gEdge:基于容器技术的云边协同的异构计算框架

汪 沄 汤冬劼 郭开诚 戚正伟 管海兵
(上海交通大学电子信息与电气工程学院 上海 200240)

摘 要 由于按需灵活配置、高可用性、高资源利用率等优点,云计算技术成为过去十年的主流计算范式.随着万物互联时代的到来,单独依赖云计算技术已经无法满足数以亿计的IoT设备及其数据流量的需求.边缘计算可以被看作是云计算的进化,它因5G网络和物联网的崛起而诞生.随着云游戏、VR技术以及人工智能技术在日常生活中的广泛运用,对计算资源的需求也在日渐增长.然而,受体积与功耗限制,处于边缘的节点设备算力较弱.本文提出了gEdge:一种基于容器技术的云边协同的异构计算框架.该框架通过GPU虚拟化技术,将云端的物理GPU资源分为多块虚拟GPU资源,按需为边缘节点提供GPU算力资源,并且对用户容器无感知.实验表明,使用gEdge框架使边缘节点使用的容器镜像体积降低了48.8%,容器启动时间降低了35.5%,平均相对运行速度提高了213%.

关键词 图形处理器;虚拟化技术;容器技术;边缘计算;云边协同
中图法分类号 TP18 **DOI号** 10.11897/SP.J.1016.2024.01883

gEdge: A Container-Based Cloud-Edge Collaboration Framework for Heterogeneous Computing

WANG Yun TANG Dong-Jie GUO Kai-Cheng QI Zheng-Wei GUAN Hai-Bing
(*School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, Shanghai* 200240)

Abstract The advantages of flexible on-demand provisioning, high availability, and high resource utilization have made cloud computing technology the dominant computing paradigm of the past decade. With the advent of the Internet of Everything era, relying on cloud computing technology alone can no longer meet the demands of hundreds of millions of IoT (Internet of Things) devices and their data traffic. Edge computing can be seen as an evolution of cloud computing, emerging from the rise of 5G networks and the IoT. With the widespread use of cloud gaming, VR (Virtual Reality) technology, and artificial intelligence technology in daily life, the demand for computing resources is growing day by day. Restricted by size, weight, and power, the node devices at the edge have weak computing resources. In this paper, we propose gEdge, a cloud-edge collaboration framework for heterogeneous computing based on container technology. The framework divides the physical GPU resources in the cloud into multiple virtual GPU resources, provides GPU compute resources for the edge nodes on demand, and is transparent to user containers, through GPU virtualization technology. Experiments show that the use of the gEdge framework can reduce the container image size used by edge nodes by 48.8%, container

收稿日期:2023-10-16;在线发布日期:2024-04-30.本课题得到国家自然科学基金(61732010,62141218)资助.**汪 沄**,博士研究生,中国计算机学会(CCF)学生会员,主要研究方向为云计算、GPU虚拟化.E-mail:yunwang94@sjtu.edu.cn.**汤冬劼**,博士,讲师,主要研究方向为云渲染与GPU虚拟化.**郭开诚**,博士研究生,中国计算机学会(CCF)学生会员,主要研究方向为机器学习系统、FPGA虚拟化.**戚正伟**,博士,教授,中国计算机学会(CCF)杰出会员,主要研究领域为系统软件、虚拟化、程序分析.**管海兵**,博士,教授,长江学者,国家杰出青年科学基金入选者,中国计算机学会(CCF)杰出会员,主要研究领域为虚拟化、云计算.

start-up time by 35.5%, and average relative running speed by 213%.

Keywords graphics processing units; virtualization; container; edge computing; cloud-edge collaboration

1 引 言

随着物联网技术在日常生活中的普及,智能物联网设备的数量和种类呈指数级增长.这一趋势不仅为人们的生活带来了巨大的便利,同时也产生了大量的原始数据.无论是普通用户所运行的常见应用程序(如增强现实、虚拟现实、云游戏等渲染任务),还是产业用户所使用的专业应用程序(如智慧城市、无人驾驶、智慧医疗等人工智能任务),均会产生庞大的数据,需要大量算力进行处理.根据思科公司2018~2023年度互联网报告①,到2023年,全球联网设备数量将达到293亿台,相较2018年的184亿台,增长率高达53.8%.此外,人均联网设备数量也将从2018年的2.4台增长至3.5台.这些设备带来的大量数据计算任务对数据处理的实时性和情景感知能力提出了一定要求.因此,处理海量边缘数据并提高数据吞吐量、降低延迟已成为未来互联网的核心挑战和关键性能指标.

与此同时,AWS、阿里云等公司在商业上的成功,使得云计算技术也作为当前最流行的计算范式被广泛采用.在传统的云计算模式中,计算资源全都集中于数据中心,用户设备产生的数据包括图片、视频、文本等通过网络传输至数据中心,利用数据中心的服务器集群进行处理.而在物联网设备海量增长的背景下,这样集中式的数据处理方式已经渐渐无法满足用户的实际需求[1-2].以装备了成百上千高清摄像设备的车站监控系统为例,为了保障公共场所的安全,需要全天候运行摄像头.采集到的视频信息如果不间断地同时传输至数据中心进行处理,数据带宽要求将远远超过当前广域网所能提供的上限.此外,时延对于增强现实(AR)和虚拟现实(VR)等应用场景具有重要意义.AR和VR设备由于其自身尺寸和体积的限制,内部渲染算力相对较低.在处理复杂场景的渲染过程中,可能会出现现实画面渲染增强与用户实际动作之间的可察觉时延.这种时延可能导致用户在与虚拟场景互动时,失去对虚拟环境的沉浸感[3-4].为了确保用户获得更为真实、流畅的虚拟体验,有必要在设计和优化AR、VR应用时充分利用到云边协同的特性,以降低时延并提升用户体验.

边缘计算是一个位于终端设备与云端数据中心之间的新计算层级.边缘计算是一种靠近数据源的计算范式.相比云计算,边缘计算的节点通常位于接近终端设备的位置,从而显著减少终端设备与边缘节点之间的通信延迟,并降低数据中心的入口流量.图1是常见的边缘计算架构,第一层是云端(Cloud),代表着位于云的数据中心服务器.第二层是边缘端(Edge),代表着处于靠近用户的边缘节点(EdgeNode).第三层是终端(Devices),代表着用户的终端设备.从图中的红色箭头可见,典型的从终端到边缘的往返时间(Round-Trip Time, RTT)低于10毫秒,这个数值远低于从终端到云计算环境25 ms的RTT[5].随着人工智能算法的不断演进,相关应用使用的模型也变得越来越复杂,使用了相关技术的应用程序对算力的要求也在不断提高.但是由于边缘节点的体积与规模的限制,它们的算力往往有限并且对能耗敏感,无法独自完成算力密集型的任务[6].因此,边缘节点通过利用云端服务器的GPU资源,特别是其大规模并行化的特点,可以显著提高数据处理能力.此外,边缘节点相较于云端服务器,在地理位置上更贴近终端设备,数据传输开销更低,从而拉近数据源头与数据处理节点的距离,降低数据处理的延时.

快速发展的轻量虚拟化技术,能够很好地满足边缘计算的可拓展性、多租户、灵活性等要求.容器技术是一种基于操作系统的轻量级虚拟化方案,相较于云计算中使用的硬件虚拟化技术,其开销更小.在边缘计算中,应用容器技术可以避免对于任何软件或者硬件的依赖.容器化后的应用程序,可以运行在云端的数据中心,也可以运行在边缘计算的节点上,能有效屏蔽云端与边缘节点的软硬件差异.由于上述特性,容器技术能提供良好的灵活性与可拓展性.

随着云计算逐渐向边缘计算演进,开发者面临

① Cisco Annual Internet Report (2018-2023) White Paper https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html 2022,7,2

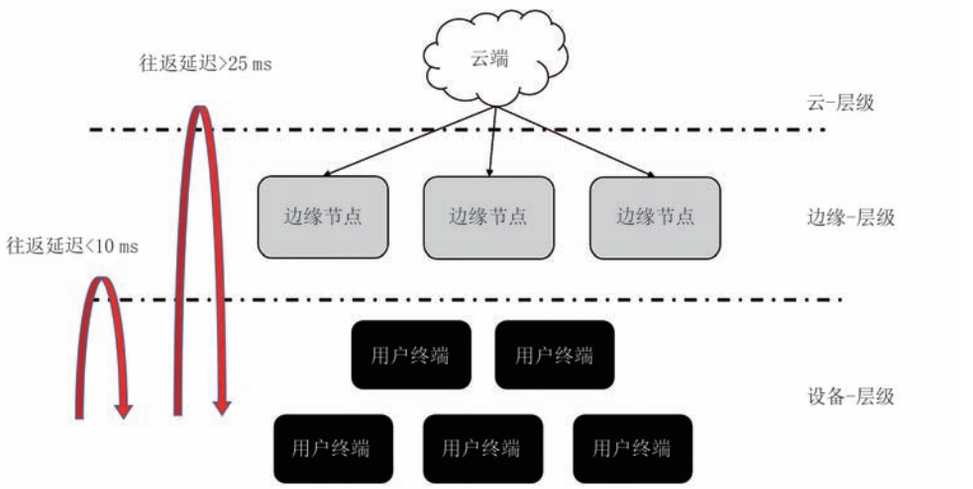

图1　常见的边缘计算架构图

着将原本在云服务器上运行的容器迁移到边缘节点的需求. 云服务器通常配备GPU以加速通用计算和图形渲染任务，而容器服务也依赖于GPU的硬件加速能力. 然而，直接将现有的容器部署至边缘节点很可能导致容器无法正常运行. 重新开发这些容器则需要投入相当的人力资源，从而大幅提高边缘系统的部署成本. 因此，在边缘节点上透明地部署依赖于GPU加速的容器成为了一个关键挑战. 现有的边缘计算框架主要采用将计算任务卸载至云端服务器的策略，借助GPU进行加速. 基于卸载思路的方案往往需要对原有的软件架构进行修改，从而无法充分利用现有的大量GPU容器镜像[2]. gEdge通过采用GPU虚拟化技术，使边缘节点能够远程访问位于数据中心的GPU资源，从而更快速地处理终端用户的数据请求. 此外，gEdge还能够复用现有的GPU容器镜像，降低人力开发成本.

