计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.220900110

深度学习容器云平台下的GPU共享调度系统

王 壮[1] 王平辉[1] 王彬丞[1] 武文博[1] 王 斌[2] 丛鹏宇[2]
1 西安交通大学智能网络与网络安全教育部重点实验室 西安 710049
2 中国移动通信有限公司研究院 北京 100053
(wangzhuang@stu.xjtu.edu.cn)

摘 要 近年来,容器由于具有轻量级以及高可扩展性,逐渐替代了虚拟机,被广泛应用于深度学习云平台中。但目前深度学习云平台在GPU资源管理上依然存在着不足,主要表现为由于容器编排技术的限制,多个容器无法共享使用GPU资源,而对于一些小规模模型的训练任务和推理任务,单个任务并不能充分利用整张GPU卡的计算资源。当前的独占模式会导致昂贵的GPU资源的浪费,降低资源效率和服务可用性。针对这一问题,提出了一种GPU共享调度系统。一方面,基于Kubernetes的Operator机制对现有集群功能进行扩展,实现了多个Pod共享使用GPU资源,同时设计了一种代理机制保证了与原生Kubernetes的兼容性。另一方面,基于GPU时间片与抢占机制,实现了GPU资源的动态管理与调度,在多个任务之间进行细粒度的协调,并减少了任务干扰。实验结果表明,与原生Kubernetes调度系统相比,该系统能够将一组深度学习训练任务的完成时间平均减少约20%,使得集群GPU资源利用率平均提升约10%。在共享使用GPU时高优先级任务性能相较于独占GPU损耗不到5%,同时能够使得低优先级任务以20%的性能运行在同一张GPU上。

关键词:深度学习云平台;GPU共享;容器调度;Docker;Kubernetes
中图法分类号 TP181

GPU Shared Scheduling System Under Deep Learning Container Cloud Platform

WANG Zhuang[1],WANG Pinghui[1],WANG Bincheng[1],WU Wenbo[1],WANG Bin[2] and CONG Pengyu[2]
1 Ministry of Education Key Lab for Intelligent Networks and Network Security,Xi'an Jiaotong University,Xi'an 710049,China
2 China Mobile Research Institute,Beijing 100053,China

Abstract In recent years,containers have gradually replaced virtual machines and are widely used in deep learning cloud platforms due to their lightweight and high scalability. However,the deep learning cloud platform still has deficiencies in GPU resource management,which are mainly manifested as multiple containers cannot share GPU resources due to the limitation of container orchestration technology. For some small-scale model training tasks and model inference tasks,a single task cannot fully utilize the computing resources of the entire GPU card. The current exclusive mode will result in a waste of expensive GPU resources,reduce resource efficiency and service availability. In response to this problem,this paper proposes a GPU sharing scheduling system. On the one hand,the Kubernetes-based Operator mechanism extends the existing cluster functions,enabling multiple Pods to share GPU resources,and designs an agent mechanism to ensure that compatibility with native Kubernetes. On the other hand,based on the GPU time slice and preemption mechanism,the dynamic management and scheduling of GPU resources is realized,fine-grained coordination among multiple tasks is performed,and task interference is reduced. Experimental results show that compared with the native Kubernetes scheduling system,the proposed system can reduce the completion time of a group of deep learning training tasks by about 20% on average,and increase the utilization of cluster GPU resources by about 10% on average. When the GPU is shared,the performance loss of high-priority tasks is less than 5% compared to the exclusive GPU,and the low-priority tasks can run on the same GPU with 20% of the performance.

Keywords Deep learning cloud platform,GPU sharing,Container scheduling,Docker,Kubernetes

到稿日期:2022-09-13 **返修日期**:2022-12-14
基金项目:国家重点研发计划(2021YFB1715600);国家自然科学基金(61902305,61922067);深圳市基础研究基金(JCYJ20170816100819428);"人工智能"教育部-中国移动建设项目(MCM20190701)
This work was supported by the National Key R & D Program of China(2021YFB1715600),National Natural Science Foundation of China(61902305,61922067),Shenzhen Basic Research Grant(JCYJ20170816100819428) and MoE-CMCC "Artifical Intelligence" Project(MCM20190701).
通信作者:王平辉(phwang@mail.xjtu.edu.cn)

