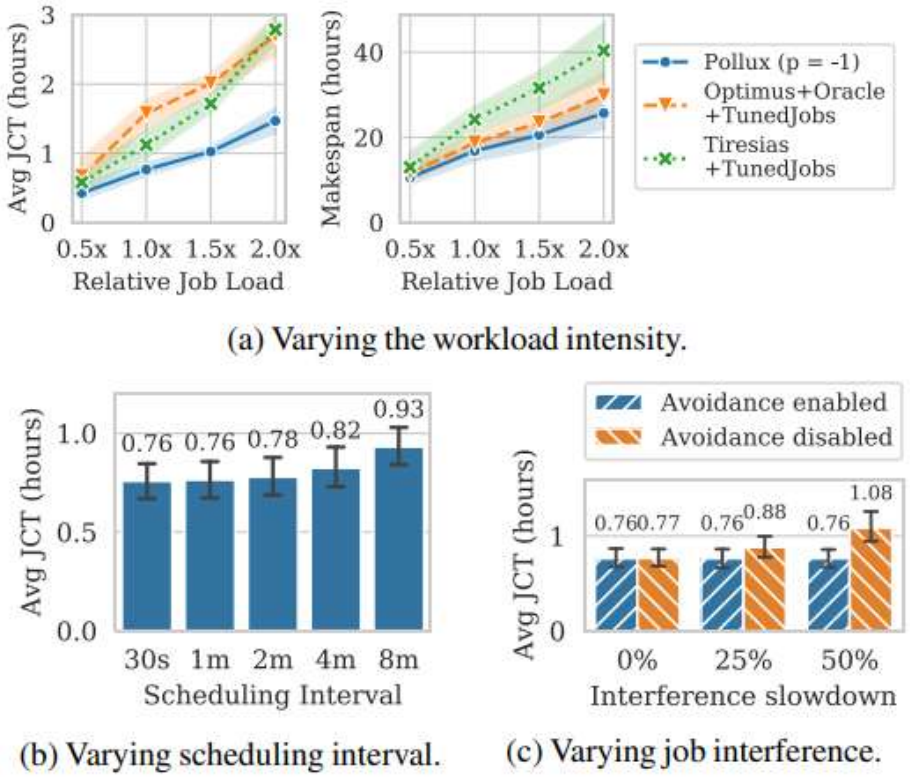


图 8 各种参数对 Pollux 的影响

然后是对历史数据的运用。实验表明 Pollux 在使用离线收集的历史数据后能够带来 JCT 的进一步提升。

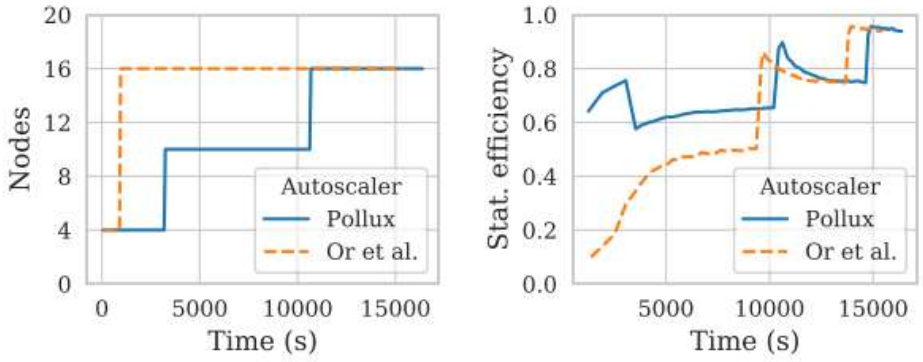
还有调度间隔的影响。图 8(b)展示了不同调度间隔下 Pollux 的平均 JCT，可以看到 Pollux 在 2 分钟的平均 JCT 方面表现得很好，但更长的时间间隔会导致性能下降。作者还发现发现排队导致了大约一半的性能下降，这表明 Pollux 仍然受益于相对频繁的资源分配调整。

最后是启用干扰避免造成的影响。作者人为注入了各种程度的干扰来测试，测试结果如图 8(c)所示。结果表明干扰造成的减速越明显，启用干扰避免带来的收益就越高。另一方面，在 0%没有干扰的情况下，开启干扰避免的 PolluxSched 的性能与不开启几乎没有差别，说明 PolluxSched 仍然能够在遵循干扰避免约束时找到有效的集群分配。

5.4 More Applications of Pollux

本节中作者主要介绍了 Pollux 更多的使用场景。

首先是**云环境下的自动扩缩容**。Pollux 可以用于云环境下 DL 模型的训练，作者以 ImageNet 的训练为例说明了 Pollux 在云环境下自动扩缩容场景的可用性。如图 9 所示。



(a) Number of nodes over time. (b) Statistical efficiency over time.

