Kubernetes 平台下国产 GPU 的并发容器共享

王 蝶[1] 高 泽[1] 韩昊翔[1] 赵雨虹[1] 胥 凌[1]
WANG Die GAO Ze HAN Haoxiang ZHAO Yuhong XU Ling

摘 要 随着技术的不断发展，机载智能系统正在向智能化发展。相较于虚拟机，轻量灵活且可伸缩的容器技术在安全性和性能均要求较高的机载智能应用中更适用。随着容器技术的不断发展，容器编排系统也迅速崛起，其中 Kubernetes 已成为主流的容器编排平台。然而目前 Kubernetes 在 GPU 管理方面存在不足，多个并发 GPU 容器无法共享 GPU 资源。考虑到日益增加的硬件自主可控需求，面向容器技术的国产 GPU 资源的共享迫在眉睫。针对上述问题，提出了一种名为 sharedGPU 的通用国产 GPU 共享方法，所提出的方法面向 Kubernetes 平台，实现了并发容器之间共享 GPU，同时保留容器独占 GPU 的使用模式。实验结果表明，sharedGPU 支持独占和共享两种使用 GPU 的模式。

关键词 机载系统；容器；Kubernetes；共享；国产 GPU

doi：10.3969/j.issn.1672-9528.2024.03.015

0 引言

随着云计算等技术的不断发展，机载系统将更加智能化 [1]。容器技术的成熟为机载智能应用提供了更为稳定和可靠的技术支持 [2-3]，其轻量、灵活、可伸缩的特性使得容器成为一种理想的应用部署方案。为了简化容器的部署、调度和管理，容器编排系统在近年来得到了迅速的发展，其中 Kubernetes[4-5]（简称 K8S）被广泛应用。

在人工智能、深度学习等领域，图形处理单元（GPU）[6] 相对于中央处理器（CPU）有更显著的性能优势，使得 GPU 成为重要的计算资源。因此，有效地管理和利用 GPU 资源成为一个重要的问题。目前，K8S 通过设备插件框架 [7] 支持对 GPU 的调度和管理，然而设备插件本身不支持 GPU 共享。这导致在 K8S 管理下，GPU 只能被单个容器独占使用。这种方式存在一个明显的不足，当容器中的应用实际使用 GPU 资源较少，导致剩余的 GPU 资源未得到利用，从而造成浪费。因此，如何在容器层面实现 GPU 的共享已经成为亟待解决的问题。

目前已有的面向容器的 GPU 共享方法都针对国外厂商的 GPU，如 Nvidia 和 AMD。这些 GPU 采用闭源的 CUDA 编程框架 [8]，已经具备完善的开发生态。随着国际形势的复杂多变，自主可控发展已经成为国家发展的重要战略 [9]。在这一背景下，机载系统中采用国产 GPU 的重要性逐渐凸显。国产 GPU 大多支持开源的 OpenCL 编程框架 [10]，其正处于高速发展阶段，但面向容器的共享技术研究相对有限。为了促使国产 GPU 在机载系统中得以广泛应用，实现容器层面的国产 GPU 共享已经成为一个值得深入研究的领域。

本文提出一种名为 SharedGPU 的 k8s 平台下的通用国产 GPU 共享方法，将一个 GPU 划分为多个虚拟计算资源，并能够动态地将虚拟计算资源分配给容器。SharedGPU 通过实时监控容器的 GPU 利用率，对其使用的 GPU 资源进行限制，从而实现容器间计算资源的有效共享；通过设计调度器确保高优先级容器得到足够的运行保障，对于重要任务和需求快速响应的任务具有重要意义，并提高系统的可靠性。

1 GPU 共享

随着 GPU 作为重要的计算资源被广泛应用，共享 GPU 的需求逐渐增加。与此同时，随着容器技术的日益成熟，应用被打包为容器运行已成为一种趋势。因此，针对容器的 GPU 共享技术展现出了广泛的前景。目前，业界和学术界已取得一些相关的研究成果。

阿里云提出的 cGPU[11] 通过实现一个内核模块，将 GPU 划分为多个虚拟设备，修改容器运行时挂载虚拟 GPU 设备。其借助用户态轻量的运行库，对容器内的虚拟 GPU 设备进行配置。Deepomatic[12] 用了一种分数分配的方法，允许对 GPU 进行分配，实现 GPU 资源的细粒度共享，但是它局限于单 GPU 节点。爱奇艺提出的 vGPU 采用空分复用的方法，将内核限制在 GPU 的部分 SM 上执行，这种方式需要将多个容器的上下文合并来达到共享的目的，使用场景很受限。阿里开源的 GPU Share[13] 基于 K8S 的设备插件将 GPU 显存划分为