我们的主要贡献归纳如下：

(1)首次在云边协同计算场景中提出基于容器技术的异构计算框架gEdge. gEdge支持将边缘节点上的GPU容器任务无缝卸载至云端服务器，借助云端强大的GPU资源实现计算加速，从而解决体积限制导致的边缘算力不足问题.

(2)在多租户的环境下，针对不同类型的GPU工作负载，提出了应用感知的资源隔离策略，为边缘节点上的容器提供良好的资源隔离性，提高了云端GPU设备的资源利用率.

(3)针对边缘节点资源受限的问题，提出边缘友好的容器裁剪方案，降低了容器镜像的大小，提高了容器启动速度，提升了运行容器的密度.

(4)本文通过实验验证了gEdge框架的性能. 结果表明，在充分考虑卸载时延的情况下，gEdge框架实现了相对运行速度平均213%的性能提升. 同时，实验还展示了在多租户场景下，gEdge能有效地为运行的容器提供优良的资源隔离.

综上所述，gEdge通过透明卸载，在边缘设备和云端之间分配计算任务，实现负载均衡，充分利用边缘设备的计算能力，减轻云端服务器的负担. 其次，通过应用敏感的资源隔离方法，提供了良好的隔离性，确保了任务之间的公平性. 最后，利用了边缘友好的容器方案，提供了易用性和可扩展性，方便开发者在不同的边缘设备和云端环境中快速部署和扩展应用. gEdge将代码开源在Github①.

2　相关工作

本章我们首先在2.1节介绍容器技术以及其在边缘计算中的常见运用；2.2节中介绍目前常见的几种GPU虚拟化技术.

2.1　容器技术在边缘计算中的应用

容器(Container)是将代码及其所有依赖项打包成标准软件单元的技术，使应用程序能够在不同计算环境中可靠地运行. 单个容器实例以一个独立进程的形式运行在宿主机操作系统的用户空间中，而操作系统的内核则由所有的容器共享[7]. 由于不需要硬件和驱动程序的虚拟化，加上共享内核的特点，容器技术可以产生更小的磁盘镜像，实现更高的实例密度，在需要高速卸载和快速迁移服务的边缘计

① gEdge-Container-Based Heterogeneous Computing Framework for Edge-Cloud Collaboration, https://github.com/Wintel/gEdge-GPU

算中越来越受欢迎[8].

在以往的研究中，Yousefpour等人[9]曾开发一个名为FogPlan的容器化框架，将物联网设备与云端和边缘节点整合在一起，以尽量减少物联网应用的响应时间. FogPlan支持物联网应用的动态资源发现和调度，然而，它没有提供任何可扩展性机制和政策. Merlino等人[10]则开发了一个基于容器的框架，用于在云端和边缘发现容器，并支持水平和垂直卸载. 然而，它没有为物联网应用的动态调度和资源的可扩展性提供任何政策. An等人[11]开发了EiF框架，将人工智能服务带到网络的边缘. 尽管EiF为网络资源提供了一些资源分配技术，它并没有为物联网应用提供任何调度和可扩展性机制. Yigitoglu等人[12]开发了一个支持容器的Foggy框架，支持动态调度具有依赖性任务的容器化物联网应用. Bellavista等人[13]曾提出一个集中式的容器启用框架，使用Docker容器和Kubernetes来扩展计算基础设施，但它没有提供任何政策来支持水平扩展、调度和资源发现. 此外，由于云协调器管理应用程序的部署，它可能对延迟敏感的物联网应用程序的响应时间产生负面影响. Noor等人[14]开发了一个集中式容器支持的IoTDoc框架来管理物联网设备和云资源之间的互动. Tong等人[15-16]提出了一种名为H-container的框架，该框架基于LLVM编译框架，旨在解决边缘环境中由于不同设备指令架构差异所导致的容器实时迁移问题. Thomas等人[17]针对边缘系统上运行的数据密集型应用，提出了一种创新的启发式调度方案，从而显著提高调度器任务分配的质量，并在最小化任务执行时间等方面实现显著优化. 杨等人[18]则提出了功能分发网络(Function Delivery Network，FDN)，为用户提供了访问边缘计算资源的统一接口和容器化计算平台. FDN在优化资源利用和计算延迟的同时，采用启发式容器编排策略实现跨集群功能，降低任务计算延迟，并在与传统算法相比时展现出显著的性能提升. 对边缘计算环境中容器启动延迟问题，Lou等人[19]提出了一种名为容器分配与层序列化的框架，通过优化容器分配和分层下载顺序，显著减少了容器启动所需的时间. 此外，他们还提出了一种层感知调度算法，该算法有效利用了镜像层的共享特性，进一步降低了启动延迟.

上述研究主要是针对于容器在边缘环境的应用. 但是多数框架缺乏有效的可扩展性机制和政策，无法适应快速变化的网络环境和不断增长的设备数量. 此外，它们通常没有提供全面的调度和资源发现策略，特别是在处理边缘计算环境下物联网应用的动态性和多样性时. 还有，一些框架如通过集中式的容器启用策略来扩展计算基础设施的方法，可能会对延迟敏感的应用产生负面影响，因为这种策略忽略了边缘计算所追求的低延迟和高响应速度的核心要求. 总的来说，尽管这些工作在将容器技术应用于边缘计算和物联网领域方面取得了初步进展，但在可扩展性、动态调度策略及低延迟资源管理方面仍需进一步的优化和改进. 相较于前述研究，gEdge框架主要针对边缘节点计算能力不足的问题，采用GPU虚拟化技术将任务无缝卸载至云端，从而加速边缘侧任务执行过程，并提高容器在边缘侧的运行密度.

2.2 GPU虚拟化

如图2所示，根据实现原理和层级的不同，GPU虚拟化技术主要可以分成四类：

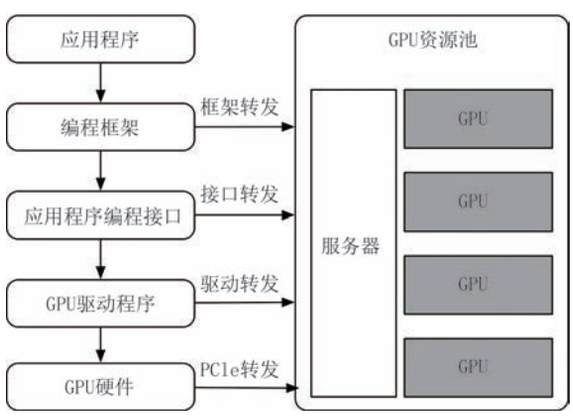

图2 GPU虚拟化方案

(1)框架转发的虚拟化：涉及对常见编程框架的转发虚拟化处理. 例如，AVEC[20]通过修改流行的深度学习框架Caffe，将边缘端训练任务卸载至远程GPU服务器. Neurosurgeon[21]进一步分析现有DNN任务网络特点，并将DNN网络分层卸载至远端进行训练，从而加速边缘计算并降低能耗. fGrain[22]基于Tensorflow提供了细粒度的CNN加速.

(2)硬件支持的虚拟化：通过硬件层设备直通实现虚拟化，客户机可直接访问GPU，利用主板芯片或GPU供应商的硬件扩展. 此种GPU设备直通通过DMA和中断重新映射至各客户机实现. 2006年出现的Intel VT-d[23]和AMD-Vi[24]是典型的硬件虚拟化代表，但主要用于设备直通，不支持多客户机共享GPU. 随后，支持复用的NVIDIA GRID[25]实现

多客户机共享单设备. NVIDIA Multi-Instance GPU(MIG)是一种将单个GPU设备划分成较小、独立的实例的技术[26]，每个实例具有各自独立的高速缓存、流多处理器和内存资源. 硬件虚拟化技术无需额外软件层，性能接近本机，但由于跳过宿主机操作系统，实施GPU调度策略较困难.

(3)驱动层虚拟化：通过驱动层提供虚拟GPU设备实现虚拟化. 借助定制GPU驱动，虚拟机访问虚拟GPU使用物理GPU资源. 全虚拟化无需修改GPU驱动，而类虚拟化需调整GPU驱动，将敏感指令传递给宿主机提高性能. Tian等人[27]提出的gVirt采用全虚拟化技术. 此类GPU虚拟化技术通常依赖于GPU供应商提供的官方文档和设备驱动的源代码.