图 9 Pollux 和基于吞吐的自动扩缩容方案的对比

然后是**超参数的调优（HPO）**。表 3 显示了使用一种流行的基于贝叶斯优化的 HPO 算法（称为树结构 Parzen 估计器（TPE）），对在 CIFAR10 数据集上训练的 ResNet18 模型进行调优的结果。经过测试，Pollux 获得了和 Baseline 类似的 Accuracy 值，而且由于随着试验的进展和自适应 batch 大小、自适应（重新）分配资源等功能，使得 Pollux 完成 HPO 的速度比 Baseline 快了 30%。

表 3 HPO 实验汇总

Policy	Accuracy (Top 5 trials)	Avg JCT	Makespan
Pollux	95.4±0.2	25min	10h
Baseline	95.5±0.3	34min	14h

6. Additional Related Work

对于 DL 调度器的先前的研究，已经在 2.3 节中讨论过了，主要是下面两点内容。

自适应 batch 大小的训练。最近的 DL 训练算法的工作探索了动态调整批量尺寸，以获得更好的效率和并行化。Adabatch 在训练期间将 batch 大小增加在预定的迭代中，同时线性缩放学习率。CABS 在使用与 Pollux 的类似渐变统计数据中自适应地调整 batch 大小和学习率。Pollux 通过调整 batch 大小和学习率与当前可用的资源量相结合，符合现有的自适应批量策略。Kungfu 通过允许程序来定义自适应策略并在训练期间进行监控。同时它还支持自适应训练算法和自适应 batch 大小。作者认为 Kungfu 启发了可用于实现 PolluxAgent 所使用的自适应策略。

超参的调优。作者认为以往的大量工作集中于调整 ML 和 DL 模型的超参数，如前所述，这通常涉及大量训练过程。虽然 batch 大小和学习率等超参都在这些系统经常优化的超参数范围内，但 Pollux 的目标是根本不同的。HPO 算法搜索最高的模型质量，而 Pollux 则在不降低模型质量的情况下，调整 batch 大小和学

习率，以实现每个作业的最高效执行。

7. Conclusion

Pollux 是一个 DL 群集调度器，它可以自适应地分配资源，同时调整每个培训作业以最佳利用这些资源。作者提出了一个 goodput 公式，它结合了分布式 DL 训练的系统吞吐量 Throughput 和统计效率 Efficiency。基于 goodput 最大化原则，Pollux 自动调整 DL 作业的资源分配、batch 大小和学习率。即使有不错的配置，Pollux 也比最近的 DL 调度器表现更好、调度更均衡。

8. Acknowledgements

作者对其导师、审稿人和同事等人表示感谢。

二、 论文内容分析

1. Brief summary

1.1 What is the problem the paper is trying to solve?

当有多个深度学习任务共享一个集群时，目前大多数的调度器都需要用户手动给任务分配资源，这样会导致资源利用率低的问题。或者自动为job分配资源，但却没有考虑深度学习任务的参数能否适应新的资源环境的问题。作者提出Pollux正是为了解决多个DL任务的调度下这两个当前的核心问题。

1.2 What are the key ideas of the paper? Key insights?

- **主要思想**
  1) 将集群的goodput代替throughput作为参考指标。
  2) 对DL任务的参数配置的动态优化。
- **关键点**
  1) 设计集群goodput的定义并建模。
  2) 动态调度资源时加入fairness的影响。

1.3 What is the key contribution to literature at the time it was written?

- **主要贡献**
  1) 提出了一种名为Goodput的新指标，将系统吞吐量与统计效率相结合。
  2) 设计并实现了协同自适应的调度器Pollux，实验表明，相比于SOTA深度学习调度程序，Pollux 将平均作业完成时间减少了37-50%，并为每个作业提供了理想的资源和训练配置。