1　引言

随着深度学习研究在各个领域逐渐落地，工业界开始探索一套从数据获取到模型训练再到模型落地的完整体系，降低模型训练的计算资源成本，缩短模型开发的周期。为达到这一目的，各大云服务提供商都推出了自己的深度学习云平台，如亚马逊 Sagemaker[1]、谷歌 GCP[2]、阿里云 PAI、百度 BML、腾讯 TI-ONE、华为 ModelArts。这些深度学习云平台解决了传统方法直接在 PC 或服务器上训练带来的计算资源成本高、缺少统一的资源监控系统、算法人员需要参与运维（如搭建硬件软件环境）等问题；同时，它们基于统一的资源监控和调度，使计算中心的资源利用率大大提升，有效地降低了成本。

当前构建深度学习云平台依赖的两大基础技术为容器虚拟化技术和容器编排技术。容器虚拟化技术是继虚拟机之后的新一代操作系统虚拟化技术，具有性能好、占用小、启动快等特点。而 Docker[3] 是容器虚拟化技术的一种，也是当前使用最广泛的容器虚拟化技术，在深度学习云平台中用于为用户提供隔离的环境。随着越来越多的应用被部署在容器中，对容器的部署、调度、管理与扩展的需求愈发凸显，便出现了容器编排技术，用于对容器进行自动化管理。Kubernetes[4] 是当前的一种主流容器编排技术，用于统一管理分布式集群里所有节点上的容器，具有高度的可扩展性，同时还具有负载均衡、存储编排、自动部署和回滚等功能。

图形处理器（Graphics Processing Unit，GPU）由于具有强大的并行处理能力而被广泛应用于图像识别、自然语言处理等计算密集型任务中，以提高计算速度。显著的性能优势吸引了云提供商在其云环境中部署 GPU[5]。而目前的云平台在 GPU 资源管理方面仍然存在不足。虽然 Kubernetes 能够对容器进行编排与管理，但其能够识别与分配的本地资源只有 CPU 和内存。Kubernetes 将 GPU 连接到其管理的容器中的方式，目前主要是通过设备插件[6]的形式来实现，而设备插件不支持对 GPU 的共享使用或部分分配。这就导致通过容器编排技术管理的容器能够使用的 GPU 资源最小单元为 1，且无法共享资源[7]。实际上，对于一些小规模的模型训练任务和模型推理任务而言，单个任务通常不能完全利用整个 GPU 卡的计算资源。因此独占模式会导致昂贵的 GPU 资源的浪费，降低资源效率和服务可用性，导致排队时间更长。

针对上述问题，本文提出了一种深度学习云平台下的 GPU 共享调度系统。一方面，基于 Kubernetes 的 Operator 对于现有集群功能进行扩展，创建自定义资源 vPod 并根据其资源优先级进行调度，使得多个任务 Pod 能够共享使用 GPU 资源；同时设计了一种代理机制以保证与原生 Kubernetes 的兼容性，实现了对可共享 GPU 的管理。另一方面，本文基于 GPU 时间片分片以及时间片抢占机制，设计了 GPU 资源动态管理模块，能够在多个任务之间进行细粒度的协调，并减少任务干扰。最后，设计实验验证了本文系统相比原生 Kubernetes 在任务完成时间、GPU 利用率以及任务性能损耗方面的提升。

实验结果表明，与原生 Kubernetes 调度系统相比，本文系统能够将一组深度学习训练任务的完成时间平均减少 19.49%，集群 GPU 资源利用率平均提升 10.64%。在保证高优先级任务的性能相比独占 GPU 损耗不到 5%的同时，本系统低优先级任务能以 20%的性能运行在该 GPU 上。

2　相关工作

随着深度学习训练需求的持续增加以及容器虚拟化技术的不断发展，其对计算资源的需求量也日益增长。因此，如何更加合理地使用 GPU，解决目前 Kubernetes 独占 GPU 使用模式造成的资源浪费问题也愈发受到关注。目前学术界出现了一些多任务 GPU 共享的研究成果。

ConvGPU[8] 是一种多个容器共享使用 GPU 的解决方案，该方案通过拦截 CUDA 库的调用，限制多个容器之间的显存使用以支持共享显存资源。然而该方案只能支持 GPU 显存的共享使用，并且只能用于单个 GPU 的共享，无法在容器集群中共享使用 GPU 显存和计算资源。

