计算机科学
COMPUTER SCIENCE

填充性载荷:减少集群资源浪费与深度学习训练成本的负载

杜昱, 俞子舒, 彭晓晖, 徐志伟

引用本文

杜昱, 俞子舒, 彭晓晖, 徐志伟. 填充性载荷:减少集群资源浪费与深度学习训练成本的负载[J]. 计算机科学, 2024, 51(9): 71-79.

DU Yu, YU Zishu, PENG Xiaohui, XU Zhiwei. Padding Load:Load Reducing Cluster Resource Waste and Deep Learning Training Costs [J]. Computer Science, 2024, 51(9): 71-79.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.231000222

填充性载荷:减少集群资源浪费与深度学习训练成本的负载

杜 昱 俞子舒 彭晓晖 徐志伟
中国科学院计算技术研究所 北京 100190
中国科学院大学 北京 100049
(duyu19@mails.ucas.ac.cn)

摘 要 近年来,大模型在生物信息学、自然语言处理和计算机视觉等多个领域取得了显著成功。然而,这些模型在训练和推理阶段需要大量的计算资源,导致计算成本高昂。同时,计算集群中存在资源利用率低、任务调度难的供需失衡问题。为了解决这一问题,提出了填充性载荷的概念,即一种在计算集群中利用空闲资源进行计算的负载。填充性载荷的计算资源随时可能被其他负载抢占,但其使用的资源优先级较低,资源成本也相对较低。为此,设计了适用于填充性载荷的分布式深度学习训练框架 PaddingTorch。基于阿里巴巴 PAI 集群的数据,使用 4 块 GPU 模拟了任务切换最频繁的 4 个 GPU 时间段上的作业调度情况,使用 PaddingTorch 将蛋白质复合物预测程序作为填充性载荷进行训练。训练时长为独占资源时训练时长的 2.8 倍,但训练成本降低了 84%,在填充性载荷填充时间段内 GPU 资源利用率提升了 25.8%。

关键词:深度学习;分布式训练;资源利用率;计算集群;编程框架
中图分类号 TP312

Padding Load:Load Reducing Cluster Resource Waste and Deep Learning Training Costs

DU Yu,YU Zishu,PENG Xiaohui and XU Zhiwei
Institute of Computing Technology,Chinese Academy of Sciences,Beijing 100190,China
University of Chinese Academy of Sciences,Beijing 100049,China

Abstract In recent years,large-scale models have achieved remarkable success in multiple domains such as bioinformatics,natural language processing,and computer vision. However,these models often require substantial computational resources during the training and inference stages,resulting in considerable computational costs. Additionally,computing clusters experience imbalances between supply and demand,manifesting as low resource utilization and difficulties in task scheduling. To address this problem,the concept of Padding Load is introduced,which leverages idle computing resources for computational tasks. Resources allocated to Padding Load can be preempted by other tasks at any time. However,they operate with a lower resource priority,leading to relatively lower costs. PaddingTorch is a distributed deep learning training framework tailored for Padding Load. Utilizing data from the Alibaba PAI cluster,job scheduling is simulated on four GPUs,specifically during peak task-switching intervals. PaddingTorch is employed to train a protein complex prediction model using the Padding Load approach. While the training duration is 2.8 times that of exclusive resource usage,there is an 84% reduction in training costs and a 25.8% increase in GPU resource utilization during the periods when Padding Load is employed.

Keywords Deep learning,Distributed training,Resource utilization,Computing cluster,Programming framework

1 引言

深度学习已逐步从实验室走向各行业的实际应用,重塑传统行业模式,颠覆现有方法,并引领未来的发展方向。数据集规模的增长为深度学习领域带来了机遇与挑战。一方面,更多的数据样本会使模型得到更充分的训练;另一方面,训练过程所需的计算资源也显著增加。以 AlphaFold2 为例,其完整的训练过程需使用 128 张 A100 显卡,耗时约 11 天[1]。值得注意的是,AlphaFold2 使用的 Uniprot 数据库在过去十年中呈指数增长,2022_03 版本中包括超过 2.27 亿条序列记录[2]。不断增加的模型复杂度和数据集规模导致模型需要更长时间的训练过程,消耗了大量的计算资源。高昂的成本在一定程度上阻碍了技术进步,使得部分研究方向成为少数团体的专属领域[3]。因此,如何降低训练成本成为一个重要课题。

到稿日期:2023-10-31 返修日期:2024-04-18
基金项目:北京市自然科学基金(4212027);国家自然科学基金(62072434)
This work was supported by the Natural Science Foundation of Beijing,China(4212027) and National Natural Science Foundation of China(62072434).
通信作者:俞子舒(yuzishu19s@ict.ac.cn)

深度学习工作负载的增加也为大型计算集群带来了挑战。现有计算集群的资源利用率较低，仅约为 25%～50%[4]，大量资源长时间处于空闲状态，同时许多任务又具有较高的排队延迟。资源的供需失衡既造成集群资源过剩，又导致需求无法完全满足。

为了解决训练成本高与计算集群资源供需失衡的问题，本文提出了填充性载荷的概念，并设计了适用于填充性载荷的分布式深度学习训练框架 PaddingTorch。填充性载荷是一种在计算集群中动态分配和执行的负载。该类负载是系统中的最低优先级负载，仅利用系统中的空闲资源，且使用的资源随时可能被集群中的其他负载抢占。应用本文框架 PaddingTorch 将深度学习训练作业转化为填充性载荷，可以降低研究者的训练成本，对于计算服务提供者而言，则能提高系统资源利用率并减少资源闲置导致的浪费。

本文具体贡献为：1)定义了填充性载荷，并对填充性载荷的资源使用时长、成本等进行了理论分析；2)实现了适用于填充性载荷的分布式深度学习训练框架 PaddingTorch，可以低成本地处理工作节点的启动和退出；3)将蛋白质复合物预测程序作为填充性载荷，使用 4 块 GPU 模拟 PAI 集群中任务切换频率最高的 4 个 GPU 时间段上的作业调度情况，使用其空闲资源进行训练，训练时间约为使用独占资源训练时间的 2.8 倍，成本则降低约 84%，被填充时段的 GPU 资源利用率提升了 25.8%。

2 研究背景与问题

2.1 集群资源使用情况

1)GPU 利用率

随着 GPU 性能的提升，有效利用 GPU 性能成为一个重要课题。许多任务由于受复杂性和规模限制，无法充分利用 GPU 的计算能力[5]。GPU 中包括多个流式多处理器(Streaming Multiprocessor，SM)，在线程块数量不足以填满所有 SM 等情况下，部分 SM 会处于空闲状态，没有被利用[6]。NVIDIA 将 GPU 的利用率定义为采样期间一个或多个内核在 GPU 上执行的时间百分比，持续时间在 1/6～1s 之间，可以简单地通过 nvidia-smi 获取相关的统计数据[7]。这种定义并不能准确反映 SM 的利用率，但仍可用于估计 GPU 的使用情况[8]。阿里巴巴 PAI 集群通过时间复用进行 GPU 共享，在该集群中，GPU 使用数量指 GPU 使用时长百分比[5]。