1.4 What are the most important things you take out from it?

系统的吞吐可能并不是我们追求的终极目标，在有限的成本下，系统的软硬件资源规模是受到成本的限制的。在这个前提下，为了提升系统吞吐而一味提升软硬件规模会导致成本的暴增。这时候吞吐作为指标不如把吞吐和统计效率相结合，考虑系统性能的同时也考虑了系统资源利用率。

2. Strength

2.1 Does the paper solve the problem well?

对于第一个问题，作者通过定义一个新的度量指标 Goodput，并对它进行了完善的建模，让其新设计的调度器 Pollux 能够很好地在调度过程中考虑到系统资源的利用率情况。

对于第二个问题，作者设计的 Pollux 具备了在 DL 任务执行时动态调优 DL 任务的参数的功能，相比当前的调度器，在减少任务执行时长上也取得了不小的提升。

3. Weaknesses

根据作者的描述，Pollux 的工作在各方面的表现都不错，可以全面超越现有的调度器。但我认为 Pollux 还存在对历史经验使用效率不高的问题。

根据文章中的测试结果，Pollux 在短作业的调度下，仅给短作业任务带来了 JCT 的 2%到 5%的下降，对长作业任务的 JCT 的影响基本没有。Pollux 当前默认是在训练期间从零开始探索每个深度学习任务的 GPU 分配，对历史数据的挖掘和利用还不够。

4. Can you do (much) better?

我认为可以在 Pollux 的基础上，集成一个历史经验池，使得 Pollux 在遇到相似的调度任务时能够更快速的给出调度的方案，减少调度开销。

5. What have you learned/enjoyed/disliked in the paper? Why?

我比较喜欢这篇文章的思路和行文，一方面这篇文章能够提出一个新的度量标准并在受限的篇幅里把它讲清楚，已经是一个了不起的事情。其次，这篇文章在行文时把 Pollux 的测试部分讲得很详细，能够把测试的动机和细节条件描述得比较完整。

三、 论文实验评价部分图表说明

1. Fig 5

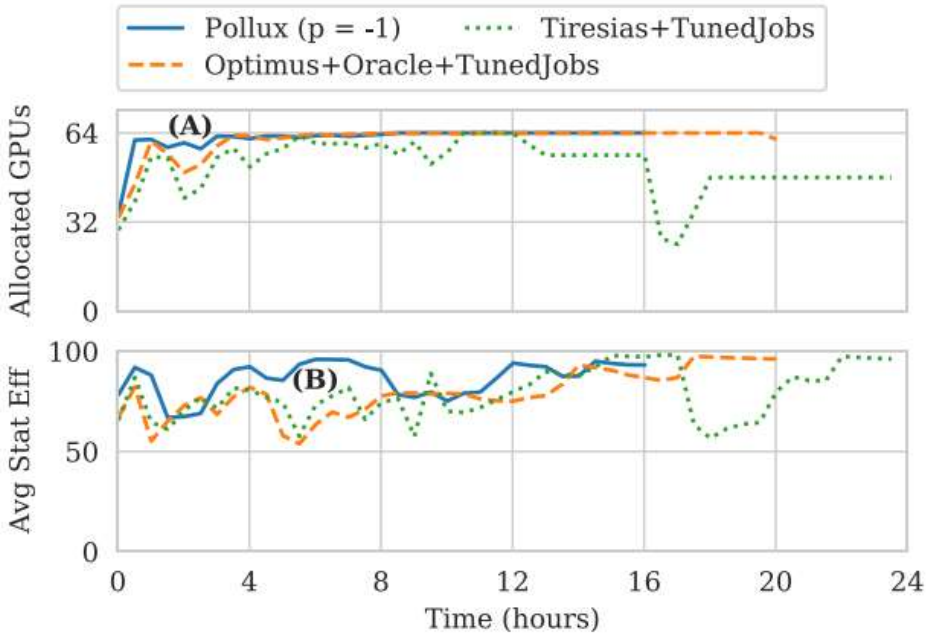


Figure 5: Comparison between Pollux ($p = -1$), Optimus, and Tiresias while executing our synthetic workload (with tuned jobs). TOP: average cluster-wide allocated GPUs over time. BOTTOM: average cluster-wide statistical efficiency over time. Tiresias+TunedJobs dips between hours 16 and 20 due to a 24-GPU job blocking a 48-GPU job from running.