(4)API重定向的虚拟化：在软件库层实现GPU虚拟化. 在GPU供应商对设备驱动闭源的情况下，API(Application Programming Interface，应用程序接口)重定向由于其不依赖于GPU硬件和系统虚拟化环境等特点成为了业界最受欢迎的GPU虚拟化方案. API重定向的前提是为客户操作系统提供一个封装库，该封装库具有与原始GPU库相同的API. 封装库在应用程序的调用到达客户操作系统的GPU驱动程序之前拦截GPU调用，被拦截的调用随后会被重定向到宿主操作系统或远程的带有GPU设备的机器进行计算，最终结果通过封装库传递给应用程序. rCUDA[28]、gRemote[29]和gVirtuS[30]均采用了接口转发技术，在特定场景下表现出了良好的性能.

如表1所示，我们将gEdge与当前的主流GPU虚拟化方案进行了功能性对比. gEdge采用了基于API重定向的虚拟化方案，与前述技术相比，它具有以下优势：(1)不依赖特定计算框架，具备更强通用性与兼容性；(2)拥有硬件独立性，与硬件厂商与设备解耦，具有更强的通用性；(3)重定向过程中便于按需调度负载，具备更强灵活性. gEdge不仅支持通用计算场景下的GPU资源虚拟化，还支持渲染场景下的计算加速，并针对边缘场景进行专门的密度优化.

表1　GPU虚拟化方案比较

	硬件独立	软件无侵入	远程卸载	通用计算	图像渲染
AVEC	√		√	√	
Neurosurgeon	√		√	√	
IntelVT-d		√		√	√
AMDVi		√		√	√
NVIDIAGRID/MIG		√		√	√
gVirt		√		√	√
rCUDA	√	√	√	√	
gRemote	√	√	√		√
gVirtuS	√	√	√	√	
gEdge	√	√	√	√	√

3　gEdge设计与实现

我们提出了gEdge，一种基于容器技术的云边协同异构计算框架. 其核心特点如下：(1)资源扩展：gEdge为边缘集群中的节点分配虚拟GPU(vGPU)设备，增强其计算能力. 这一设计有助于减小边缘节点的响应延迟，从而实现更快速的数据处理和响应；(2)资源隔离：在gEdge框架中，每个vGPU设备都设有各自的资源限制. 这种策略使得物理GPU资源得以划分与隔离，确保每个vGPU设备在运行过程中不会与其他设备产生资源竞争，保障系统的稳定性；(3)运行时资源裁剪：考虑到边缘节点的特性，gEdge采用了一种与容器技术相结合的边缘友好的裁剪技术. 这一技术能够有效地加快容器的启动速度，同时减小容器镜像体积，提高整体运行效率；(4)性能加速：通过以上设计，gEdge框架在异构计算环境中为边缘节点提供性能加速，有助于实现更高效的云边协同. gEdge通过裁剪容器运行时，显著减少了应用容器的启动延迟，有效弥补了网络传输导致的10毫秒延迟. 这使得对延迟敏感的应用程序也能在gEdge平台上实现性能提升. gEdge基于API重定向技术，与厂商硬件型号解耦，可以兼容运行在最新的硬件平台. 这一框架旨在解决传统边缘计算中性能瓶颈问题，提升边缘计算的可扩展性和可靠性.

3.1　gEdge的架构

作为一种基于容器技术的边缘-云协同GPU虚拟化方案，gEdge主要包含四个组件，即gEdge容器运行时(gEdge Container Runtime)、资源管理器(Resource Manager)、vGPU设备以及gEdge后端(gEdge Backend).

图3展示了gEdge在端-边-云的典型三层计算架构中的部署情况. 位于第一层的物联网设备终端，包括智能可穿戴手表、自动驾驶汽车、虚拟现实(VR)和增强现实(AR)设备等，主要负责采集各类数据，如图像、音频和视频等. 第二层为边缘节点，部署有gEdge容器运行时、vGPU资源管理器和

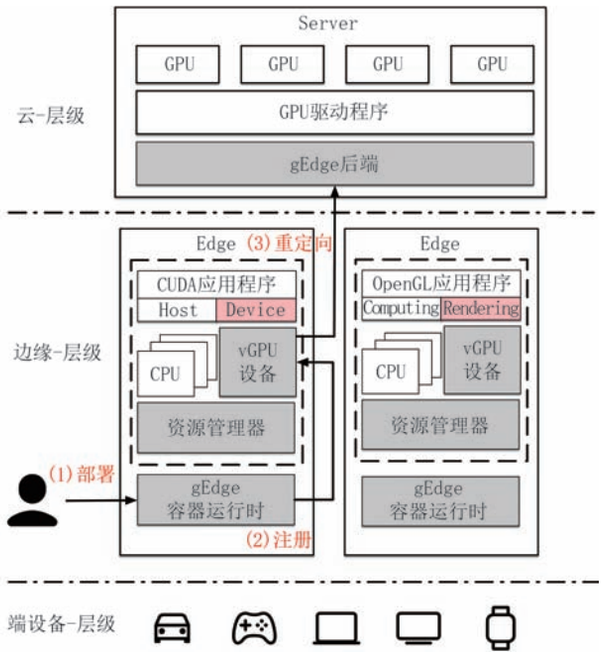

图3 gEdge架构图

vGPU设备组件. 其中,gEdge容器运行时使得边缘节点上的容器能够透明地利用远程GPU计算资源. vGPU资源管理器负责管理节点上运行的容器所需的vGPU资源. 位于容器内的vGPU设备组件为容器内应用程序提供虚拟GPU设备. 第三层为云服务器,部署有gEdge后端,主要负责管理和调度服务器端的物理GPU资源. 首先,用户的各种应用程序通过容器镜像的形式,利用gEdge容器运行时部署在边缘端,涵盖VR渲染、AI图像识别等场景. 这些过程中所使用的镜像与Docker镜像兼容. 接下来,gEdge容器运行时会通过启动钩子(Pre-Start Hook)的方法,将vGPU设备组件注册到容器的用户态文件系统中,完成虚拟GPU设备的配置,确保所有GPU调用都能通过该组件重定向到远端资源. 最后,位于云端的gEdge后端接收来自vGPU设备组件的GPU调用请求,并利用云端的物理GPU资源执行加速处理. gEdge各组件的具体实现和原理将在后续章节详细介绍.

传统的端-边-云部署移动计算处理方案,主要有两类:(1)云端主导型. 在这种方案中,用户数据首先传输到边缘服务器,然后再到云端进行处理. 然而,受网络带宽的制约,计算性能可能受到影响. (2)边缘端主导型. 在此方案下,用户数据传输至边缘端,同时应用程序也部署在边缘端. 所有计算都在边缘服务器上完成. 但是,由于边缘服务器的算力资源限制,计算性能有限,导致性能表现不尽理想.

如图4所示,gEdge采用了(3)云边协同的方案,利用了部署与计算分离的思想,将用户采集的数据传输至边缘端,在边缘端的用户程序进行预处理后,将模型数据传输至云端. 这种设计既缩短了应用程序与终端设备之间的距离,又充分利用了云端GPU的计算性能.

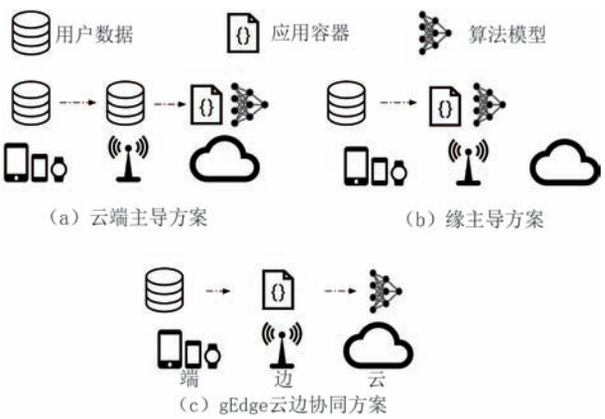

图4 移动计算任务的处理方法

异构计算通过结合不同类型的处理器或计算资源来提升性能、效率和计算能力. gEge框架实现异构计算主要是通过协调边缘端的CPU资源与云端的GPU资源. CPU在执行复杂的逻辑和控制任务方面表现出色,而GPU则擅长并行数据处理和图形计算. 将特定任务分配给最适合的处理器,异构计算能够实现超越传统单处理器系统的高效率和性能. 在边缘计算场景中,合理地将任务分配到适当的计算节点,还可以实现资源的最大化利用.

对于任意的异构计算任务,可以基于GPU编程的模型的特点,将程序分为宿主代码(Host Code)与设备代码(Device Code). 根据云-边-端的特点,gEdge对任务进行以下划分:

(1)端侧执行:端侧主要负责数据采集工作. 由于它与数据源(例如传感器、用户设备等)距离最近,因此将数据采集放在端侧可以减少数据传输延迟,同时可以利用端侧设备的处理能力进行初步的数据预处理和过滤.

(2)边缘端执行:边缘端主要运行宿主代码. 这部分代码通常涉及到任务的管理、控制逻辑以及数据的初步处理. 边缘计算可以提供比端侧更强大的计算能力,同时相比云端有更低的延迟,适合处理需要较快响应时间的任务.