1. 中国航空工业集团公司西安航空计算技术研究所
陕西西安 710000

多份虚拟设备以此来实现 GPU 的共享，但是它仅支持共享模式，不支持容器独占 GPU。Kang 等人提出的 ConvGPU[14] 实现了 GPU 显存的共享，通过对 CUDA 库进行拦截实现对容器使用的 GPU 显存进行限制，从而实现显存的资源隔离，但是它只能用于单个 GPU 的共享。

上述工作针对 NVIDIA GPU 实现，且大多依赖于 NVIDIA 容器运行，这导致其无法在国产 GPU 上运行。因此，本文实现了一种面向 k8s 平台的国产 GPU 共享方法，采用 API 重定向方式 [15] 对 OpenCL 库进行拦截实现 GPU 的共享。

2 SharedGPU 方法

2.1 总体概述

本文提出的 SharedGPU 方法整体架构如图 1 所示，包含资源共享、资源调度、虚拟资源服务、资源限制四个功能模块。资源共享模块中将一个 GPU 划分为多个虚拟计算资源，实现了 K8S 设备插件框架的接口。该模块主要完成虚拟计算资源的注册、上报更新和设备分配功能。资源调度模块通过 K8S 的扩展调度器框架实现了 GPU 容器的最优节点选择和 GPU 资源的有效分配。虚拟资源服务模块负责传递容器配置和保存资源限制模块容器所需的关键信息。资源限制模块包含了一个封装了 OpenCL 驱动 API 的拦截库，通过拦截应用程序对 OpenCL 驱动 API 的调用，实现 GPU 的共享，并在应用程序运行时有效地限制其 GPU 的利用率，实现容器间的资源隔离。

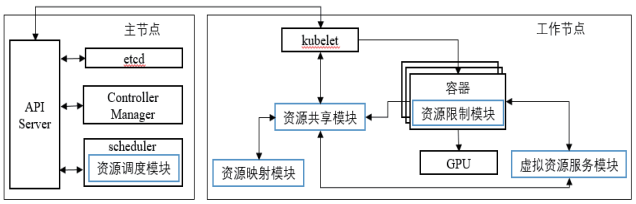


图 1 sharedGPU 整体架构图

sharedGPU 支持两种使用 GPU 的方式：独占和共享。在共享模式下，一个容器只能分配一个 GPU，即一个容器所请求的虚拟计算资源数量应在 0 ～ 100 的范围内，请求不能超过 100。在独占模式下，容器申请的虚拟计算资源数量必须为 100 的整倍数，如申请 200 个虚拟计算资源表示独占两个 GPU。

2.2 资源共享模块

资源共享模块基于设备插件框架实现，即其本质上是一个设备插件，该模块的主要功能包括将 GPU 资源划分为虚拟计算资源、注册虚拟计算资源、将虚拟计算资源信息上报给 kubelet、为容器分配虚拟计算资源等。通过这些功能，该模块能够有效地管理资源的分配和共享。

资源共享模块必须注册自身到节点的 Kubelet，以便与 Kubelet 进行通信。注册成功后，它会收集当前节点上所有可用的 GPU 设备信息并维护设备状态。不同于设备插件框架的默认功能，该模块将一个 GPU 划分为 100 个虚拟计算资源，并将虚拟计算资源列表上报给 Kubelet。

当容器申请了 vGPU 资源并被调度到当前节点时，Kubelet 会从虚拟计算资源列表中随机选择相应数量的设备进行分配。但这里分配的虚拟计算资源 ID 列表并无实际意义，这些 ID 没有真实对应的物理设备，需要将虚拟计算资源映射到 GPU 上，这个功能在资源调度模块中实现。资源调度模块首先通过为容器添加注解，表明该容器应该调度到当前节点的哪个 GPU 上运行，然后根据这个注解为容器挂载相应的 GPU 设备信息，最后资源共享模块会向 Kubelet 返回一个响应，包含了设置的 LD_PRELOAD 环境变量、GPU 设备信息和数据卷（主机上需要挂载到容器内的目录及文件）。