1.1 本实验的目标

希望说明 Pollux 相比其他 Baseline 的调度方案，能够拥有更灵活的调度和更高的资源利用率。

1.2 本实验的设计思想

对比 Pollux、Tiresias+TunesJobs 以及 Optimus+Oracle+TunedJobs 调度下的集群的 GPU 使用率和平均统计效率。

1.3 本实验的具体配置

（1） 环境配置

Task	Dataset	Model	Optimizer	LR Scaler	M_0	Validation	Size	Frac. Jobs
Image Classification	ImageNet [12]	ResNet-50 [24]	SGD	AdaScale	200 imgs	75% top1 acc.	XL	2%
Object Detection	PASCAL-VOC [16]	YOLOv3 [55]	SGD	AdaScale	8 imgs	84% mAP	L	6%
Speech Recognition	CMU-ARCTIC [38]	DeepSpeech2 [3]	SGD	AdaScale	20 seqs	25% word err.	M	10%
Question Answering	SQuAD [54]	BERT (finetune) [14]	AdamW	Square-Root	12 seqs	88% F1 score	M	10%
Image Classification	Cifar10 [39]	ResNet18 [24]	SGD	AdaScale	128 imgs	94% top1 acc.	S	36%
Recommendation	MovieLens [23]	NeuMF [25]	Adam	Square-Root	256 pairs	69% hit rate	S	36%

Table 1: Models and datasets used in our evaluation workload. Each training task achieves the provided validation metrics. The fraction of jobs from each category are chosen according to the public Microsoft cluster traces.

（2） 实验参数

Policy	Job Completion Time		Makespan
	Average	99%tile	
Pollux ($p=-1$)	**0.76h**	11h	**16h**
Optimus+Oracle+TunedJobs	1.5h	15h	20h
Tiresias+TunedJobs	1.2h	15h	24h
Optimus+Oracle	2.7h	22h	28h
Tiresias	2.8h	25h	31h
Pollux ($p=+1$)	0.83h	**10h**	**16h**
Pollux ($p=-10$)	0.84h	12h	18h

Table 2: Summary of testbed experiments.