Hu 等[9]使用 GPU 分配率来表示 GPU 利用率，即处于活动状态的 GPU 占 GPU 总量的比例。本文使用的 GPU 利用率指在统计时间内 GPU 处于工作状态的时间百分比。

2)系统资源利用率

在大型集群中，GPU 资源的利用率为 25%～50%，且活动 GPU 大约占总 GPU 的 65%～90%[4,9]。许多集群的 GPU 分配率可以达到 85%～90%，但 GPU 分配率过高会导致部分待处理任务开始堆积，尽管此时仍有数百个 GPU 未被使用[4]。Lyra 使用的集群[10]中，GPU 分配率为 82%，部分 GPU 经常处于空闲状态，平均排队时间却超过 3000s。由于一些任务需要所有资源同时就位，并进行组调度(Gang-scheduling)，因此集群中会保留部分可用资源，直到满足组调度的资源需求[11]。组调度需要的 GPU 数越多，集群因等待造成的资源浪费越大。

集群中面临棘手的调度问题，尽管 GPU 利用率和分配率仍有提升空间，但进一步提高 GPU 分配率可能会导致更高的排队延迟，降低用户体验。填充性载荷具有可以随时被高优先级资源抢占的灵活性，将空闲资源分配给填充性载荷，常规负载排队延迟平均仅增加 0.1%。同时，通过将部分需要进行组调度的负载转化为能够利用动态资源的填充性载荷，不仅降低了调度复杂性，还减少了集群在组调度过程中等待所有节点就绪时产生的资源浪费。

2.2 云服务付费方式

常见的云服务提供商有亚马逊云[12]、阿里云[13]等，这些云厂商提供的云服务器包括按量付费、包年包月和竞价型实例 3 种付费方式。按量付费方式适用于短期购买，价格最高；包年包月方式长期持有资源，价格约为按量付费方式的 80%；竞价型实例可用资源和价格根据市场供需关系波动，价格约为按量付费方式的 10%～20%[12,14]。当资源库存不足或市场价格高于用户出价时，用户资源会被回收。资源回收时，用户会在提前 30s 至 5min 不等的时间收到中断回收预警，但用户数据并不会自动保存[13]。用户需自行处理任务的关闭、转移和重新部署，确保运行程序具有良好的容错功能。

与填充性载荷使用的最低优先级资源相比，现有云计算系统中竞价型实例的可用时间较长。例如，阿里云、亚马逊云为用户设置若干小时保护期，保护期内资源不会被回收；阿里云的竞价型实例在一小时保护期限过后，系统每五分钟会比较用户价格与市场价格，判断是否释放实例。填充性载荷可以使用更加细粒度的空闲计算资源，如仅有数十秒空闲时长的资源。本文假设填充性载荷使用资源的价格低于竞价型实例，即低于按量付费方式的 10%。

2.3 分布式深度学习框架

在分布式深度学习中，需要考虑参数同步和节点间通信等问题。由于集合通信需要一些学习成本，其他领域的研究者可能并不想成为通信原语方面的专家，他们更愿意将时间和精力放在研究工作中[15]。分布式深度学习框架需要为研究者提供简单易用的 API，并确保框架对代码的侵入性较低，只需对单节点训练代码进行少量修改即可实现多节点训练。

被广泛使用的分布式深度学习平台，如 PyTorch，TensorFlow 等，对分布式训练的支持有限，并未对研究者隐藏所有通信和同步操作等细节。深度学习训练框架 Horovod[15]，PyTorch Lightning[16]等在深度学习平台的基础上提供了一些高级接口，并提供了更为全面的弹性容错功能以支持使用竞价型实例，但这些框架对弹性容错功能的支持仍然有限。Horovod 通过定期在内存中备份状态来进行容错。用户需要指定若干轮次或若干批次进行状态备份，难以确定合适的备份频率。备份频率较高时，带来的额外开销增加；备份频率

较低时，出错后损失的训练进度较多。PyTorch Lightning 除周期性备份检查点以外，还可以采用精准检查点保存的方法，捕获到中止信号时保存检查点，出现故障时从上一个保存的检查点恢复。但节点故障时同样需要重启所有节点，且从检查点恢复训练进度时不支持改变节点数。

本文实现了 PaddingTorch，该框架不需要从检查点恢复，由主节点负责处理失败的任务；失败的节点不会对其他节点产生影响，不需进行重启。当节点资源再次可用时，可以直接加入原有的通信组。

2.4　本文的研究问题

为了解决大模型训练成本高、大型计算集群资源利用率低的问题，本文提出了填充性载荷的概念。填充性载荷利用系统空闲资源降低执行成本，使用的资源数量可以动态变化，随时可能被高优先级负载抢占计算资源。

现有的分布式深度学习训练框架在弹性容错方面并不能满足填充性载荷的需求。填充性载荷使用的低优先级资源可能在分钟级别被频繁中止与重启，这意味着单一机器上的可执行时间最短为若干分钟[17]，现有框架需要在分钟级别更频繁地保存检查点，否则丢失的训练进度会使整个训练过程难以持续。为满足填充性载荷的需求，本文实现了 PaddingTorch，该框架具有更低成本的容错功能以及更健壮的通信组。

3　填充性载荷

3.1　定义

填充性载荷是一种在计算集群中动态分配和执行的负载，旨在充分利用空闲资源，提高系统资源的利用率并降低负载的执行成本。

定义 1　任务(Task)是一个计算请求，用三元组表示为：

$$T=\langle D,C,S(c,g,m,t)\rangle \tag{1}$$

其中，D 是输入数据；C 是需要运行的代码；S 是该计算请求的资源需求，包括 c 个 CPU 核、g 个 GPU 处理器、m GB 的内存，以及型号为 t 的 GPU 处理器。任务可以在任意满足资源需求的一个计算节点上执行。

定义 2　任务发起时间(s_i)指用户将任务 i 提交到计算集群的时间。

定义 3　任务执行时间需求(e_i)指若节点不会被抢占，则任务 i 在节点上从开始计算到计算完成的时间。

定义 4　任务结束时间(f_i)指任务 i 被完全完成，交付给用户的时间。

定义 5　资源使用时长($\delta_{j,q}$)指节点 j 第 q 次开始计算来自同一作业的任务，直至所有任务均已完成，或该节点被来自其他作业的更高优先级任务抢占的时长。