2.3 资源调度模块

资源调度模块基于 K8S 的扩展调度器实现为容器绑定最优节点和为应用分配 GPU 两个功能。

在预选阶段，根据用户提交的容器应用信息，筛选出符合要求的节点。对于高优先级容器，若某节点存在空闲 GPU，及时更新容器信息添加 GPU ID，并将节点加入候选列表后退出节点的遍历。若无空闲 GPU，则将运行低优先级且可用资源满足容器需求的 GPU 的节点并加入候选列表。对于低优先级容器，只需节点上有 GPU 可用资源满足需求即可加入候选列表。

在优选阶段，对预选的候选列表进行筛选，选择最合适的节点和 GPU。对于高优先级容器，若容器信息包含 GPU ID，则选择唯一的节点为最优节点。若无 GPU ID 信息，遍历候选列表为每个节点打分，考虑节点上可用资源越多，待调度容器被绑定的概率越大。对于低优先级容器，优先考虑调度到仅运行低优先级容器的节点，其次选择运行高优先级容器的节点，最后将最高分的节点绑定到待调度容器，更新选择的 GPU ID 到容器信息中，供资源共享模块使用。

为确保高优先级容器正常运行，若高优先级容器在预选阶段且候选节点列表为空，将触发压缩调度策略。该策略旨在通过对当前可用资源最多的 GPU 上运行的低优先级容器占用资源，根据式（1）和式（2）进行压缩，以满足待调度高优先级容器的需求。当 Q_{vD} 为 0，每个容器需要减少的资源为 R_{vD}。当 Q_{vD} 不为 0，对容器根据占用的虚拟资源由多到少进行排序，前 Q_{vD} 个容器需要减少的资源为 R_{vD}+1，其余容器需减少的资源为 R_{vD}。

$$Q_{vD} = (P_{vD} - G_{A_{vD}})\%N_G \quad (1)$$

$$R_{vD} = (P_{vD} - G_{A_{vD}})/N_G \quad (2)$$

式中：Q_{vD} 表示需减少的无法被 GPU 上已有容器平分的资源；P_{vD} 表示待调度容器申请的资源数；$G_{A_{vD}}$ 表示 GPU 上可用的

资源数；N_G 表示该 GPU 上正在运行的容器数量；R_{vD} 表示每个在运行的容器平均需减少的占用资源数。

2.4 虚拟资源服务模块

虚拟资源服务模块主要完成容器配置信息的传递，并提供资源限制模块所需的容器的 GPU 资源限制信息以及容器内的进程在主机上的进程号信息。

当一个容器请求 GPU 资源时，资源共享模块会将容器的配置信息发送给虚拟资源服务模块。虚拟资源服务模块为该容器在主机上创建一个唯一的目录，其路径包含该容器所属的 Pod UID。这个目录包含在资源共享模块返回给 Kubelet 的响应中，启动容器时会将该目录挂载到容器内，容器内部能够读取和保存与容器相关的信息。

为了确保系统资源的高效利用，清理处于非活跃状态的容器在主机上创建的目录，虚拟资源服务模块维护了一个容器列表，记录了所有处于活跃状态且已分配虚拟计算资源的容器。当容器处于非活跃状态时，该容器会从列表中移除，并相应地删除其主机上的目录。

虚拟资源服务模块为资源限制模块提供注册服务。资源限制模块在启动时向虚拟资源服务模块注册，并获得一个响应信息。在注册过程中，虚拟资源服务模块会将容器的 GPU 限制信息、容器中进程和在主机上的进程号信息保存到之前创建的目录文件中。如果该文件不存在，则会创建新的文件，如果存在，则刷新文件信息。最终，虚拟资源服务模块将文件中的信息包含在返回的响应信息中。

2.5 资源限制模块

资源共享模块中会为容器挂载 OpenCL 拦截库并设置 LD_PRELOAD 环境变量，使得拦截库能够在真正的 OpenCL 驱动库之前加载，拦截上层应用对 OpenCL 驱动库的调用。拦截库由资源限制模块生成，并在拦截库中实现资源的监控、限制和分配。

该模块中采用 API 重定向方式对 OpenCL 驱动库中的 API 进行二次封装。它在 API 调用路径上插入自定义逻辑，完成一些额外的功能后再调用真正 OpenCL 库的 API。资源限制方面采用了弹性资源限制的方法，即当系统有剩余资源时，即使容器的资源使用量超过了其请求量，仍然可以获得额外的资源分配。这种设计允许可以更加灵活地利用系统资源，从而提高容器的运行效率。