目前在 Kubernetes 集群中共享 GPU 的一种方式是修改设备插件，在向设备插件注册 GPU 设备时，注册的设备数量是其管理的 GPU 数量乘以一定的比例系数[9-10]。这样用户只需在 Pod 的资源规范中申请需要使用的虚拟 GPU 设备的份数，便能够申请小数位的 GPU 资源。GaiaGPU[11] 通过修改设备插件，使得多个任务能够在容器集群中共享使用 GPU 资源，并采用监控调节的方式实现资源隔离。

KubeShare[12] 在共享方面采用自定义资源的方式来实现 GPU 共享调度，并设计了一种位置感知调度算法，来缓解任务之间的干扰问题。在资源隔离方面采用时间片分片实现算力和显存的隔离。

这些方案虽然能够使得多个任务共享使用一张 GPU 卡，一定程度上提高了 GPU 资源的利用率，但很少在生产集群中使用，原因是虽然提高 GPU 资源利用率是有益的，但保证一些优先级较高的任务的性能也是至关重要的。这些方案在同一张 GPU 上同时执行多个作业时，对于任务之间的性能干扰问题考虑较少，导致高优先级作业的性能显著降低。因此现有 GPU 生产集群大多采用独占模式分配 GPU，以此来保证任务的性能[13]。

3　系统整体设计

本文构建的容器 GPU 共享调度系统结构如图 1 所示。本文基于 Operator 对 Kubernetes 现有的功能进行扩展，构造一种名为 vPod 的自定义资源，该资源对象底层是对 Pod 资源的封装，并对 Pod 的功能进行扩展，支持多个任务共享使用 GPU 资源。为了对自定义资源进行调度与生命周期的管理，本文基于自定义控制器实现了 vPod 调度器与 vPod 管理器。

从工作流程来看，GPU 共享调度系统能够接收用户提交的任务信息（包括 vPod 名称、vPod 命名空间等）以及任务的优先级和显存数量，完成自定义资源 vPod 的调度以及工作 Pod 的创建和初始化工作，并协调不同优先级任务 GPU 资源的使用。对应的前端交互界面的设计如图 2 所示。

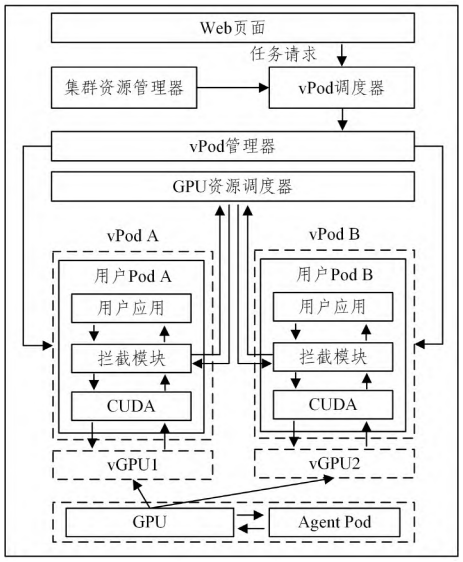

图 1 GPU 共享调度系统架构图

Fig. 1 Architecture of GPU shared scheduling system

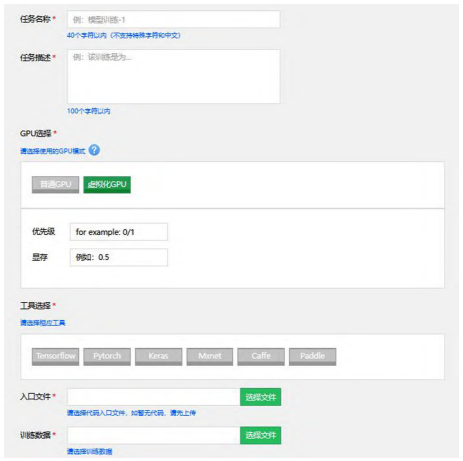

图 2 系统前端交互设计

Fig. 2 System front-end interactive design

从整体功能模块来看，该系统可以分为 GPU 共享模块和资源隔离模块。GPU 共享模块能够使多个深度学习任务共享同一 GPU 的计算和显存资源，主要包括 vPod 调度器、vPod 管理器、集群资源管理器等子模块；资源隔离模块能够根据任务的优先级协调不同任务的 GPU 使用权限与使用时间，主要包括拦截子模块和 GPU 资源调度子模块。