定义 6　填充性载荷(Padding Load)是一种由 N 个任务组成的负载，对计算节点的需求量范围为$[S_{\min},S_{\max}]$。$S_{\max}$ 为最大可并行度，该值通常较大，数量级可为数百至数千；$S_{\min}$ 为最小可并行度，指负载执行需要的最少资源数。当任务之间有同步操作等约束时，填充性载荷中的任务存在依赖关系。填充性载荷的执行时间为 P_{exec}，则

$$P_{\text{exec}}=\max(f_i)-\min(s_i) \tag{2}$$

理想情况下，任务在最大可并行度下的执行时间为 P_{ideal}，故

$$P_{\text{ideal}}=\frac{\sum_{i=1}^{N}e_i}{S_{\max}} \tag{3}$$

即为

$$P_{\text{exec}}=k\times P_{\text{ideal}} \tag{4}$$

其中，$k\geqslant 1$，k 为资源可被抢占场景下执行时间与资源独占场景下执行时间的比值。在节点 j 上执行任务 i，除任务计算外的其他时间开销为 $c_{i,j}$，该时间开销用于通信等操作。$\varphi_{j,q}$ 为节点启动与退出的总时间开销。任务有效的必要条件为：

$$\delta_{j,q}>e_i+c_{i,j}+\varphi_{j,q} \tag{5}$$

满足该条件的 $\delta_{j,q}$ 为任务 i 可用资源的最小粒度。

图 1 给出了填充性载荷的结构。填充性载荷中的任务可以分为两类，一类为任务管理器，任务管理器是高优先级任务，使用的资源不会被其他负载抢占；另一类为普通任务，使用的资源随时可能被其他任务抢占。任务开始时间 s_i 等于作业提交时间，任务结束时间 f_i 等于作业完成时间。任务管理器始终存在，其数量并不计入并行度 S。

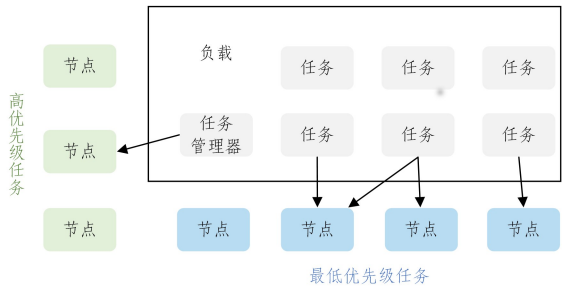


图 1　填充性载荷结构图

Fig. 1　Structure of padding load

适用于填充性载荷的负载持续时间较长，可以达到数百小时，在执行过程中增加一些时间对用户来说是可容忍的。然而，高时延敏感性的负载，如实时数据处理和智能驾驶系统的突发识别任务，以及整体执行时间较短的负载并不适合作为填充性载荷。这些任务的交付时间一旦延迟，会对用户体验造成较大影响。

填充性载荷应具有以下性质：

1)正确性。填充性载荷不应给模型精度带来负面影响，导致模型收敛速度变慢或不收敛等情况。

2)可伸缩性。填充性载荷使用的资源是动态变化的，随时可能有节点加入或退出，需要保证节点可以随意加入或退出通信组且开销较小，不影响其他节点，无需重建通信组。

3)容错代价小。填充性载荷使用的资源随时可能被抢占，为了确保单一节点出故障不会干扰其他节点的训练过程，组成填充性载荷的任务应具有以下性质：

(1)原子性。任务只能成功或失败。

(2)幂等性。任务多次执行与单次执行的效果相同。

3.2 成本分析

本节从理论上对填充性载荷的资源使用时长和价格成本进行分析。

对于资源 j 上的任务，资源 j 第 q 次被分配给该载荷，资源使用时长为：

$$\delta_{j,q}=\sum_{i=1}^{r_{j,q}}(e_{m_i}+c_{m_i})+\varepsilon_{j,q}+\varphi_{j,q} \tag{6}$$

其中，$0\leqslant\varepsilon_{j,q}<e_{m_{r+1}}+c_{m_{r+1}}$，$r_{j,q}\geqslant 0$，即任务 $m_{r_{j,q}+1}$ 在资源 j 上未完成，资源被高优先级任务抢占，$\varepsilon_{j,q}$ 是节点被抢占导致的额外开销。资源 j 第 q 次被分配给该填充性载荷时完成了 $r_{j,q}$ 个任务的计算，任务的成功率为 $\frac{r_{j,q}}{r_{j,q}+1}$，资源可用时长 $\delta_{j,q}$ 的有效比例为：

$$\theta_{j,q}=\frac{\sum_{i=1}^{r_{j,q}}e_{m_i}}{\sum_{i=1}^{r_{j,q}}(e_{m_i}+c_{m_i})+\varepsilon_{j,q}+\varphi_{j,q}} \tag{7}$$

对于同种硬件型号的资源，作为普通资源时价格为 R_s，作为随时可被高优先级任务抢占的资源时价格为 R_p。使用普通资源进行计算，资源使用总时长为 $\sum_{i=1}^{N}e_i$；使用填充性载荷时，资源使用总时长为：

$$\sum_{j=1}^{M}\sum_{q=1}^{Q_j}\delta_{j,q}=\sum_{j=1}^{M}\sum_{q=1}^{Q_j}\left(\sum_{i=1}^{r_{j,q}}(e_{m_i}+c_{m_i})+\varepsilon_{j,q}+\varphi_{j,q}\right) \tag{8}$$

资源使用总时长与资源价格的乘积即为该载荷的总成本。故使用填充性载荷与否的价格比值为：

$$\rho=\frac{R_p\times\sum_{j=1}^{M}\sum_{q=1}^{Q_j}\left(\sum_{i=1}^{r_{j,q}}(e_{m_i}+c_{m_i})+\varepsilon_{j,q}+\varphi_{j,q}\right)}{R_s\times\sum_{i=1}^{N}e_i} \tag{9}$$

α 为填充性载荷资源价格与普通资源价格的比值，使用 θ_j 可以表示为：

$$\rho=\frac{R_p\times\sum_{j=1}^{M}\sum_{q=1}^{Q_j}\delta_{j,q}}{R_s\times\sum_{j=1}^{M}(\delta_{j,q}\times\theta_{j,q})}=\alpha\times\frac{\sum_{j=1}^{M}\sum_{q=1}^{Q_j}\delta_{j,q}}{\sum_{j=1}^{M}\sum_{q=1}^{Q_j}(\delta_{j,q}\times\theta_{j,q})} \tag{10}$$

抢占式实例资源的价格约为按需实例价格的 1/10[11]，填充性载荷使用的最低优先级资源比抢占式实例的资源更加碎片化，本文假设其价格低于按需实例价格的 1/10，即 $\alpha=\frac{R_p}{R_s}<\frac{1}{10}$。因此，作为填充性载荷的负载应至少确保所有资源有效时长比例的加权平均值 $\bar{\theta}>10\%$。同时，$\frac{e_{m_i}}{e_{m_i}+c_{m_i}}$ 越大，该负载作为填充性载荷的成本越低。