(3)云端执行:云端则专注于执行设备代码,特

别是利用GPU硬件进行的加速计算．云端的强大计算能力适合处理需要大规模数据处理和复杂计算的任务，比如深度学习模型的训练和大数据分析．

通过这种方式，云-边-端三方的计算资源可以协同工作，以实现最高的性能加速．

3.1.1　gEdge容器运行时

为了在gEdge框架下无缝运行容器应用程序，我们设计了一个定制的容器运行时．考虑到与现有容器计算平台的兼容性，gEdge运行时基于Docker设计，并支持其操作．运行在gEdge运行时上的容器可以透明地卸载至云端加速．如图5所示，用户首先拉取指定镜像．在启动容器时，用户可以通过fGPU和vMemory参数设定运行要求．其中，fGPU的值指的是分配给容器可使用的vGPU数量，vMemory则是分配给容器的显存大小．

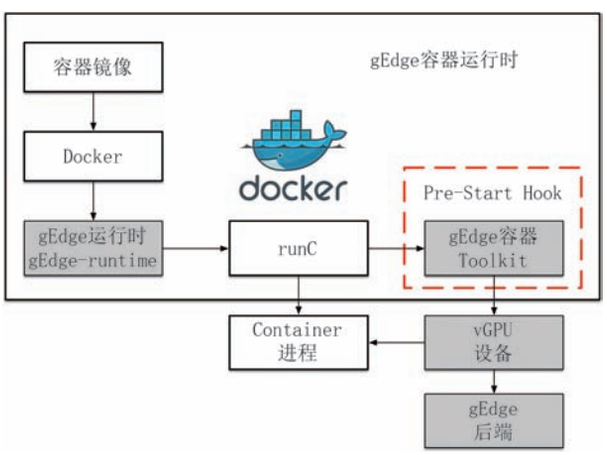

图5　gEdge容器运行时流程图

软件无侵入：为了确保与现有应用的兼容性，gEdge-runtime会自动生成配置文件并传递给runC．runC是一个基于开放容器标准（Open Container Initiative，OCI）的命令行工具，用于生成Linux容器．OCI规范提供了Pre-Start的Hook机制，允许容器在启动前运行自定义代码．gEdge容器运行时将gEdge容器工具包（以下简称Toolkit）注册为runC的Pre-Start Hook①．通过利用这一机制，gEdge可以在用户代码运行之前完成对边缘容器的环境设置，从而确保容器高效启动．这个机制也使用户的容器镜像无需任何修改，达到软件无侵入的效果．

Toolkit会根据gEdge-runtime生成的vGPU与vMemory配置，将gEdge所需的vGPU组件挂载至容器实际进程的文件系统命名空间内．随后，runC开始运行用户指定的容器镜像，并利用vGPU提供的虚拟GPU进行硬件加速．利用Toolkit，gEdge成功地实现了算力的透明卸载，使得原本需要物理GPU设备才能正常运行的容器，现在可以在边缘设备没有GPU的环境下正常运行．

3.1.2　vGPU资源管理器

vGPU资源管理器（vGPUResourceManger，后文简称为资源管理器）的用途是监控与管理容器使用的虚拟GPU资源．它运行在容器的内部，对容器内使用GPU资源的进程进行管理．资源管理器在容器启动时并不会立即向云端的gEdge服务器申请物理GPU资源，而是采取按需申请的策略．之前研究发现[31-32]，大多数容器在启动的时候，会做大量的准备工作，这部分工作不需要GPU资源的参与，可以通过边缘节点的算力资源来完成．只在容器内的程序真正需要GPU加速时，资源管理器才会向云端申请GPU．

3.1.3　vGPU设备

容器内的vGPU组件为应用程序提供虚拟GPU资源．GPU设备的驱动分为两个模块，分别为用户态（User-mode）驱动与内核态（Kernel-mode）驱动．其中，内核态的驱动与内核版本紧密耦合．但是根据容器的特点，运行在同一台机器上的容器共享宿主机的内核．因此所有的GPU容器都会使用宿主机提供的GPU内核态驱动，而在容器内只会携带用户态的设备驱动．

远程卸载：gEdge独特地利用了共享的内核态驱动机制，通过将vGPU组件注册为容器的用户态GPU驱动，成功在容器内实现了完整GPU功能的支持．在容器内的所有GPU设备操作都经由vGPU组件层进行中转，而vGPU组件则将这些操作精妙地重定向至位于云端的gEdge服务器，以借助服务器端的物理GPU设备来进行相应的处理和加速．

3.1.4　gEdge后端

gEdge后端（gEdgeBackend，后文简称为后端）运行在云端服务器上，它的主要作用是对物理GPU资源进行调度，向GPU提交实际的任务负载，并将计算结果传回相对应的vGPU容器．

根据GPU任务的特性，我们将其分为两个主要类别：一是渲染型任务，以AR（增强现实）、云游戏等为代表；二是通用计算型任务，以语音识别、图像识别等为代表．通用计算型任务的计算输出结果通

① Open Container Initiative Runtime Specification，https://github.com/opencontainers/runtime-spec/blob/v1.0.0/config.md#hooks 2022，7，2

常远小于其输入数据，因此后端处理往往不需要对其计算结果进行特殊处理．然而，对于渲染型任务，其计算输出结果是渲染后的动画场景．在云端与边缘端之间传输未经处理的像素信息会消耗大量带宽，可能对使用同一服务器的其他vGPU实例造成不利影响．

图像渲染：后端采用了回传优化策略．通过应用编码技术，如H.264，我们将原始像素信息以视频流的形式传回vGPU实例，从而有效降低整个传输链路上的带宽消耗．对于渲染型任务，我们在服务器端采取了一种优化方案，即在GPU渲染后的图像上应用视频编码，然后将其传回边缘设备．这一策略尤其适用于边缘设备带宽有限、数据处理能力较弱的情况，因为它在带宽受限的环境下仍能保持出色的图像质量．渲染型任务在后端的主要开销包括使用GPU进行渲染和使用CPU进行图像编码．

云游戏是指通过云计算技术将游戏运算处理和存储等工作都放在云端，用户可以在各种终端（如PC、平板电脑、智能手机等）上通过网络连接实现游戏的在线体验．与云游戏不同，gEdge将应用程序部署在边缘端，能够充分利用边缘端与终端距离相近的优点，此外通过转发命令的方式同样能够利用云端的硬件加速．这样的方式使得gEdge能够达到更低的延迟与更合理的CPU-GPU资源使用分布．

3.2 gEdge的GPU资源隔离

GPU的工作负载主要分为两类，一类是以Vulkan与OpenGL接口为代表的渲染型任务，与之对应的是AR与VR应用程序．另一类是以CUDA与OpenCL接口为代表的通用计算型任务，与之对应的是科学计算与图像识别等应用程序．gEdge对这两类任务采取应用敏感的资源隔离策略，在保证资源隔离的前提下，达到最佳的服务质量（Quality of Service，QoS）．对于不同类型的任务，资源隔离的要求不同．通用计算型任务可以通过限制容器使用的算力和显存（QoS指标）来实现GPU隔离性，即用户的资源使用率．对于渲染型任务，达到30FPS（QoS指标）是重要指标，固定分配的系统资源无法保证该指标，因此gEdge以渲染型任务的性能表现来保证隔离性．两者统一为QoS指标，gEdge通过保证客户的QoS，确保不同gEdge客户不会产生干扰与冲突，从而实现GPU资源隔离．

3.2.1 通用计算型任务

当用户在边缘节点部署通用计算型任务时，可以指定相应的fGPU参数与vMemory参数．其中，fGPU代表整块GPU算力的百分比，它是一个浮点参数，大小在0-1之间．vMemory代表的是显存大小参数，单位为256 MB，即1单位vMemory代表的是256 MB的显存资源．相应的参数会被gEdge运行时解析，并由gEdge后端进行相应的映射，从而生成对应的配置传递给容器内的vGPU组件．fGPU参数与vMemory参数取决于用户容器任务与后端实际硬件的算力，不同场景下，参数的取值会发生变化．因此gEdge采用了一种基于性能分析的参数选取方案（Profile Guided Parameter Selection）．在初次运行用户容器时，将物理GPU独占分配给该任务，gEdge后端会对用户容器的资源使用情况进行记录，如最高显存使用量、运行时间等．基于该全量运行的性能分析，gEdge会给出该任务的最小fGPU与vMemory参数（基于平均算力与显存用量），用户可以根据该数值进行动态调整．

对于显存与算力的隔离，我们以指令拦截的方式实现．以cudaMemAlloc程序调用为例，当容器内的应用程序使用cudaMemAlloc向vGPU申请一块内存时，vGPU会通过比较其运行时指定的vMemory参数以及当前已使用的显存信息，决定是否向云端服务器申请相对应的内存资源．对于算力而言，也采用同样的方式．

由于GPU的非抢占特性，一个任务提交给硬件执行后，只有当其全部完成以后，该任务使用的计算资源才会被释放．所以对单个容器而言，gEdge采用静态的资源分配策略，即容器运行前就需要决定其使用的GPU资源，在容器运行时无法弹性地使用超出其规定的资源．如果采用了弹性的资源分配策略，在多租户的情况下，恶意用户就能够通过简单的循环语句消耗所有的计算资源，并且无法得到有效的防御．

3.2.2 渲染型任务

与通用计算任务不同，多种因素如模型、风格和光线影响渲染型任务的性能．通过指定几个简单的资源参数无法有效保证渲染型任务达到一致的性能表现．不同应用程序，甚至相同应用程序的不同场景达到QoS要求时，需要的GPU资源也有所不同．此外，渲染型任务通常接收如用户动作或环境画面等实时连续输入，并需要实时输出．利用这种特性，我们可以调节输入速度来控制输出．