3.1 任务资源类型

根据用户的不同资源需求，本文设计了两种优先级类型的任务，即高优先级任务和低优先级任务。用户需要设置任务的 GPU 内存使用量 Memory_Limit，即该任务能够使用的最大的 GPU 内存量。例如，Memory_Limit=0.4 表示该任务能够占用 GPU 卡 40%的内存空间。本文的设计目的是在保证高优先级任务性能的前提下，通过低优先级任务窃取空闲的 GPU 时间周期，以此来提高 GPU 的利用率。

下面将具体介绍为实现上述两种资源类型任务的 vPod 调度和 GPU 资源动态分配，以及为了解决 vGPU 与原生 Kubernetes 调度器的兼容问题，防止出现资源竞争所做的工作。

3.2 vPod 调度器与共享调度算法

vPod 调度器是为了控制自定义资源 vPod 调度到集群中合适的节点与 GPU 上而实现的自定义调度器。其作用是根据用户设置的任务资源类型、GPU 内存请求量，以及集群剩余资源量，通过 GPU 共享调度算法，确定 vPod 与设备 ID 的映射关系。

vPod 调度器采用的共享调度算法如算法 1 所示，根据任务的优先级和显存请求量不同，调度器执行的具体策略也有所不同。总体来讲，vPod 的调度过程分为预选阶段和优选阶段。

算法 1 GPU 共享调度算法

输入：vPod 的创建请求 r；目前可分配的 vGPU 列表 D

输出：为该 vPod 分配的 vGPU 的设备 ID

```
/* Step1：预选阶段：根据任务的优先级和资源请求量筛选出符合要求的 vGPU 列表 S
1. S=[];
2. if r.priority == high then
3.     for each d ∈ D do
4.         if r.memory > d.memory then continue;
5.         if d.highTag ≠ null then continue;
6.         S=append(S,d);
7. else if r.priority is low then
8.     for each d ∈ D do
9.         if r.memory > d.memory then continue;
10.            if d.gpu_util > gpu_util_limit or len(d.lowTag) > low_nums_limit then continue;
11.        S=append(S,d);
/* Step2：优选阶段：从符合资源条件的设备列表中选择最优的 vGPU 设备
12. d=worst_fit(r,S);
13. if d == null then d=new dev();
14. if r.priority is high then
15.     d.highTag=r.highTag;
16. else if r.priority is low then
17.     d.lowTag=append(d.lowTag,r.highTag);
18. return d.id.
```

在预选阶段，共享调度算法会根据任务申请资源量与集群资源管理器收集的各 vGPU(virtual GPU)设备的空闲资源量，筛选出满足任务类型以及资源需求的 vGPU 列表。其中 vGPU 是由本系统管理的可共享 GPU。当任务为高优任务时，如算法 1 的第 2－6 行所示，如果该 vGPU 的显存剩余量无法满足 vPod 的显存请求量或者该 vGPU 上已经存在高优先级任务，则不会将该 vGPU 加入候选列表 S。这里主要是为了防止多个高优先级任务抢夺 GPU 资源，以保证高优先级任务的性能。当任务为低优任务时，如算法 1 第 7－11 行所示，除了显存资源的限制外，只有在过去的 1min 内 GPU 利用率低于 gpu_util_limit(本文设置为 0.8)且该 vGPU 上的低优先级任务数量不超过 low_nums_limit(GPU 上并发任务的最大数量，可根据 GPU 硬件参数进行调整)的 vGPU 才能被选为候选 vGPU。这里主要考虑若 GPU 利用率过高或任务数量过多，会导致低优先级任务的性能下降。

在优选阶段，调度算法会从可调度的 vGPU 列表中选择

最优的 vGPU，具体的调度策略如算法 1 第 13－19 行所示。对于待调度的任务，均采用最坏适应算法的思想。对不同优先级的任务，调度方式又有所不同。1）待调度的任务是高优先级任务，会优先选择仅运行低优先级任务的 vGPU，且低优先级任务数量越少、剩余显存量越多，则调度概率越大；若不存在满足要求的 vGPU，则会返回一个新的设备 ID，表示该任务需要在一个新的 vGPU 上启动。2）待调度的任务是低优先级任务，则会优先选择仅存在低优先级任务的 vGPU，其次会调度到运行着高优先级任务的 vGPU，且调度概率与该 vGPU 上低优先级任务数量以及显存量成负相关，否则会返回一个新的设备 ID。共享调度算法的设计思想是，共享 GPU 上运行的低优先级任务数量越多，对于高优先级任务的性能产生影响的可能性就越大。