3.3 实验负载

1）蛋白质复合物预测。本文使用的蛋白质复合物三维结构预测模型与数据集由分子之心[1)]提供，使用的训练框架为 PaddingTorch。如表 2 所列，在该作业中，一个小批量的平均计算时间为 10.9 s，计算完成后的平均通信时间为 2.8 s，工作节点启动与退出的总时间为 7 s，即 $\tilde{e}$ 为 10.9 s，$\tilde{c}$ 为 2.8 s，$\tilde{\varphi}$ 为 7 s，计算可得 $31.7\%<\theta_{j,q}<79.6\%$，蛋白质复合物预测作业适合作为填充性载荷。此时，训练成本约为使用独占资源训练成本的 12.5%～31.5%。

2）图像分类。使用 ResNet-18 模型和 CIFAR-10 数据集进行图像分类，使用 PaddingTorch 作为训练框架。如表 5 所列，在该作业中，一个小批量的平均计算时间为 0.025 s，计算完成后的平均通信时间为 0.9 s，工作节点启动与退出的总时间为 7 s，即 $\tilde{e}$ 为 0.025 s，$\tilde{c}$ 为 0.9 s，$\tilde{\varphi}$ 为 7 s，计算可得 $0.3\%<\theta_{j,q}<2.7\%$。将该作业作为填充性载荷，会有大量时间浪费在处理通信与节点启动、退出等步骤上，此时，训练成本约为使用独占资源成本的 3.7～33.3 倍，该作业不适于作为填充性载荷。

4 PaddingTorch 训练框架

图 2 为 PaddingTorch 的结构图，处理流程如下：主节点维护一个资源池，资源池中包括所有可用节点。①表示随时可能有新节点向资源池提交注册信息并加入资源池；②和③表示从资源池和任务池分别取出一个可用节点和一个待处理的任务进行绑定；④表示当任务超时没有收到反馈信息时，将超时任务对应的节点标记为不可用，从资源池中删除；⑤表示将绑定的任务分发给相应节点，模型参数更新时发送新的参数信息；⑥表示接收到工作节点反馈的梯度；⑦表示将梯度交给进度管理器处理，进度管理器保存进度；⑧表示对任务池中的任务进行更新。

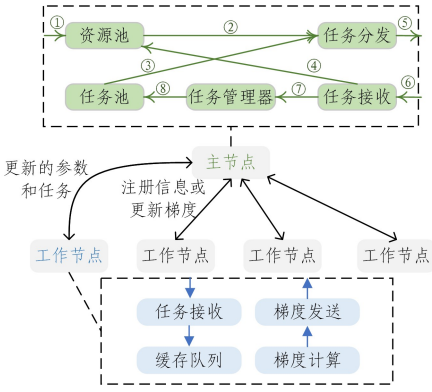


图 2 PaddingTorch 执行模型

Fig. 2 Execution model of PaddingTorch

PaddingTorch 保证了 3.1 节中提到的填充性载荷的相关性质。

1）正确性。使用该框架不会影响模型精度。框架采用参数服务器的通信方式，在收集到所有梯度信息后进行同步过程，没有进行异步更新。

2）可扩展性。框架支持节点随时加入与退出。新节点加入需向主节点发送注册信息。主节点收到注册信息即可将节点加入资源池，并在下一次分发任务时向该节点发送任务。节点退出即为节点被抢占（或因网络抖动导致任务超时等），主节点将该节点从注册表中删除。当同一节点试图重新加入通信组时，再次向主节点发送申请即可。

3）容错代价小。PaddingTorch 保证了工作节点上的任务具有幂等性和原子性，节点故障不影响其他节点的训练

1) http://moleculemind.com/

进度。进度管理器保障了任务的幂等性，仅会将同一梯度注册一次。任务接收器保障了任务的原子性，任务只有完全完成后才会提交到任务接收器并向进度管理器注册进度。即使一个节点被抢占，其他节点的训练进度也已保存到主节点中，不会受到影响。

批量大小会对模型的性能产生影响[18]，PaddingTorch 不改变框架使用者设定的批量大小。GPU 显存限制了单一节点上计算的小批量的上限，如在蛋白质复合物预测任务中，V100 GPU 仅可以同时计算一个蛋白质样本，小批量大小为 1，大批量大小为 128。PaddingTorch 根据小批量大小上限划分小批量的数量，并将小批量的计算任务分配给节点。在这种情况下，一个节点被高优先级任务抢占仅会损失一个小批量的计算进度。

此外，PaddingTorch 还使用了自适应任务调度方法来减少计算节点性能差异带来的等待时间。在大型计算集群中，可用资源的种类是多样的，资源的计算速度也不同[5]。集群中空闲资源充足时，每个节点被分配一个任务，计算完成后进行同步操作；空闲资源不足时，每个节点被分配多个任务，这些任务无依赖关系，全部完成后再进行同步操作。平均分配任务时，性能较强的节点计算完成后需要等待性能较差的节点。在同构节点上，等待时间几乎可以忽略，但在性能原本有较大差异的异构节点上，该问题造成的影响是不可忽视的。PaddingTorch 按计算能力分配任务，如果任务池中有任务，就从资源池中取出一个可用节点发送任务，节点完成当前任务后立刻再次加入资源池。节点的计算速度越快，被分配的任务就越多。

图 3 给出了平均分配任务与按能力分配任务的时间表。V100 显卡的性能低于 A100 显卡，在 3.4 节提到的蛋白质复合物预测实例中，执行一个任务的时长约为 4∶3。同时使用两种显卡进行训练，平均分配任务导致 A100 显卡需要等待；按计算能力分配任务，A100 显卡完成当前任务后直接开始下一任务的计算，相同时间内可以计算更多任务。

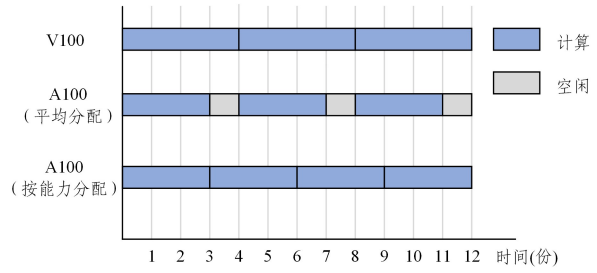


图 3　平均分配与按能力分配对比

Fig. 3　Equal distribution versus distribution based on capacity

5　实验设计与分析

实验使用的机器包括 4 个型号为 Intel(R) Xeon(R) Gold 6138 CPU @ 2.00GHz 的 CPU，共有 80 核 160 线程；以及 4 个型号为 Tesla V100-PCIE 的 GPU，每个 GPU 的显存为 32GB。