在面向渲染任务的场景中，帧率被视为一项关键的性能指标．低帧率可能导致用户在视觉上感受到画面的不流畅，从而降低终端用户的体验质量．

当帧率超过特定阈值后，用户通常无法感知其差异．过多的渲染帧画面则可能导致图形处理器(GPU)资源的浪费．因此，gEdge引入了一种基于启发式的动态资源调度算法．

算法1遵循强化学习策略，旨在通过不断互动和反馈学习如何最优地分配GPU资源以满足服务质量(QoS)指标．为预测GPU任务所需的资源，我们采用了基于Braun等人[33]提出的方法，确保了更准确的资源估计．通过神经网络的预测，算法为每个容器选择增加资源、减少资源或迁移的最佳动作．当容器的帧率低于QoS要求时，算法会采取相应的动作来调整其资源．算法中的步骤3-14描述了如何基于当前状态和Q-network的输出选择和执行最佳动作．步骤15-17则处理经验回放和Q-network的训练，确保随着时间的推移，算法能够更好地学习如何分配资源．最后，算法返回需要调整和迁移的容器集合，从而确保系统内的所有容器都满足QoS标准．

算法1.　基于DQN的动态资源调度算法

输入：当前时刻t；期望帧率v_{Qos}；当前容器数量n；当前容器集合C；容器利用率$U_i(i=1,2,\cdots,n)$；容器帧率$fps_i(i=1,2,\cdots,n)$；Q网络*Q-Network*；经验回放D；

输出：在t时刻*需要迁移的容器*T

1. Initialize replay memory D to capacity N
2. Initialize Q-network with random weights
3. FOR each $c_i \in C$ do
4. current_state = GetState(fps_i, U_i)
5. best_action = ChooseAction(current_state, Q-table, ε-greedy)
6. IF best_action == "IncreaseResource" THEN
7. Increase_GPU_Res(c_i)
8. ELSEIF best_action=="DecreaseResource" THEN
9. Reduce_GPU_Res(c_i)
10. ELSEIF best_action == "Migrate" THEN
11. $c_m \leftarrow$FindBestMigrationTarget(c_i, Q-table)
12. Migrate(c_i, c_m)
13. $T = T \cup \{c_i, c_m\}$
14. ENDIF
15. new_state = GetStateAfterAction(best_action, c_i)
16. reward = CalculateReward(fps_i, v_{Qos})
17. UpdateQTable(Q-table, current_state, best_action, reward, new_state)
18. ENDFOR
19. RETURNT

在算法1中，ChooseAction()函数(步骤5)利用了基于ε-greedy策略的动作选择机制，确保算法在大部分时间内采取最佳动作，但偶尔也会进行探索．此函数的时间复杂度为$O(1)$．FindBestMigrationTarget()函数(步骤11)使用Q-network为每个可能的迁移目标预测Q-values，从而选择最佳的迁移目标．这涉及到对所有可能的目标进行评估，因此其时间复杂度为$O(n)$．

由于算法1需要对数量为n的容器集合进行遍历并为每个容器评估所有可能的迁移目标，因此算法的总时间复杂度为$O(n^2)$．但由于在实际的边缘计算环境中，单个边缘节点上的容器数量n通常是可管理的，算法1的时间效率仍然是可接受的．

由于算法使用了神经网络和经验回放存储，其空间复杂度受到这两个组件的影响．但在实际应用中，这些组件的大小是固定的，因此算法1的空间复杂度可以视为$O(1)$．这确保了算法不会对系统产生过大的内存压力．

3.3　边缘友好的容器裁剪方案

容器的冷启动流程，首先是从公有或者私有的镜像仓库(container registry)拉取用户指定的压缩镜像文件，拉取成功后进行本地解压，最后使用解压后的镜像文件作为运行容器的根文件系统启动容器．

用户指定的容器镜像是由用户编写的Dockerfile文件编译生成的，在3.1.3小节中我们观察到由于GPU容器的特点，GPU设备的用户态驱动需要随着容器镜像一同打包，并在运行时加载，以达到使用本地GPU设备的目的．故GPU厂商往往会提供GPU设备的基础镜像(Baseimage)给用户们使用，基础镜像也包括了用户态的GPU驱动文件．gEdge通过Pre-Hook的机制，绕过了容器镜像内的本身的用户态驱动，为容器内的应用提供了vGPU设备进行加速计算．即使用gEdge框架后，基础镜像的用户态文件不会被使用到，但是仍然会在容器的冷启动过程中，被拉取并加载到根文件系统中．所以在带宽与存储都可能成为瓶颈的边缘环境下，我们提出了使用gEdge基础镜像替换厂商提供的基础镜像的方式，对容器的体积进行优化．用户只需要在Dockerfile中将基础镜像改为gEdge镜像，即可将生产镜像的基础镜像改为gEdge基础镜像．更换基础镜像后，用户的实际镜像不会包含硬件相关的驱动文件，从而减少了容器镜像的大小．容器镜像体积的减少，意味着冷启动过程中拉取镜像时间也随之减少，进而加快冷启动速度．

与Lou等人[19]的分层感知调度算法相比，我们

也利用了容器镜像分层的特点. 但是我们的容器优化方法特别考虑了GPU容器的冗余组件体积大的特点,实现了有针对性的精简. 以Lou的方法为代表的分层内容感知算法,无法避免冗余的GPU组件传输,从而无法降低整体的系统传输开销. 我们的裁剪方案在是根据gEdge远程卸载的特点,定向裁剪冗余的系统组件,降低系统的资源消耗,提高启动时延. 另外,我们的方法可以与分层感知调度算法结合使用,可能进一步降低容器的启动延迟.

4 实 验

在本节中,我们将通过多组实验来验证本文提出的基于容器技术的云边协同异构计算框架gEdge的性能. 为确保实验的准确性和真实性,我们在真实的公有云平台和商用的人工智能开发平台上搭建了一个异构的云边系统.

实验环境使用腾讯云的GPU服务器,搭载Tesla P4显卡. 对于边缘节点,我们选用了NVIDIA公司的NVIDIA Jetson Xavier NX人工智能开发系统. 它与传统云服务器的x86架构不同,搭载了一个六核心的ARMv8.2 CPU和一个NVIDIA Volta GPU. 这是少数几个能在市场上找到的能运行CUDA软件的边缘系统. 云端配备的TeslaP4 GPU有独享的8GBGDDR5显存,而位于边缘节点的Jetson Xavier NX与系统内存共享8 GB的LPDDR4显存. 此外,在GPU核心数上,边缘节点只有384个核心数,而云端服务器则有2560个,这将对计算能力与并行能力产生直接影响. 这也体现了一个典型的云端与边缘端的计算资源差异. 云端和边缘端的操作系统、Docker版本和CUDA版本等配置详见表2.

作为渲染型任务的隔离性评估基准,我们选择了AWS的Elastic GPU①. ElasticGPU是AWS公司提出的一个工业界最高水平的GPU加速产品,它采用了与gEdge类似的虚拟化方案,但是仅支持渲染任务.

表2 实验环境配置

	云端	边缘端
CPU	Intel(R) Xeon(R) Silver 4110 CPU @ 2.10GHz x4	6-core NVIDIA Carmel ARM®v8.2 64-bit CPU
GPU	NVIDIA Tesla P4 (89.12 TFLOPS FP16)	384-core NVIDIA Volta GPU with 48 Tensor Cores (6 TFLOPS FP16)
RAM	18 GB	8 GB 128-bit LPDDR4x
OS	Ubuntu 20.04 LTS	Ubuntu 18.04.5 LTS
Docker	Docker 20.10.17 CE	Docker 19.03.6 CE
CUDA	11.0.228	10.2.89

4.1 gEdge性能测试

我们使用Rodinia基准测试套件[34-36]对gEdge框架的性能进行评估,其中我们分别测试了gEdge框架在云端本地运行的性能情况以及将gEdge部署在云边协同的边缘环境下的性能情况.

4.1.1 gEdge云端运行情况

图6展示了gEdge框架在云端服务器上运行,其中gEdge容器和gEdge后端均部署在云端,并使用一个vGPU实例. 实验结果都以原生运行时间为基准进行了归一化处理. 相对运行时间是gEdge虚拟化后与原生运行时间的比值. 该比值用于衡量引入gEdge虚拟化方案所带来的开销,比值越低表明性能损失越小.

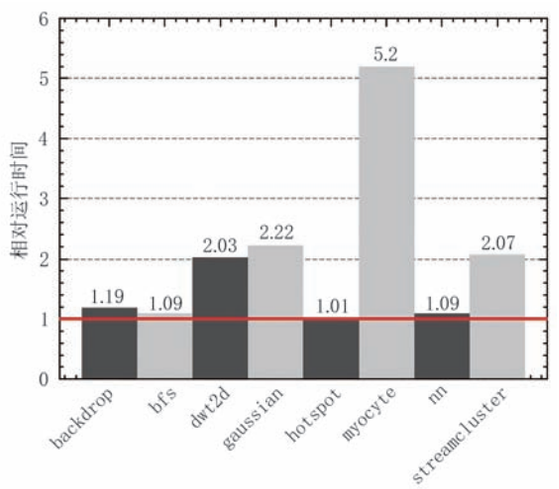

图6 本地环境下gEdge性能测试

在此实验设置下,我们得以评估gEdge加速框架所引入的GPU虚拟化开销. gEdge的虚拟化带来了平均70%的开销. 与当前最先进的基于虚拟机的GPU虚拟化方案AvA[37]对比,AvA利用了虚拟机注入拦截与远程执行加速等技术,同样实现了CUDA指令的转发与加速. 在相同性能测试程序中AvA的开销为100.0%的情况下,gEdge的开销降低了15.0%. 由表3可知,在大部分测试中,gEdge均实现了比AvA更低的开销. 然而,在bfs和nn两项测试中,与AvA相比,gEdge的开销分别增加了3.8%和7.9%. 这主要源于bfs和nn任务的显存密

① Amazon Elastic Graphics, https://aws.amazon.com/ec2/elastic-graphics/2022, 7, 1

集特性. gEdge针对显存的隔离引入了额外开销，从而导致在这两项测试中gEdge性能表现不如AvA.