3.3　vPod 资源管理器与 vGPU 管理

vPod 管理器是本文实现的自定义控制器，主要负责创建和管理 vGPU，根据 vPod 的资源规范创建实际工作的任务 Pod，并在 Pod 中安装拦截库以及初始化 Pod 的环境变量。

当 vPod 管理器通过 Kubernetes Watch 机制监听到 vPod 调度完成后，会获取该 vPod 调度的节点名称和设备 ID。随后 vPod 管理器会创建实际工作的任务 Pod 并完成容器的初始化，具体做法是将容器的 NVIDIA_VISIBLE_DEVICES 变量设置为该 vGPU 的设备 ID 对应的 UUID，并在容器的环境变量中设置 GPU 资源调度器的 IP 地址以及服务端口，然后在相应节点上创建工作 Pod。

由于本文的 GPU 共享调度系统与原生 Kubernetes 是共存的，因此，如何保证 vGPU 与 Kubernetes 原生调度器[14]的兼容也是一个关键性的问题。通过上述的 GPU 共享调度方法能够将任务调度到同一张 GPU 上，但是 Kubernetes 却无法感知到该 GPU 是否为本系统管理的可共享 GPU，因此可能会导致资源争用的问题。例如，当一个高优先级任务已经申请使用一张 vGPU，在工作容器中设置了该 vGPU 的 UUID 并安装了拦截库，但 Kubernetes 的原生调度器无法获知这一变化，后续可能会将该 GPU 分配给其他的 Pod 使用。由于 vPod 管理器只会在由 vPod 创建的工作 Pod 安装拦截库，因此无法限制其他 Pod 对 GPU 资源的使用。这样就会导致我们无法保证该高优先级任务获得的 GPU 资源量以及任务的性能。为解决这一问题，本系统设计并实现了一种代理机制来创建 vGPU。当 vPod 管理器获得了一个新的设备 ID 时，首先会启动一个 Agent Pod 来申请 GPU，并获取该 GPU 的 UUID，然后将其与新的设备 ID 形成映射并存储在 vPod 管理器中。Agent Pod 不会进行任何计算任务，只是向 Kubernetes 调度器申请 GPU，该 GPU 后续不会被分配给 Kubernetes 的其他 Pod。当该 vGPU 上不存在任何任务时，vPod 管理器会删除对应的 Agent Pod，Kubernetes 即可获知该 GPU 资源已经被释放，从而能够避免资源争用问题的发生。

3.4　GPU 资源隔离与抢占式调度

多个任务容器共享使用 GPU 时，容器对 GPU 资源的过度占用往往会导致任务性能下降，当显存过度分配时任务甚至会因显存溢出而失败。因此，除了实现多个容器共享使用 GPU 外，还需要限制容器可以使用的 GPU 资源数量，以保证高优先级任务性能不受影响。

在深度学习任务中，GPU 的算力资源主要用来处理核函数的下发，显存资源主要用于存储模型的参数以及计算过程中产生的临时变量。因此，为了隔离容器之间的 GPU 使用，本文系统基于 Linux 的 LD_PRELOAD 机制设计，通过编写核函数下发、显存分配和拷贝，以及上下文同步相关的拦截函数，将其编译成动态库后，经过 LD_PRELOAD 加载，当程序中调用 CUDA 库函数时，会被拦截函数拦截并进行相应的资源调度与处理。表 1 列出了本文拦截的 CUDA API 库函数。

表 1　拦截的 CUDA API 库函数

Table 1　Intercepted CUDA APIs functions

API Functions	Features
cuLauncKernel	核函数下发
cuLaunchCooperativeKernel	线程核函数下发
cuMemAlloc	设备内存分配
cuMemAllocPitch	二维矩阵内存分配
cuMemAllocManaged	统一内存管理分配
cuArrayCreate	二维矩阵创建
cuArray3DCreate	三维矩阵创建