本文选择 PyTorch Lightning 框架与 PaddingTorch 框架进行对比，表 1 列出了 PaddingTorch 与 PyTorch Lightning 的部分功能实现对比。PyTorch Lightning 是 Pytorch 的高级封装库，主要目的是简化复杂网络代码的编写过程，使研究者专注于研究，减少花费在工程上的时间。PyTorch Lightning 提供了对弹性训练的支持，可以精确地保存当前任务状态，并在启动时恢复。该功能有一定局限性，仅支持使用 PyTorch 内置的 3 种采样器，即 RandomSampler，SequentialSampler 和 DistributedSampler；且对于 DistributedSampler，不支持将样本随机重排(shuffle)[16]。为了进行性能测试，本文将复合物预测模型中的采样器暂时修改为使用 DistributedSampler 且不进行随机重排。

表 1　PaddingTorch 与 PyTorch Lightning 的功能对比

Table 1　Function comparison between PaddingTorch and PyTorch Lightning

功能	PaddingTorch	PyTorch Lightning（精准检查点保存）	PyTorch Lightning（周期检查点保存）
故障恢复	主节点保存训练进度并恢复失败节点损失的进度	捕获中止信号并保存当前训练进度	指定若干轮次或若干批次对状态备份，难以确定备份频率；频率过高备份开销大，频率过低故障损失的进度多
节点退出	收到信号后直接退出用时约 1s	保存进度后退出用时约 13s	直接退出用时约 1s
节点加入	待启动节点向主节点申请并加入现有通信组用时约 6s	所有节点重新启动并重建通信组用时约 18s	所有节点重新启动并重建通信组用时约 18s
改变节点数	支持使用并行度范围内的任意节点数	不支持在训练过程中改变节点数	不支持在训练过程中改变节点数
采样器	支持使用任意采样器	支持 PyTorch 内置的 RandomSampler，SequentialSampler，DistributedSampler 3 种采样器	支持 PyTorch 内置的 RandomSampler，SequentialSampler 和 DistributedSampler 3 种采样器
随机重排	支持进行随机重排	DistributedSampler 采样器不支持随机重排	DistributedSampler 采样器不支持随机重排

本文使用 PyTorch Lightning 的精准检查点保存作为故障恢复方法。如表 1 所列，周期检查点保存需要指定若干轮次或若干批次进行状态备份，出现异常时需从上一个备份中恢复训练进度，并重建通信组。模型越大，备份开销越大。因此用户必须在故障造成的影响与备份带来的开销之间自行做出权衡，使用小粒度容错需要容忍较大的系统开销，使用大粒度容错则在出现故障时会损失更多训练进度。精准检查点保存对模型状态进行细致追踪，始终保持对采样器、循环进展等的记录，并捕获所有外部信号。捕获到中止信号时，PyTorch Lightning 将当前的所有状态保存到检查点，以备重启程序时精确地恢复训练进度。该方法的优点是只需保存一次，不需要指定保存频率。后续实验中 PyTorch Lightning 节点加入与节点退出均采用精准检查点保存方法。

5.1 基准实验

1)蛋白质复合物预测。表2中对比了使用不同框架在复合物预测模型上执行一个批量需要的计算时间与通信时间、工作节点启动和退出时间。PaddingTorch的启动时间减少了67%,退出时间减少了92%,启动与退出的时间开销共减少了77%。其中,启动时间指从进程开始执行到节点开始计算的时间。PyTorch Lightning的通信组不支持节点退出与加入,当有节点退出或加入时,必须重启所有节点并重新建立通信组。因此,在该时间内,PyTorch Lightning需要重启所有节点,启动4个进程耗时约8～12 s,从进程全部启动到开始计算的时间约为8s。PaddingTorch不需要重启所有节点,新加入的节点向主节点注册后即可加入通信组参与计算,该时间少于6s。工作节点退出时间指节点被抢占时释放资源需要的时间。PyTorch Lightning在收到退出信号时,必须保存当前的所有进度到检查点,如果保存工作未完成,则所有进度均会丢失,保存工作平均需要13 s。

表2 不同框架进行蛋白质复合物预测时的性能

Table 2 Performance of different frameworks in protein complex prediction

(s)

框架	节点启动时长	节点退出时长	启动退出开销	计算时长	通信时长
Padding Torch	6	1	7	10.9	2.80
PyTorch Lightning	18	13	31	10.7	0.13

本文设置了4个工作节点,规定每隔固定时间会有一个工作节点被抢占,并会在间隔15 s后被重启,另外3个工作节点始终不被抢占。训练任务在完全独占资源的情况下执行时间约为2745 s,即约为46 min。

值得注意的是,PyTorch Lightning在节点退出时保存所有状态需要花费一些时间,该时长主要在6～21 s区间范围内。本文使用的方案为每隔固定时间向PyTorch Lightning发出中止信号,PyTorch Lightning检查点保存完成后开始计时,等待15 s并重启工作节点。由于PyTorch Lightning保存检查点所需的具体时间难以预测,因此无法严格保证Padding-Torch与PyTorch Lightning框架均使用固定时长的资源。使用相同的间隔时间会导致PyTorch Lightning实际使用资源的时间较长,因为该框架收到信号后不会立刻释放资源,仍会继续占用资源直到进度保存完成。PaddingTorch需要释放资源时,可以直接释放,无需进行保存操作。如图2所示,任务接收器检测到任务超时,将对应的资源标记为不可用,并将其从资源池中剔除。进度管理器将任务重新加入任务池,寻找其他可用资源重新分配任务。

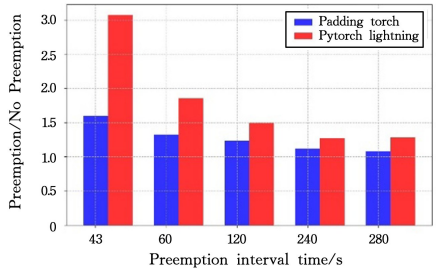


图4 抢占场景下执行时间与独占场景下执行时间的比值

Fig. 4 Ratio of execution time in preemptive scenarios and exclusive scenarios

在每隔30 s有一个工作节点被抢占时,PyTorch Lightning无法训练,PaddingTorch仍然确保了训练总时间为使用独占资源训练时间的1.88倍。如图4所示,在平均每隔43 s会有一个工作节点被抢占的情况下,PaddingTorch训练时间为使用独占资源训练时间的1.6倍,而PyTorch Lightning的训练时间超过原有时间的3倍,PaddingTorch的性能提升了46.7%;每隔60～480 s会有一个节点被抢占时,Padding-Torch的性能提升了15%～28.4%。