表3　gEdge与AvA方案相对运行时间对比

	gEdge相对运行时间	AvA相对运行时间	开销下降
backdrop	1.19x	1.43x	16.8%
bfs	1.09x	1.05x	−3.8%
dwt2d	2.03x	2.42x	16.1%
gaussian	2.22x	2.44x	9.0%
hotspot	1.01x	1.20x	15.8%
myocyte	5.20x	7.08x	26.6%
nn	1.09x	1.01x	−7.9%
streamcluster	2.07x	3.30x	37.2%
几何平均(geomean)	1.70x	2.00x	15.0%

对于myocyte这个调用密集型程序(API-intensive task)，传统基于API转发的方法，会增加调用次数，从而增大网络开销，因此性能表现不佳. gEdge通过本地缓存与批处理等方式进行了优化，相比于AvA方案gEdge降低了26.6%的性能开销(5.20x vs. 7.08x).

4.1.2　gEdge云边协同运行情况

图7是gEdge框架在云边协同环境下的运行速度情况，即gEdge容器运行时运行在边缘节点，gEdge后端运行在云端，这是一个典型的云边协同场景. 所有的运行速度都以边缘节点原生运行速度为基准做了归一化处理.

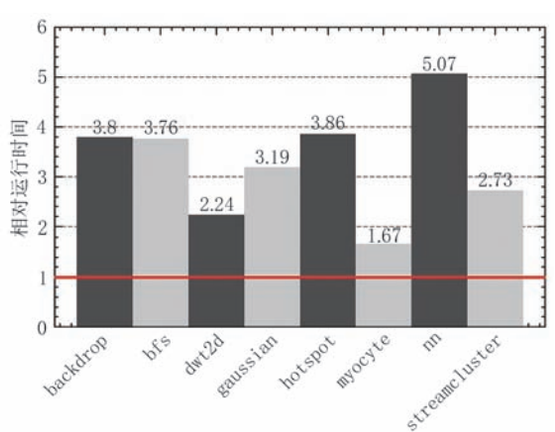

图7　云边协同下gEdge性能测试

相对运行速度是指在边缘节点上，使用gEdge进行任务卸载后的运行时间与原生运行时间之间的比值的倒数. 其中，卸载造成的网络传输时间也包括在运行时间内. 该比值用于衡量引入gEdge卸载方案相较于原生运行所实现的速度提升. 该比值越高，表明性能提升越显著.

从实验结果中可以观察到，gEdge在所有测试中都提高了性能. 如图7所示，对于调用密集型的myocyte任务，性能提升达到了67%，平均相对运行速度提升为213%. 由于myocyte涉及频繁的API调用，导致大量网络转发，从而产生较大的网络开销，因此性能提升相对较小. 然而，其性能仍高于本地运行. 另外，在nn测试程序中，性能提升达到了407%，表明gEdge在边缘环境下常见的AI类任务中具有显著的性能提升潜力. 这主要得益于nn测试中主要的数据传输仅发生在测试初期，使网络开销较小，性能提升显著.

该实验结果表明，即使远程卸载，gEdge仍显著提升性能. 这源于边缘设备的计算性能(6 TFLOPS)仅约为云端GPU设备(89.12 TFLOPS)的十五分之一. 因此，在考虑到卸载至云端所带来的网络开销后，其所获得的性能收益仍然十分显著.

4.1.3　gEdge网络开销分析

图8展示了gEdge在云边协同环境的网络开销. 计算gEdge容器的数据传输时间与任务总时间的比例，帮助我们衡量网络开销. 较低的网络传输时间占比意味着较小的网络开销.

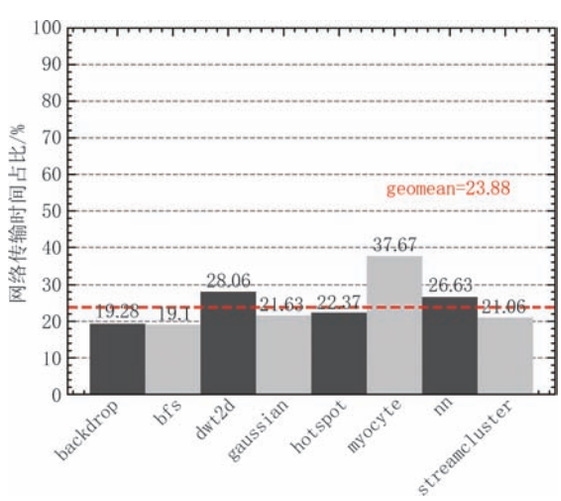

图8　云边协同下gEdge网络开销分析

实验结果显示，在Rodinia基准测试套件中，gEdge的平均网络开销为23.88%. 在Rodinia测试中，由于myocyte频繁地进行GPU调用，从而导致大量的网络请求，使得网络开销达到了37.6%. 然而，根据第4.1.2节的分析，由于gEdge采用批处理和本地缓存策略，myocyte的执行时间仍然优于仅在边缘端执行. 另一方面，尽管nn测试项目的网络

开销占比为26.63%，但由于网络传输主要发生在测试初始化阶段，并且调用较少，因此在第4.1.2节中实现了显著的加速效果.

通过对比本实验与第4.1.2节的实验结果，我们可以发现，由于GPU任务具有一定的复杂性和较长的执行时间，将任务卸载至云端所产生的网络开销占比并不显著，但加速效果依然明显.

4.1.4 gEdge实际应用的性能分析

图9展示了在云端、边缘端以及采用AvA与gEdge框架下对VGG16-SSD神经网络的训练时间对比. 本研究通过比较VGG16-SSD网络在不同任务配置下每个epoch的完成时间，以评估各配置对性能的影响.

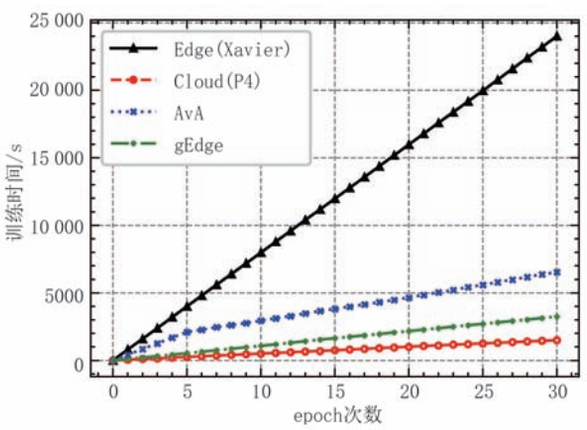

图9 VGG16-SSD网络训练时间

实验结果表明，gEdge框架相比于仅使用边缘端算力的方案具有显著的性能优势，其平均每个epoch的训练时间降低了86.5%. 此外，与AvA方案相比，gEdge在更加复杂的实际应用场景中实现了50.3%的训练时间减少. 特别是在前5个epoch中，由于gEdge的容器启动优化，其训练性能提升达到了74.9%. 这一实验结果证实了gEdge框架在实际应用场景中能够有效提升性能. 与先前框架相比，gEdge受网络延迟影响较小.

4.2 资源隔离性

为验证gEdge的资源隔离性，确保不同客户间的资源使用不互相干扰. 我们设计了两个实验来验证gEdge针对不同类型任务所采用的应用敏感的资源隔离策略的有效性.

4.2.1 通用计算型任务

针对通用计算任务，我们在2、4、6、8个gEdge容器共用单块GPU时，测量了它们的GPU利用率和显存用量. 通过比较每个容器的GPU利用率与显存用量的差值，来衡量gEdge的隔离性与公平性. 其中，每个gEdge都运行一个使用CUDARuntimeAPI实现的ResNet50[38]的应用程序，未经隔离的CUDA任务之间会竞争显存与算力，导致任务之间的算力与显存使用不平衡. ResNet50在多个领域和任务中表现出了很强的通用性能，包括图像分类、物体检测、实例分割、人脸识别等. 这使得ResNet50成为了一个广泛应用的神经网络模型，在边缘应用任务中具有典型性. 每个gEdge容器的vGPU资源配置参数如表4所示. 测量结果如图10-11所示. 从实验可以看出，gEdge容器在不同的vGPU配置下，都能保持良好的资源隔离性. 其中如图10所示，由于训练过程中，需要将训练集从边缘端的系统内存拷贝至云端的设备显存，所以每个容器使用的显存资源在最初的80秒内线性增长，增长速度取决于云端与边缘节点的带宽速度. 图11展示了通过NVIDIA-SMI采集的随机抽取的2个gEdge容器的GPU利用率，在不同数量的vGPU配置下，vGPU1与vGPU2的都是随机选取的. 当vGPU数量为2的情况下，每个容器的平均GPU使用率为38.5%和38.7%，云端物理GPU的最高利用率为75.2%. 在vGPU为4、6、8的情况下，每个容器的GPU资源利用率差距都在1%左右. 这个实验证明了gEdge框架对于GPU资源隔离的有效性.