本文的 GPU 资源隔离可以分为显存隔离和算力隔离两部分。在显存隔离方面，本文采用空间复用的方式使得多个任务各自使用 GPU 显存的一部分，且不得超出申请的显存数量。在任务调用显存申请的 CUDA API 时进行拦截并检查该任务的剩余可使用显存量。若任务剩余显存量满足申请量，则在记录该任务的显存使用量后将请求转发至真正的 CUDA API 进行显存的申请与分配；若任务剩余显存量不足，则会抛出显存不足溢出的错误（Out of Memory）。

在算力隔离方面，本文采用时间复用的方式在多个任务之间共享 GPU 算力资源。任务在使用 GPU 算力资源下发核函数前需要向 GPU 资源调度器申请时间片，只有得到 GPU 时间片的任务才能够使用 GPU 下发核函数。GPU 资源调度器是运行在宿主机上的守护进程，负责 GPU 时间片的调度与回收。任务容器得到时间片后，该任务的核函数下发请求会被转发至真正的 cuLauncKernel 等 CUDA API 函数上，若超过该时间片，任务想要使用 GPU 必须再次申请时间片。为了降低任务之间的干扰，保证高优先级任务的性能，本系统针对不同资源类型的任务，采取不同的 GPU 时间片调度方案。

当高优先任务和低优先级任务同时请求使用 GPU 时，为保证高优先级任务的性能，资源调度器会优先将 GPU 时间片分配给高优先级任务。当只有低优先级任务请求使用 GPU 时，资源调度器才会将 GPU 时间片分配给低优先级任务。这里存在的挑战是，当低优先级任务持有时间片时，高优先级任务需要等待该时间片结束后，才能得到 GPU 时间片，这会导致高优先级任务性能的降低。

为了降低对高优先级任务的干扰，本文对高优先级任务设置了时间片的抢占机制。当 GPU 资源调度器检测到高优先任务申请时间片时，若此时低优先级任务正持有时间片，

那么调度器会回收低优先级任务的 GPU 时间片，并将时间片调度给高优先级任务。如此，高优先级任务最多会延迟一个核函数的执行时间而非整个时间片，降低了对于高优先级任务性能的影响。当 GPU 上只有低优先级任务时，能够动态地增加分配给低优先级任务的时间片，以此来提升低优先级任务的性能。

4 实验验证

基于上述理论设计，本文实现了 GPU 共享调度系统原型。为了验证系统的有效性与可靠性，本文从系统的任务完成时间、GPU 利用率、任务的性能损耗与性能提升 3 个方面与原生 Kubernetes 系统进行对比，实验环境如表 2 所列。

表 2 集群硬件配置

Table 2 Cluster hardware configurations

Configuration	Model or Size
CPU	Intel(R) Xeon(R) CPU E5-2690
RAM/GB	512
GPU	NVIDIA Tesla V100
Memory/GB	32
GPU Nums	4

4.1 数据集

本文从 Github 中选择了 5 种常见的深度学习模型以及两种开放数据集用于对比实验，如表 3 所列。

表 3 深度学习模型与数据集

Table 3 Deep learning models and datasets

Models	Frame	Datasets
VGG11 VGG16 VGG19 MobileNet SqueezeNet	Pytorch[15]	Cifar10 Cifar100

4.2 GPU 利用率与任务完成时间

本文在集群节点上进行实验，从模型列表的 5 种模型中随机选取模型启动深度学习训练任务并进行调度。对于每种模型，随机选取一种数据集进行训练。在 1 h 内共启动 12 个深度学习训练任务，任务的资源类型默认为低优先级任务。在实验过程中调用 nvml 库[16]的接口每秒钟采集一次 GPU 利用率。为了便于展示 GPU 利用率结果图，每隔 10s 计算一次 10s 内的 GPU 平均利用率。实验的结果如图 3 和表 4 所示，其中图 3 给出了 30min 内本文系统与原生 kubernetes 的 GPU 利用率变化情况。

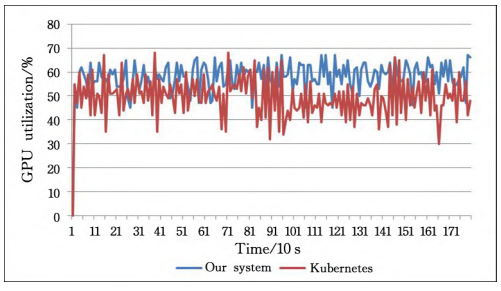

图 3 GPU 利用率对比情况

Fig. 3 Comparison of GPU utilization

表 4 GPU 利用率与任务完成时间

Table 4 GPU utilization and job completion time

	GPU Utilization	Completion Time
Kubernetes	49.71%	59 min 50 s
Our System	60.35%	48 min 10 s
Improvement	10.64%	19.49%