PaddingTorch减少了工作节点启动与退出77%的固定开销,仅需7 s,PyTorch Lightning则需要31 s。故Padding-Torch可以利用的资源粒度更小,对资源使用时长$\delta_{j,q}$较小资源的利用率更高。

表3列出了使用PaddingTorch进行训练时完成的任务数$r_{j,q}$、使用资源有效时长的比例$\theta_{j,q}$的值,以及训练成本ρ的理论值和实验值。在该表中,理论成本ρ_{ideal}始终小于实际成本ρ_{exp},且间隔时间越短,二者差距越大。在3.2节的理论分析中,并没有考虑主节点检测工作节点失败所用的等待时间,主节点会等待一个预期时间,超时未收到则认为任务失败。在此等待期间,所有工作节点已经完成分发的计算任务,等待分发下一阶段任务,浪费了一定的训练时间。工作节点失败频率越高,浪费的时间越多,ρ_{ideal}与ρ_{exp}差异越大。在间隔时间为480 s时,工作节点能完成约34个任务,一个任务失败造成的等待时间几乎可以忽略不计,ρ_{ideal}与ρ_{exp}相差约0.1%(假设α为0.1)。

表3 PaddingTorch在不同时间间隔下的性能与成本分析

Table 3 Performance and cost analysis of PaddingTorch at different intervals

	30 s	43 s	60 s	120 s	240 s	480 s
完成的任务数$r_{j,q}$	1～2	2～3	3～4	8～9	17～18	34～35
资源有效比例$\theta_{j,q}$	0.36～0.72	0.50～0.76	0.54～0.72	0.72～0.81	0.77～0.81	0.77～0.79
理论成本ρ_{ideal}	1.05α～1.13α	1.04α～1.10α	1.06α～1.10α	1.04α～1.07α	1.05α～1.06α	1.05α～1.06α
实际成本ρ_{exp}	1.72α	1.50α	1.26α	1.20α	1.10α	1.07α

表4列出了使用PyTorch Lightning进行训练时完成的任务数$r_{j,q}$、使用资源有效时长的比例$\theta_{j,q}$的值,以及训练成本ρ的理论值和实验值。在该表中,理论成本ρ_{ideal}整体与实际成本ρ_{exp}接近,部分值误差较大,原因可能是计算时选取了节点启动退出开销$\varphi_{j,q}$与任务计算时长e的平均值,完成的任务数$r_{j,q}$与资源有效时长比例$\theta_{j,q}$均为估算值。

表 4　PyTorch Lightning 在不同时间间隔下的性能与成本分析

Table 4　Performance and cost analysis of PyTorch Lightning at different intervals

	43 s	60 s	120 s	240 s	480 s
完成的任务数r_j	1	2～3	8～9	19～20	41～42
资源有效比例$\theta_{j,q}$	0.25～0.50	0.35～0.50	0.71～0.80	0.84～0.89	0.91～0.93
理论成本ρ_{ideal}	2.71α	2.33α～3.50α	1.40α～1.57α	1.19α～1.25α	1.10α～1.13α
实际成本ρ_{exp}	2.87α	1.76α	1.45α	1.25α	1.27α

资源可用时长较短，被频繁抢占时，节点启动退出开销$\varphi_{j,q}$是影响资源有效时长比例$\theta_{j,q}$的主要因素，因此 PaddingTorch 资源利用率高于 PyTorch Lightning；而当间隔时间超过 120s 时，其他开销c_i为影响资源有效时长比例$\theta_{j,q}$的主要因素，PaddingTorch 的通信开销远大于 PyTorch Lightning，这导致 PaddingTorch 资源利用率低于 PyTorch Lightning。节点被抢占时，PyTorch Lightning 的所有节点均需退出并重启，PaddingTorch 未被抢占的节点不受影响，尽管资源可用时长$\delta_{j,q}$较大时 PaddingTorch 的资源利用率 $\theta_{j,q}$较低，但 PaddingTorch 的训练成本与训练时长均小于 PyTorch Lightning。

2）图像分类。表 5 列出了图像分类作业中每批量平均计算时间与通信时间。PaddingTorch 通信时间远长于 PyTorch Lightning，为 PyTorch Lightning 通信时间的 45 倍。该作业一个训练周期时长较短、对计算资源需求不高，并不适用于 PaddingTorch，这也说明 PaddingTorch 在降低通信开销方面仍有待进一步优化。

表 5　不同框架进行图像分类时的性能

Table 5　Performance of various frameworks in image classification

(s)

框架	计算时长	通信时长
PaddingTorch	0.025	0.90
PyTorch Lightning	0.030	0.02

5.2　集群实验

阿里巴巴团队提供了来自实际生产的集群追踪，其中包括 Alibaba Platform for Artificial Intelligence（PAI）在持续的两个月时间内在超过 6 500 个 GPU（约 1 800 个机器）上的 AI/ML 工作负载[5,19]。该集群中共有 135 个 V100M32 8-GPU 节点，即共有 1 080 个 V100M32 GPU。从图 5 可知，有 60%的资源空闲时间大于 1 h，这说明目前集群的调度器仍有提升空间，部分资源存在较长的空闲时间。

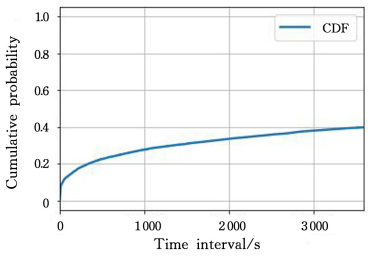


图 5　V100M32 GPU 的空闲时间间隔 CDF 图

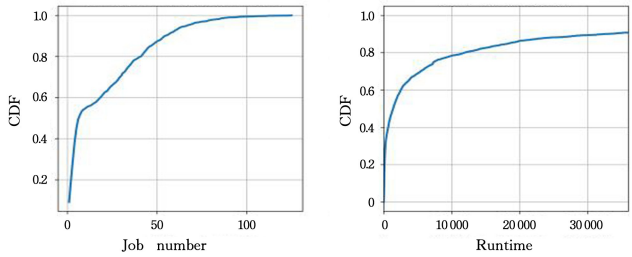
Fig. 5　CDF of idle time intervals of V100M32 GPU

资源 j 第 q 次被使用时完成的任务数为$r_{j,q}$，若$\dfrac{r_{j,q}}{r_{j,q}+1}$趋向于 1，则$\varepsilon_{j,q}+\varphi_{j,q}$可忽略，有效时长比例为：

$$\theta_{j,q}=\frac{\sum_{i=1}^{r_{j,q}} e_{m_i}}{\sum_{i=1}^{r_{j,q}}(e_{m_i}+c_{m_i})} \tag{11}$$