表4 vGPU资源配置参数

vGPU数量	计算资源fGPU	显存资源vMemory
2	0.50	16(4 GB)
4	0.25	8(2 GB)
6	0.15	4(1 GB)
8	0.10	3(768 MB)

4.2.2 渲染型任务

渲染型任务代表着边缘环境下加速用户使用的VR、AR、云游戏等应用程序的性能，是边缘环境下具有代表性的任务场景之一. 我们选择了四个典型的OpenGL渲染程序进行验证. 其中，我们使用了渲染型任务的帧率作为我们的性能指标. 但是由于ElasticGPU的闭源性，我们无法获取到其真实运行的硬件信息，也无法统一两个平台的软件环境. 因为直接比较两个平台的绝对帧率无法验证隔离性，所以我们选择相对帧率作为参考指标. 我们选择使用8个典型的OpenGL程序作为测试程序.

(1)隔离性测试：我们设计了以下实验，我们对单个容器运行8个渲染程序的渲染帧率进行测量，

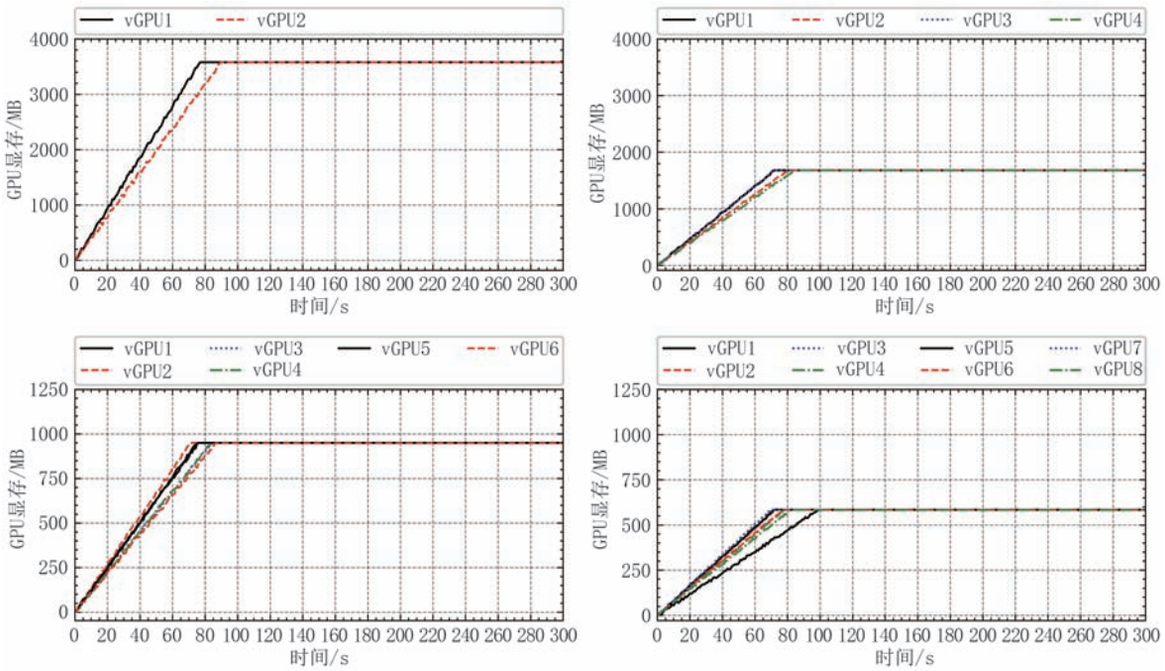

图10　不同vGPU配置下GPU显存使用情况

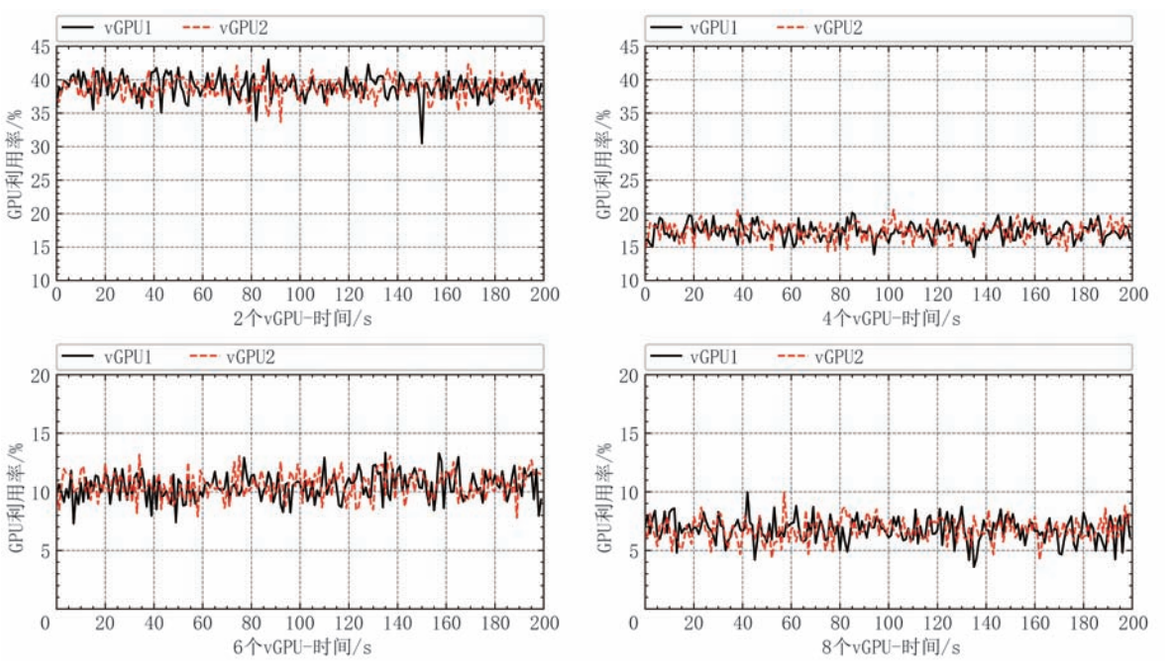

图11　不同vGPU配置下随机抽取2个GPU利用率情况

并以此为基准. 同时，再对运行10个容器的情况下，每个容器的渲染帧率进行测量，并计算与单个容器的相对性能比. 从而比较再划分出10个vGPU的情况下，多个容器运行的应用程序的隔离性情况. 如性能下降得越快，则说明多个容器同时运行时相互之间的性能影响越大，隔离性越差.

实验结果如图12所示，我们可以明显看出ElasticGPU的性能随着容器实例数量的增加而下降明显. 在同时运行10个容器的时候，ElasticGPU的平均相对帧率只有23.46%. 作为对比，gEdge在所有的情况下，平均相对帧率为93.36%. 从方差也可以看出，gEdge的多路容器性能波动较小，而多路Elastic GPU之间性能相差较大.

(2)回传带宽：由于渲染任务的特殊性，产生的结果要以画面的形式从云端传回边缘节点的容器端，我们对单个容器下ElasticGPU与gEdge使用

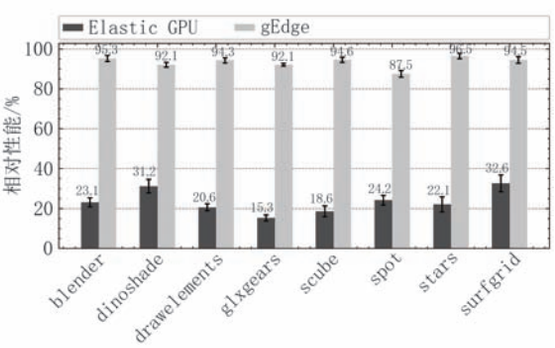

图12 多路容器的相对帧率

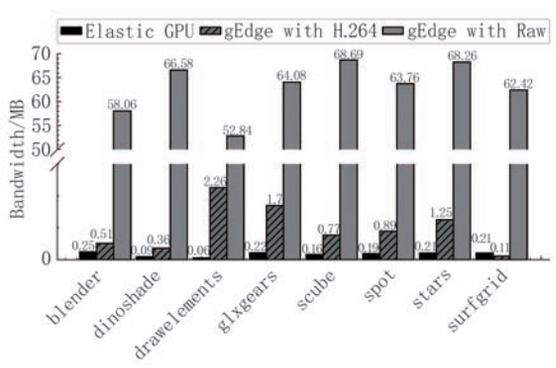

图13 带宽使用情况

的回传带宽进行了测量. 结果如图13所示,如果对回传的图像不做任何处理,平均使用的带宽为63.1 MB/s,占了总可用带宽(1 GBbps)的49.2%,这对于边缘系统而言是无法接受的. 可以看出使用了H.264编码后,gEdge的带宽都在2.5 MB/s以下,平均使用带宽下降高达98.5%,减少了大量的带宽资源. 但是与ElasticGPU使用的带宽资源相比,仍然存在差距,原因可能是ElasticGPU使用了更为先进的编码方案(如H.265等),说明gEdge的回传压缩方案仍然存在优化的空间.