对比实验结果可以发现，与原生 Kubernetes 系统相比，本文系统的 GPU 利用率平均提升了 10.64%，任务总完成时间减少了 19.49%。

4.3 性能干扰与性能提升

本实验中每两个任务为一组，从模型列表的 5 种模型中随机选取模型和数据集启动深度学习训练任务。设置其中一个任务为高优先级任务，另一个任务为低优先级任务，测试采用本文系统和 Kubernetes 原生系统以及直接在主机中将两个任务在同一块 GPU 卡上运行任务时，高优先级任务相较于独占 GPU 时的性能损耗以及低优先级任务性能的提升。

对比图 4 和图 5 的实验结果可以发现，采用直接打包的方式将两个任务放置在同一张 GPU 上时，虽然相比原生 Kubernetes 系统与本系统方案，其任务总完成时间最短，但采用直接打包的方式运行时高优先级任务相比原生 Kubernetes 系统性能平均降低了 32.88%。故在多个任务共享使用 GPU 时，采用直接打包的方式无法保证高优先级任务的性能；并且，由于原生 Kubernetes 不支持共享 GPU，因此无法将直接打包的方式应用于基于 Kubernetes 的深度学习云平台中。而采用本系统方案，在兼容原生 Kubernetes 的同时，高优先级任务性能相比原生 Kubernetes 系统性能平均仅降低了 4.63%。同时本文系统的低优先级任务相比原生 Kubernetes 系统中该任务的性能提升约 20.79%。

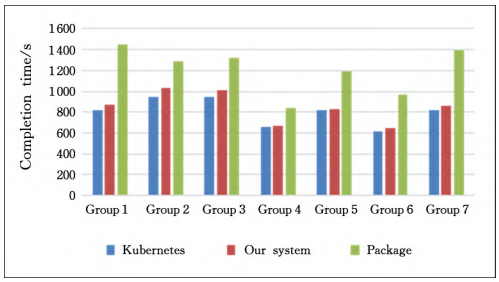

图 4 高优先级任务完成时间对比

Fig. 4 Completion time comparison of high priority jobs

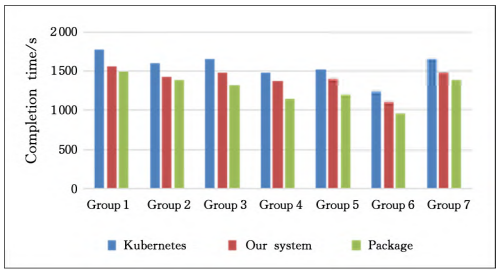

图 5 任务完成总时间对比

Fig. 5 Comparison of job completion time

4.4 系统的可扩展性

本系统基于 Kubernetes 的 Operator 自定义资源和控制器来支持 GPU 共享调度，使得多个任务容器能够共享 GPU。

Operator 是 Kubernetes 提供的用于扩展集群现有资源和功能的框架。在本文中，vPod 是自定义的资源，vPod 的调度器和管理器是自定义的控制器，通过自定义资源和控制器，我们能够在不修改 Kubernetes 源码以及不影响现有集群状态的情况下，实现 GPU 共享的功能。用户能够通过申请 vPod 并设置任务资源类型来使用共享 GPU，也能够直接通过 Kubernetes 申请独占 GPU 的 Pod。此外，本文系统将 vPod 调度器与管理器解耦成两个独立的模块，因此我们能够根据需求在调度器模块使用不同的调度算法。当集群节点或集群资源发生变化时，本文利用 Kubernetes 中的 DaemonSet 资源对象的特性，使得 GPU 资源调度模块能够根据节点数量动态伸缩，进而使得每个节点上都能运行 GPU 资源调度模块，以保证对各节点的可共享 GPU 的资源管理。由此可见，本系统具有较好的可扩展性。