当容器中的 OpenCL 应用进行初始化时，资源限制模块并不知道容器所申请的 GPU 资源信息。因此，资源限制模块通过读取容器内的文件获得容器的关键信息。在容器运行过程中，根据容器在主机上的进程号通过 GPU 利用率监控函数获得应用的 GPU 利用率。当监测到 OpenCL 容器使用的计算资源超出限制时，对其资源进行限制；若未超出限制，则分配资源并将调用转发给真正的OpenCL驱动库进行计算。通过这个机制实现了对 OpenCL 容器的动态资源管理，确保了计算资源的合理分配和利用。

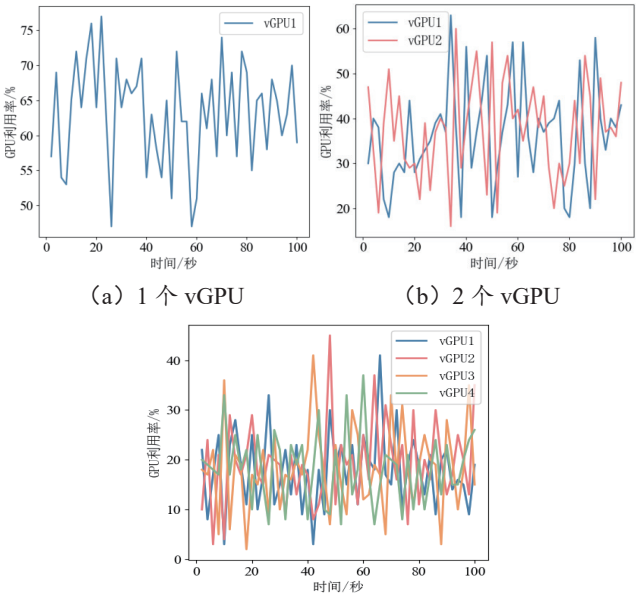
3 实验

实验设备采用汉为飞腾 D2000 开发板，搭载 Kylin V10 SP1 操作系统，配备芯动科技的风华 2 号 GPU。

表 1 vGPU 划分

vGPU 数量	1	2	4
每个 vGPU 占用的虚拟计算资源	100	50	25

为了验证 SharedGPU 的共享效果，并且支持独占 GPU 的模式，本文将一个 GPU 分别分为 1、2 和 4 个虚拟 GPU（vGPU）进行实验。表 1 表示不同 vGPU 数量下每个 vGPU 占用的虚拟计算资源。在每个vGPU上运行一个OpenCL容器，不同 vGPU 数量下容器的 GPU 利用率如图 2 所示。



（a）1 个 vGPU　（b）2 个 vGPU

（c）4 个 vGPU

图 2 各容器运行时的 GPU 利用率

在图 2 中，（a）表示独占 GPU 时容器的 GPU 利用率；（b）表示将一个 GPU 划分为 2 个 vGPU 时，两个并发容器其各自的 GPU 利用率情况；（c）表示将 GPU 划分为 4 个 vGPU 时，4 个并发容器其各自的 GPU 利用率情况。从这些结果中可以清晰看出，SharedGPU 实现了国产 GPU 的共享，并支持独占和共享两种模式下，GPU 资源的分配和管理。

为验证 SharedGPU 能够确保及时响应高优先级容器，进行了以下实验。将 GPU 划分为两个虚拟 GPU（vGPU），其中 vGPU1 占用的虚拟计算资源为 40，vGPU2 占用的虚拟计算资源为 60。低优先级任务 A 占用 vGPU1 运行，而高优先级容器 B 则申请了 70 个虚拟计算资源。此时，vGPU2 的虚拟计算资源无法满足 B 的需求。为了响应 B，系统会

将 vGPU1 占用的虚拟计算资源减少 10，并且将其划分给 vGPU2，然后将 B 调度到 vGPU2 上运行。

容器运行过程中的 GPU 利用率如图 3 所示，当高优先级容器 B 开始执行时（第 50 s），SharedGPU 有效降低了低优先级容器 A 对 GPU 资源的占用，这说明 SharedGPU 在实现对高优先级任务的运行保障方面表现出了可行性和有效性。