1.4 本实验的图中各种标记的含义

横坐标：实验持续时间(hours)。

纵坐标：

（1）Allocated GPU，分配的 GPU 个数（个）；

（2）Avg Stat Eff，平均统计效率（%）。

图注：分别是 Pollux、Tiresias+TunesJobs 以及 Optimus+Oracle+TunedJobs 三种不同的调度方案。

1.5 实验结论

相比于其他调度方案，Pollux 能够提高资源利用率和系统的统计效率。

1.6 实验解释

图中的（A）处，显示了 Pollux 在集群初期任务较为空闲时能给 DL 任务分配

更多的 GPU，提高资源的利用率。

图中的（B）处，显示了 Pollux 在集群任务压力较大时，也能够通过动态调整 DL 任务的参数，使用更小的 batch 大小，从而提高系统的统计效率。

2. Fig 6

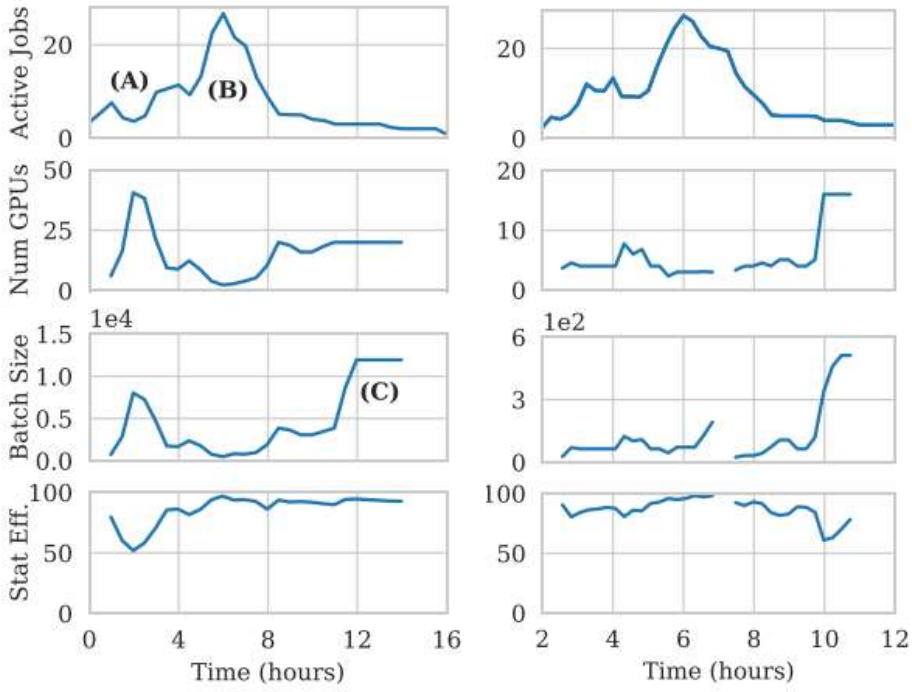


Figure 6: Co-adaptation over time of one ImageNet job (LEFT) and two YOLOv3 jobs (RIGHT) using Pollux ($p = -1$). ROW 1: number of jobs actively sharing the cluster. ROW 2: number of GPUs allocated to the job. ROW 3: batch size (images) used. ROW 4: statistical efficiency (%).

2.1 本实验的目标

说明 Pollux 共同自适应 DL 任务参数的能力，即在调度过程中动态调优任务参数的能力。

2.2 本实验的设计思想

在 ImageNet 任务和 YOLOv3 任务下，通过一段时间内任务数量、任务使用的 GPU 数量、任务 batch 大小、系统统计效率四个维度的对比。通过对比任务的不同时段 Pollux 产生的不同操作产生的不同效果来说明问题。

2.3 本实验的具体配置

（1） 环境配置

Task	Dataset	Model	Optimizer	LR Scaler	M_0	Validation	Size	Frac. Jobs
Image Classification	ImageNet [12]	ResNet-50 [24]	SGD	AdaScale	200 imgs	75% top1 acc.	XL	2%
Object Detection	PASCAL-VOC [16]	YOLOv3 [55]	SGD	AdaScale	8 imgs	84% mAP	L	6%
Speech Recognition	CMU-ARCTIC [38]	DeepSpeech2 [3]	SGD	AdaScale	20 seqs	25% word err.	M	10%
Question Answering	SQuAD [54]	BERT (finetune) [14]	AdamW	Square-Root	12 seqs	88% F1 score	M	10%
Image Classification	Cifar10 [39]	ResNet18 [24]	SGD	AdaScale	128 imgs	94% top1 acc.	S	36%
Recommendation	MovieLens [23]	NeuMF [25]	Adam	Square-Root	256 pairs	69% hit rate	S	36%

Table 1: Models and datasets used in our evaluation workload. Each training task achieves the provided validation metrics. The fraction of jobs from each category are chosen according to the public Microsoft cluster traces.

（2） 实验参数

Policy	Job Completion Time		Makespan
	Average	99%tile	
Pollux ($p=-1$)	**0.76h**	11h	**16h**
Optimus+Oracle+TunedJobs	1.5h	15h	20h
Tiresias+TunedJobs	1.2h	15h	24h
Optimus+Oracle	2.7h	22h	28h
Tiresias	2.8h	25h	31h
Pollux ($p=+1$)	0.83h	**10h**	**16h**
Pollux ($p=-10$)	0.84h	12h	18h

Table 2: Summary of testbed experiments.

2.4 本实验的图中各种标记的含义

横坐标：实验持续时间(hours)。

纵坐标：

（1）Active Jobs，运行的任务数（个）；

（2）Num GPUs，分配的 GPU 个数（个）；

（3）Batch size，任务的 batch 大小；

（4）Stat Eff，统计效率（%）。

2.5 实验结论

Pollux 具有共同自适应 DL 任务参数的能力，即在调度过程中动态调优任务参数的能力，且能够由此提升系统的统计效率。

2.6 实验解释

图中要点（A），说明了当集群刚启动时，任务数量并不多，有更多 CPU 被分

配给 ImageNet 任务，导致任务使用更大的 batch 大小，以及导致更低的统计效率。

图中要点（B），说明了集群压力增大时，Pollux 会分配更少的 CPU 给 ImageNet 任务，导致任务使用更小的 batch 大小，从而提高统计效率。

图中要点（C），说明了当集群压力回撤时，ImageNet 再一次被分配更多的 GPU，并使用更大的 batch 大小。但此时每个 GPU 的 batch 大小会明显高于开始空闲时的状态，因为任务此时处于最后的阶段，是训练过程中比较高统计效率的阶段。