使用填充性载荷带来的额外时间开销并不明显，即资源可用时长$\delta_{j,q}$越大，资源被抢占的频率越低，抢占带来的时间开销越少。在蛋白质复合物预测作业中，当$\delta_{j,q}$大于 1 800 s 时，$\theta_{j,q}$接近极限值 0.8，此时工作启动和退出花费的时间、资源被抢占导致任务未完成浪费的时间可忽略不计。

图 6 为 V100M32 GPU 上执行作业数与作业执行时长的 CDF 图。在两个月的时间内，不同 GPU 上执行的作业数从 1 到 125 不等，接近 20%的作业执行时间超过 3 h。GPU 有若干小时的空闲时长时，填充性载荷被抢占带来的开销几乎可以忽略不计；GPU 连续若干小时进行同一作业的计算工作时，GPU 利用率较高，没有空闲资源用于填充性载荷。这两种情况均不是本文关注的重点。



(a) CDF of GPU job number　　(b) CDF of runtime

图 6　V100M32 GPU 的作业数与执行时间 CDF 图

Fig. 6　CDF of job number and runtime of V100M32 GPU

将 PAI 追踪中 135 个 V100M32 GPU 的使用情况划分为连续的 4h 窗口，并统计每个 GPU 在 4 h 内的作业切换次数。作业切换不频繁代表该时间段内当前 GPU 长时间处于工作状态或长时间处于空闲状态。本文选择了作业切换次数最多的 4 个窗口，图 7 展示了对应 GPU 的工作负载使用情况，由于 4 个窗口的起始时间不同，因此在图中使用了相对时间。

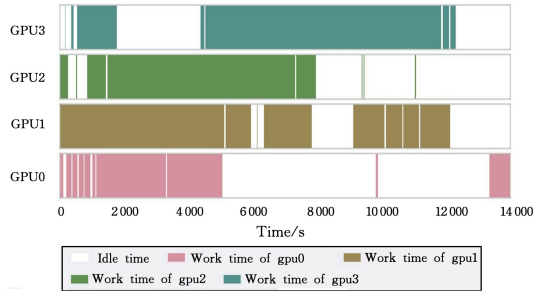


图 7　4 个 GPU 的使用情况

Fig. 7　Usage of four GPUs

本文使用 4 个 V100 GPU 对上述 4 个 GPU 的作业执行情况进行了重放，已有作业被认为是高优先级作业。填充性

载荷为最低优先级作业，仅使用图 7 中的空闲资源。模拟调度器按照 PAI 追踪中 GPU 的作业执行情况进行调度，资源空闲时启动填充性载荷，到达普通负载开始时刻会终止填充性载荷的执行，直到资源空闲时再次启动。

每个工作节点拥有一块 V100 GPU，其中 PaddingTorch 为蛋白质复合物预测作业的训练框架，训练时长为 7 693 s。在该作业独占 4 块 GPU 资源时，训练时长为 2 745 s，此时使用空闲资源的训练时长是独占资源情况下的 2.8 倍，即：

$$P_{\text{exec}}=2.8\times P_{\text{ideal}} \tag{12}$$

将每个 GPU 的使用时长相加，可以得到填充性载荷的资源使用量，即：

$$\sum_{j=1}^{M}\sum_{q=1}^{Q_j}\delta_{j,q}=8147\ \text{s}\cdot\text{GPU} \tag{13}$$

在独占资源情况下训练时长为 2 745 s，资源使用总时长为 10 980 s·GPU。使用填充性载荷后，资源使用时长减少了 26%。不使用填充性载荷时，由于所有节点需要进行同步操作，训练速度快的节点需要等待训练速度慢的节点完成当前训练批次并进行同步。独占资源时，如图 8 所示，会浪费红色虚线代表的等待时间。仅使用图 7 中的空闲资源时，同时可用的资源数量小于等于 2 的时长占比为总时长的 78.08%，可用资源数量为 4 的时长占比仅为 7.31%。资源数量为 1 或 2 时，与资源数量为 4 相比，红色虚线代表的等待时间有所减少。此外，PaddingTorch 可以低开销地适应动态变化的资源数量，节点被抢占、重启造成的时间开销较低。因此使用填充性载荷的总资源使用时长比独占所有资源时减少了 26%。

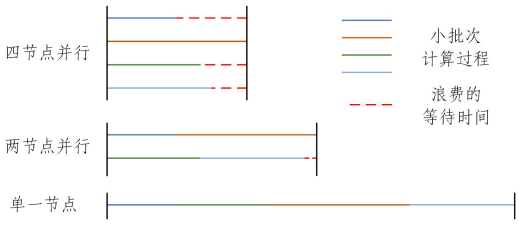


图 8 不同并行度下任务的同步(电子版为彩图)

Fig. 8 Task synchronization at different parallelism levels

在阿里云中，V100 GPU 资源价格约为 4US $/h，包括 32GiB 内存与 8vCPU，相同处理器型号且包括 32GiB 内存与 8vCPU 的 CPU 资源价格约为 0.5 US $/h[13]。假设填充性载荷使用的最低优先级资源价格为高优先级资源价格的十分之一，则使用填充性载荷的成本为使用独占资源情况下的 16%，降低了 84%的训练成本。训练过程中，4 个工作节点共启动了 28 次。工作节点退出需要花费约 1 s 的时长，高优先级普通负载的等待时间需要增加 1 s。PAI 集群中平均等待时间约为 1 884 s，填充性载荷导致的额外排队时延小于 0.1%。将工作节点启动与退出开销视为填充性载荷使用资源切换导致的额外开销，此时间段的 GPU 资源并没有真正用于任务。通过计算可得，填充时段内的系统资源利用率平均提升了 25.8%。

PyTorch Lightning 不适宜作为该场景下的训练框架。如表 1 所列，PyTorch Lightning 不支持在训练过程中改变节点数，这意味着 PyTorch Lightning 仍需要进行组调度，必须等待所有资源可用。在图 7 中，所有资源均空闲的时长仅占 7.31%，对整体资源利用率的提升有限。

6 相关工作

1)集群调度策略。为了提升集群资源利用率，许多工作致力于探索更好的调度策略。Weng 等[5]通过使用 GPU 共享技术、预测重复实例的持续时间来提高 GPU 机器的利用率，并尽可能完成更多任务。Hu 等[9]提出了一种准最短服务优先(Quasi-Shortest-Service-First)调度方法来减少集群的平均作业完成时间。Xiao 等[11]在训练框架的内存和计算管理单元中引入两种新的动态缩放机制，使用机会调度(Opportunistic Scheduling)提高集群资源利用率。Weng 等[4]使用碎片梯度下降(Fragmentation Gradient Descent)对任务进行打包，减少 GPU 碎片的增长，提高 GPU 的分配率。Li 等[10]引入容量借贷(Capacity Loaning)，将闲置的推理服务器用于训练作业，并使用作业级弹性扩展以充分利用借用的服务器。尽管已有许多相关研究工作，但现有集群中的整体资源利用率仍较低，约为 25%～50%[4]。