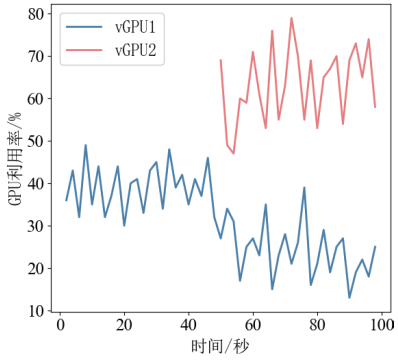


图 3 GPU 利用率

4 结论

SharedGPU 是一种专为 k8s 平台设计的国产 GPU 共享方法，其主要目标是实现并发容器共享同一个 GPU 的计算资源，避免空闲 GPU 计算资源的浪费。此外，SharedGPU 还能确保快速地响应高优先级任务，提高系统的灵活性。通过实验证明，SharedGPU 有效地实现了容器间 GPU 计算资源的共享，并为高优先级任务提供了可靠的运行支持。然而，本文的设计仍存在一些需要改进和优化的方面。

（1）用户对虚拟计算资源的指定：尽管用户在创建应用时可以指定需要申请的虚拟计算资源，但在实际情况中，用户可能难以准确确定应用所需的 GPU 资源。在这方面，可以探索引入自动化机制，根据应用的特性和需求，动态调整容器所申请的虚拟计算资源。

（2）自定义调度器的实现：需要确保其安全性，以防止恶意攻击或由错误导致的安全漏洞。后续工作应更全面地考虑和完善安全性方面的问题。

这些改进措施有望进一步提升 SharedGPU 的性能和用户友好性，使其更好地适应不同应用场景和硬件环境。这些调整将有助于推动 SharedGPU 在实际应用中的广泛采用。

参考文献：

[1] 方忆平，杨韧．国外航空机载系统发展历程及启示 [J]. 国防科技工业 ,2013(3):64-65.

[2] 马晓光，孙大军，吴登勇，等．容器虚拟化技术在飞腾 1500A 平台的应用 [J]. 信息技术与信息化 ,2017(6):39-41.

[3] 吴晗，王一凡，胡宁．基于容器的 j 分布式资源管理技术研究 [J]. 信息技术与信息化 ,2022(4):38-41.

[4] BERNSTEIN D. Containers and cloud: from lxc to docker to kubernetes[J]. IEEE cloud computing, 2014, 1(3): 81-84.

[5] Google. Kubernetes cluster management [EB/OL]. [2023-12-05]. http://kubernetes.io/.

[6] OWENS J D, LUEBKE D, GOVINDARAJU N, et al. A survey of general‐purpose computation on graphics hardware[J]. Computer graphics forum: journal of the european association for computer graphics,2007, 26(1): 80-113.

[7] NVIDIA C. K8s-device-plugin [EB/OL]. [2023-12-05]. https://github.com/NVIDIA/k8s-device-plugin.

[8] SANDERS J, KANDROT E. CUDA by example: an introduction to general-purpose GPU programming[M]. Boston:Addison-Wesley Professional,2010.

[9] 余茂琴，刘帅，陈辰，等．“十四五”时期电子信息产业发展趋势 [J]. 金陵科技学院学报，2020,36(4):27-30.

[10] 高胜寒，熊庭刚 .OpenCL 在国产 GPU 上的实现 [J]. 舰船电子工程 ,2021,41(9):113-116+125.

[11] ALIBABA Cloud. Install and use CGPU on a docker container [EB/OL].[2023-12-16].https://www.alibabacloud.com/help/zh/egs/developer-reference/install-and-use-cgpu-on-a-docker-container.

[12] Deepomatic. Shared-gpu-nvidia-k8s-device-plugin [EB/OL]. [2023-12-05].https://github.com/Deepomatic/shared-gpu-nvidia-k8s-device-plugin.

[13] ALIBABA Cloud.Gpushare-scheduler-extender [EB/OL]. [2023-11-29].https://github.com/AliyunContainerService/gpushare-scheduler-extender.

[14] 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). Piscataway:IEEE, 2017: 301-309.

[15] TANG D, LI L, MA J, et al. GRemote: cloud rendering on GPU resource pool based on API-forwarding[J]. Journal of systems architecture, 2021, 116: 102055.

【作者简介】

王蝶（1996—），女，陕西宝鸡人，硕士研究生，研究方向：高性能计算。

高泽（1994—），男，陕西延安人，硕士研究生，研究方向：高性能计算。

韩昊翔（1998—），男，陕西渭南人，本科，研究方向：高性能计算。

赵雨虹（2000—），女，河南洛阳人，本科，研究方向：高性能计算。

胥凌（1985—），男，陕西汉中人，博士，研究方向：计算机体系结构。

（收稿日期：2024-01-12）