图左右两个实验，在 ImageNet 和 YOLOv3 上展示了类似的趋势，说明了上述结果具有一定普遍性。

四、 学习感想

这篇文章给了我一个全新的看问题的角度，那就是度量标准并非一成不变的。文章从性能指标的根因开展分析，讨论了使用 Throughput 作为调度的唯一目标的不足之处，并提出了一种全新的度量方式，取得了不错的效果。此外，作者将分布式深度学习当中的建模和调度工作相结合，也为我在资源调度方向提供了新的思路：原来调度的过程不光可以分配硬件资源，还可以尝试去修改任务的内部参数配置，可以通过修改参数配置来使得任务与硬件资源更加匹配，从而使得整个集群的资源能够被更好地使用。

在计算机体系结构中，任务的调度也是一个重要的问题。随着摩尔定律机器的硬件性能不断提升，但终于在近几年逐渐出现了瓶颈，摩尔定律不再那么有效，说明硬件的发展没有那么简单。在不能无限制提升系统硬件资源规模的情况下，任务调度就显得尤为重要。作为一个处理外部请求、向外提供服务的系统而言，对任务的响应是至关重要的。我很难想象如果系统没有调度或是调度不合理会是什么情况。可能一个小任务分配了大量的资源，而大任务却只能使用贫瘠的资源，或可能紧急的任务得等到空闲任务处理完才能被处理，甚至永远在排队。这时，运用计算机系统结构的相关知识就能够得到答案，其重要性可见一斑。

五、 知识点及题目

考点：性能与成本的权衡

概念：

- CPI：每条指令的时钟周期数；MIPS：每秒执行的百万指令数
- 晶圆、晶片：每个晶圆上晶片数 = $\frac{\pi*(晶圆直径/2)^2}{晶片面积}-\frac{\pi*晶圆直径}{\sqrt{2*晶片面积}}$

评价指标：CPI、MIPS、成本

题目：假设跑一个基准测试程序，处理器的频率为 800MHz，基准测试程序的特征如下：

指令类型	频率(%)	时钟周期
ALU	40	1
Load / Store	30	2
Branch	20	3
FP	10	5

你正在考虑使用一个成本更低的处理器，计划是移除浮点运算硬件来减少芯片面积。假设生产该芯片的晶圆直径为 12cm，成本为$1500。

当前处理器的芯片大小为 9mm*9mm，新处理器的芯片大小为 6mm*6mm，由于没有浮点运算单元，新处理器上浮点运算的时钟周期比原处理器翻倍，为 10 个时钟周期。但能够有更大的热设计功耗余量使得新处理器的主频提高 25%。

请分别计算该基准测试程序在新旧处理器上的 CPI 和 MIPS 以及每个新处理器节约的生产成本。

解答：

$$CPI_{old} = 40\% * 1 + 30\% * 2 + 20\% * 3 + 10\% * 5 = 2.1$$

$$MIPS_{old} = \frac{800MHz}{2.1 * 10^6} = 380.95$$

$$N_{old} = \frac{\pi * \left(\frac{12}{2}\right)^2}{0.81} - \frac{\pi * 12}{\sqrt{2 * 0.81}} \approx 110，\ Cost_{old} = \frac{\$1500}{N_{old}} = \$13.64$$

$$CPI_{new} = 40\% * 1 + 30\% * 2 + 20\% * 3 + 10\% * 10 = 2.6$$

$$MIPS_{old} = \frac{800 * (1 + 25\%)MHz}{2.6 * 10^6} = 384.62$$

$$N_{new} = \frac{\pi * \left(\frac{12}{2}\right)^2}{0.36} - \frac{\pi * 12}{\sqrt{2 * 0.36}} = 269.73 \approx 269$$

$$Cost_{new} = \frac{\$1500}{N_{new}} = \$5.58$$

$$Cost_{save} = Cost_{old} - Cost_{new} = \$8.06$$