Przybylski 等[17]设计了一个名为 HPC-Whisk 的 FaaS 层，使用 FaaS 负载填充高性能集群。约 90%的 FaaS 负载可在不到 1min 内完成，空闲时间段内可能会填入一个或多个 FaaS 调用者作业，一台物理机器也可以同时承担数十甚至数百个函数。与填充性载荷相比，FaaS 负载执行时间短、资源需求低。上述工作没有考虑 GPU 资源的使用情况，且只能使用 FaaS 类型的负载。填充性载荷可以将资源需求大、执行时间长的深度学习训练负载填充到系统的空闲资源中。

2)混合部署。混合部署(Colocation)指将时延敏感的在线作业与对时延要求不高的离线作业部署在同一集群中[20]。为了确保在线作业的低延迟，集群通常会为在线作业分配过量的计算资源，而离线作业快速增长的计算需求导致资源不足。将部分在线作业的资源分配给离线作业可以缓解这一矛盾，但可能导致在线作业与离线作业的性能干扰。在文献[21]中，仅有 52%的作业服务质量不受影响，15%的作业 QoS 波动大于 10%。使用填充性载荷，仅会导致高优先级负载的排队时延增加约 0.1%。

结束语 本文提出了填充性载荷——一种在计算集群中动态分配和执行的负载。填充性载荷仅利用集群中的空闲资源进行计算，可以随时被抢占，导致高优先级负载的平均排队时延增加 0.1%左右，实现了适用于填充性载荷的训练框架 PaddingTorch，该框架拥有健壮的通信组，可以容忍节点随时加入与退出，使用动态的资源数量进行计算。使用 4 块 GPU 模拟了 PAI 集群中最繁忙的 4 个 GPU 时间段上的作业调度情况，利用该时段的空闲资源进行蛋白质复合物预测作业，训练时间为使用独占资源训练时间的 2.8 倍，但成本降低了 84%，填充时段内的 GPU 资源利用率提升了 25.8%。

在通信开销方面，PaddingTorch 使用参数服务器方法进行通信，该方法使其具有健壮的通信组，同时也增加了通信开销。未来会设计兼具通信开销较小、通信组健壮两个优点的通信与同步方法。在 GPU 利用率方面，目前仅关注了处于完全空闲状态的 GPU 资源，未来会加入对正在运行但其 SM 计算单元和显存利用率相对较低的 GPU 资源的管理和

优化。未来也将在更大规模的集群上对不同批量大小以及具有小批量上限的真实负载进行实验测试与性能分析。

参 考 文 献

[1] JUMPER J，EVANS R，PRITZEL A，et al. Highly accurate protein structure prediction with AlphaFold[J]. Nature，2021，596(7873)：583-589.

[2] BATEMAN A，MARTIN M J，ORCHARD S，et al. UniProt：the Universal Protein Knowledgebase in 2023[J]. Nucleic Acids Research，2023，51(D1)：D523-D531.

[3] DISKIN M，BUKHTIYAROV A，RYABININ M，et al. Distributed deep learning in open collaborations[J]. Advances in Neural Information Processing Systems，2021，34：7879-7897.

[4] WENG Q，YANG L，YU Y，et al. Beware of Fragmentation：Scheduling GPU-Sharing Workloads with Fragmentation Gradient Descent[C]// 2023 USENIX Annual Technical Conference (USENIX ATC 23). 2023.

[5] WENG Q，XIAO W，YU Y，et al. MLaaS in the wild：Workload analysis and scheduling in Large-Scale heterogeneous GPU clusters[C]// 19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22). USENIX Association，2022：945-960.

[6] JIA Z，MAGGIONI M，STAIGER B，et al. Dissecting the NVIDIA volta GPU architecture via microbenchmarking[J]. arXiv：1804. 06826，2018.

[7] NVIDIA[EB/OL]. [2023-06-29]. https：//developer. download. nvidia. com/compute/DCGM/docs/nvidia-smi-367. 38. pdf.

[8] YEUNG G，BOROWIEC D，FRIDAY A，et al. Towards GPU utilization prediction for cloud deep learning[C]// Proceedings of the 12th USENIX Conference on Hot Topics in Cloud Computing. 2020：6-6.

[9] HU Q，SUN P，YAN S，et al. Characterization and prediction of deep learning workloads in large-scale gpu datacenters[C]// Proceedings of the International Conference for High Performance Computing，Networking，Storage and Analysis. 2021：1-15.

[10] LI J，XU H，ZHU Y，et al. Lyra：Elastic scheduling for deep learning clusters[C]// Proceedings of the Eighteenth European Conference on Computer Systems. 2023：835-850.

[11] XIAO W，REN S，LI Y，et al. AntMan：Dynamic Scaling on GPU Clusters for Deep Learning[C]// OSDI. 2020：533-548.

[12] Amazon Web Service(AWS)- Cloud Computing Services[EB/OL]. [2023-04-15]. https：//aws. amazon. com/.

[13] Alibaba Cloud[EB/OL]. [2023-04-15]. https：//www. alibabacloud. com/.

[14] Tencent Cloud[EB/OL]. [2023-04-15]. https：//cloud. tencent. com/.

[15] SERGEEV A，DEL BALSO M. Horovod：fast and easy distributed deep learning in TensorFlow[J]. arXiv：1802. 05799，2018.

[16] PyTorch Lightning[EB/OL]. [2023-04-15]. https：//lightning. ai/docs/pytorch/latest/.

[17] PRZYBYLSKI B，PAWLIK M，UK P，et al. Using unused：noninvasive dynamic FaaS infrastructure with HPC-whisk[C]// SC22：International Conference for High Performance Computing，Networking，Storage and Analysis. IEEE，2022：1-15.

[18] GOYAL P，DOLLAR P，GIRSHICK R B，et al. Accurate，Large Minibatch SGD：Training ImageNet in 1 Hour[J]. arXiv：1706. 02677，2017.

[19] Alibaba Cluster Trace Program [EB/OL]. [2023-04-15]. https：//github. com/alibaba/clusterdata.

[20] WANG K J，JIA T，LI Y. State-of-the-art Survey of Scheduling and Resource Management Technology for Colocation Jobs[J]. Journal of Software，2020，31(10)：3100-3119.

[21] ROMERO F，DELIMITROU C. Mage：Online and interference-aware scheduling for multi-scale heterogeneous systems[C]// Proceedings of the 27th International Conference on Parallel Architectures and Compilation Techniques. 2018：1-13.

DU Yu，born in 2001，postgraduate. Her main research interests include distributed systems and deep learning frameworks.

YU Zishu，born in 1996，Ph. D candidate. His main research interests include distributed systems and runtime management.

（责任编辑：杨雪敏）