结束语　针对基于 Kubernetes 的深度学习容器云平台 GPU 卡资源的独占问题，本文提出了一种 GPU 共享调度系统，该系统能够在保证兼容原生 Kubernetes 的前提下，实现将多个用户任务 Pod 调度到同一块 GPU 上，使得多个任务能够共享使用一块 GPU 卡资源。

本文系统基于 Kubernetes Operator 机制，创建自定义资源 vPod 并设计开发了自定义调度模块和资源管理模块，实现 GPU 共享调度和保证与原生 Kubernetes 调度方案的兼容性。针对 GPU 共享环境中任务之间的干扰问题，本文基于 GPU 时间片分片以及时间片抢占机制，设计了 GPU 资源动态管理模块，能够在多个任务之间进行细粒度的协调，并减少任务干扰。

实验结果表明，与原生 Kubernetes 调度系统相比，本系统能够将深度学习训练任务的完成时间平均减少 19.49%，集群 GPU 资源利用率平均提升 10.64%；同时，本系统能够在高优先级任务性能平均损耗 4.63%的情况下，将低优先级任务的性能平均提升 20.79%。

参 考 文 献

[1] JOSHI A V. Amazon's machine learning toolkit: Sagemaker [M] // Machine Learning and Artificial Intelligence. Cham: Springer, 2020: 233-243.

[2] BISONG E. Google colaboratory [M] // Building Machine Learning and Deep Learning Models on Google Cloud Platform. Berkeley: Apress, 2019: 59-64.

[3] MERKEL D. Docker: lightweight linux containers for consistent development and deployment [J]. Linux Journal, 2014, 239: 76-90.

[4] SAYFAN G. Mastering Kubernetes [M]. Birmingham: Packt Publishing, 2017.

[5] VAUCHER S, PIRES R, FELBER P, et al. SGX-Aware Container Orchestration for Heterogeneous Clusters [C] // 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS). Austria: Vienna, 2018: 730-741.

[6] NVIDIA Corporation. k8s-device-plugin [EB/OL]. https://github.com-/NVIDIA/k8s-device-plugin/blob/master-/READ-ME. md.

[7] HE K M, ZHANG X Y, REN S Q, et al. Deep residual learning for image recognition [C] // Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2016: 770-778.

[8] KANG D, JUN T J, KIM D, et al. ConVGPU: GPU Management Middleware in Container Based Virtualized Environment [C] // 2017 IEEE International Conference on Cluster Computing (CLUSTER). IEEE, 2017: 301-309.

[9] Alibaba Cloud. GPU Sharing Scheduler Extender in Kubernetes [EB/OL]. https://github.com/AliyunContainerService/gpushare-scheduler-extender.

[10] Intel. Intel device plugin for kubernetes [EB/OL]. https://github.com/intel-/intel-device-plugins-for-kubernetes.

[11] GU J, SONG S, LI Y, et al. GaiaGPU: Sharing GPUs in Container Clouds [C] // IEEE International Conference on Parallel Distributed Processing with Applications, Ubiquitous Computing Communications, Big Data Cloud Computing, Social Computing Networking, Sustainable Computing Communications. 2018: 469-476.

[12] YEH T, CHEN H, CHOU J. Kubeshare: A framework to manage gpus as first-class and shared resources in container cloud [C] // The 29th International Symposium on High-Performance Parallel and Distributed Computing. Sweden: Stockholm, 2020: 173-184.

[13] GREGORY M K, VANESSA S, MICHAEL W B. Singularity: Scientific containers for mobility of compute [J]. PLOS ONE, 2017, 12(5): 1-20.

[14] Kubernetes. Kubernetes Scheduler [EB/OL]. https://kubernetes.io-/docs/concepts/scheduling-eviction/kube-scheduler.

[15] PASZKE A, GROSS S, MASSA F, et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library [C] // Advances in Neural Information Processing Systems 32. 2019: 8024-8035.

[16] NVIDIA Corporation. NVIDIA Management Library (NVML) [EB/OL]. https://developer. nvidia. com/nvidia-management-library-nvml.

WANG Zhuang, born in 1997, postgraduate, is a student member of China Computer Federation. His main research interests include cloud computing and GPU virtualization.

WANG Pinghui, born in 1984, Ph. D, professor, Ph.D supervisor, is a member of China Computer Federation. His main research interests include mobile Internet security, network graph data mining and knowledge discovery.

（责任编辑：何杨）