(3)密度测试:同时,我们也对gEdge容器的密度进行了测试,实验结果如图14所示. 在容器数量小于32的情况下,随着容器数量的增长,云端服务器的资源使用率呈线性增长的趋势,容器的性能保持平稳,平均帧率保持在30FPS以上. 当容器数量超过32时,容器的CPU利用率达到了100%左右,CPU成为了系统的性能瓶颈:此时CPU需要同时对32路的图像流进行H.264编码,并且每个视频流的输入帧率都高于30FPS;由于CPU资源受限,此时容器内的工作负载也因为分配的CPU时间片下降,而导致容器的平均帧率出现下降趋势. 当容器数量超过40时,其性能降到不足30FPS. 此实验证明:在资源充足(即CPU资源不受限)的情况下,gEdge框架有着良好的可拓展性,资源使用率与容器密度成线性增长的趋势. 故当系统的CPU与GPU资源性能配比更加均衡时,gEdge能够达到更高的容器密度.

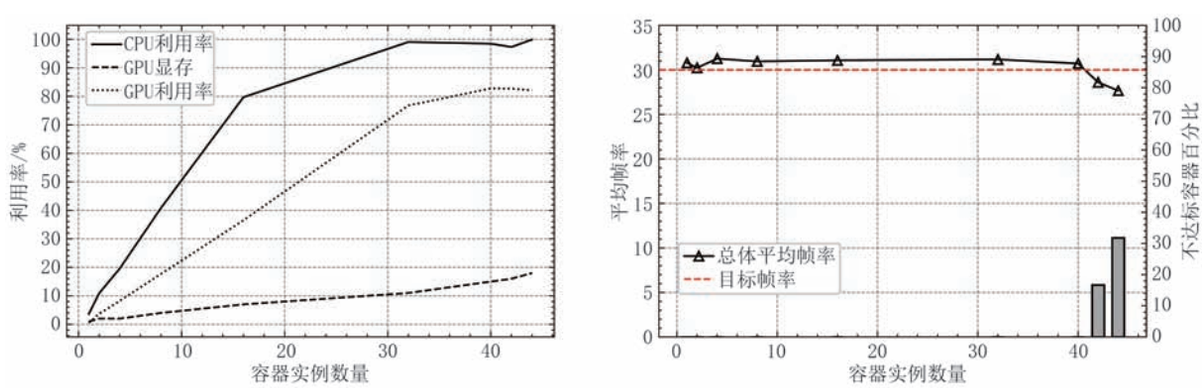

图14 容器密度实验

(4)资源分配实验:在本实验中,我们选用先来先服务(First Come, First Served,FCFS)的调度策略作为实验的基准对照,它也是GPU硬件默认的资源分配方式. 传统的FCFS策略根据任务到达的顺序为任务分配资源,而后到达的任务常常因先前的任务而受限,从而得到较少的资源. 我们运行了前述的8个基准测试项目,在每个节点上部署40个容器,并测量其平均帧率(FPS),以评估容器运行的性能状况.

如图15所示,通过比较中位数,我们发现与FCFS方法相比,基于DQN的动态资源调度算法在容器上实现了更优秀的FPS性能,且FPS大大超过了QoS要求的30帧. 此外,在帧率分布方面,基于DQN的节点的FPS范围更为集中,且主要分布在30FPS以上. 然而,采用FCFS方法的容器节点由于缺乏对资源的智能分配和调整,导致其FPS的分布范围广泛,且很多容器的FPS并未达到30帧.

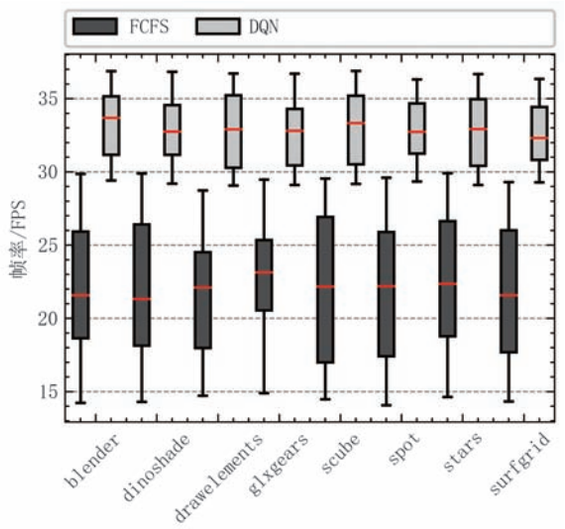

图15　资源分配实验

DQN算法通过学习环境和反馈来智能地分配资源，确保每个容器都获得其所需的资源以满足性能要求．这种方法不仅提高了整体性能，还减少了资源的浪费和竞争．根据实验结果，我们可以得出结论：针对渲染型任务，基于DQN的动态资源调度算法能够实现出色的资源分配效率和性能稳定性，显著优于传统的FCFS策略．

4.3　容器镜像优化

由于gEdge后端会将用户程序中对于GPU的使用转发至云端的服务器中，容器本身不需要对物理GPU直接进行操作．故容器镜像内包含的GPU用户态驱动如OpenGL、CUDA等文件在gEdge的架构下不会被利用到，因此可以将这部分文件从镜像中裁剪，使得容器的体积得到大幅下降．由于gEdge容器运行时与gEdge后端对于应用程序的GPU访问能够无感知地截获并转发，故该优化在gEdge架构下适用性涵盖了所有利用OpenGL与CUDA的GPU容器镜像．针对3.3节中提到的边缘友好的容器裁剪方案，我们选取四个典型的边缘容器镜像使用了容器裁剪优化，分别是L4T CUDA、L4T Tensorflow、L4T Pytorch、L4T ML.镜像的具体参数如表5所示．从表中可以看到，gEdge优化后的容器镜像体积平均降低48.8%，其中最为常用的CUDA基础镜像下降了59.9%．

gEdge的镜像部署方案沿用了原生的容器镜像部署方案，当用户的容器启动时，才会从部署在云端的容器仓库中拉取对应的容器镜像．我们设计了一个容器启动速度实验，来说明优化后的容器镜像对比原生的容器镜像的优化效果．我们选取了L4T ML镜像作为测试镜像．L4T ML镜像是由NVIDIA官方维护，专为边缘场景和NVIDIA边缘设备设计的机器学习加速镜像，它包含了开发者常用的机器学习框架软件，包括TensorFlow、PyTorch、numpy、pandas、scipy、scikit-learn、JupyterLab等，是一个具有普适代表性的镜像．

表5　容器镜像优化前后大小

镜像名称	镜像标签	镜像大小（MB）	优化后大小（MB）
L4T CUDA	10.2.460-runtime	874.5	351.2
L4T Tensorflow	r32.4.3-tf2.2-py3	891.7	513.4
L4T Pytorch	r32.4.3-pth1.6-py3	702.02	290.1
L4T ML	r32.4.3-py3	1423.36	943.1

通过在边缘节点上运行1000个容器，测量其花费的启动时间，比较gEdge优化前后的容器启动性能，实验结果如图16所示．得益于gEdge对于容器镜像大小的优化，优化后的镜像启动速度比优化前的容器快35.5%，其中优化前的第99分位时间是1.007秒，优化后的第99分位时间为0.743秒．以上实验可以得出gEdge的边缘友好容器优化效果非常显著．

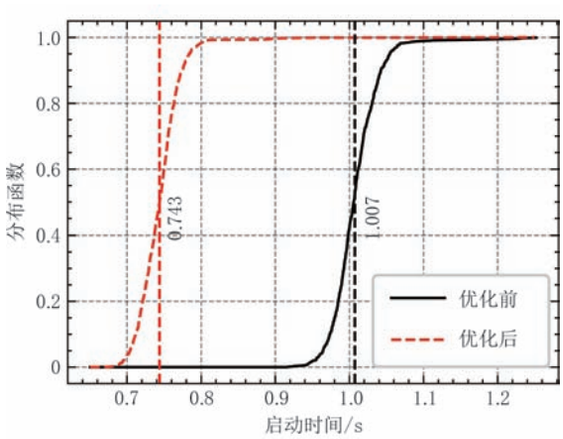

图16　优化前后容器启动时间

5　总结与展望

针对云边环境下存在的算力不足的问题，本文提出一个名为gEdge的基于容器技术的云边协同异构计算框架．gEdge主要解决了以下三个问题，首先，是如何将位于云端的服务器GPU提供给处于边缘系统的节点使用；其次，是如何在多租户的情况下，为远程边缘节点提供资源隔离；最后，使用云端

的GPU资源能否提高边缘节点的执行速度.gEdge通过API重定向技术,为边缘节点提供了vGPU设备进行计算加速,整个过程对于运行在边缘节点上的容器透明,用户不需要进行代码的改动.针对于不同类型的工作负载,gEdge采用应用敏感的资源限制策略,实验表明,gEdge使用的资源限制策略为多路容器运行提供了良好的资源隔离性.最后,在网络带宽受限的情况下,使用gEdge运行的容器性能比原生的容器相对运行速度提高了213%.

在后续的研究过程中,首先,由于边缘节点天然的移动性以及灵活性,用户可能在多个边缘节点内切换,gEdge应该增加一个能够在不同云端服务器迁移的能力.考虑到GPU应用程序的特点,我们主要会采用两种方式来提高gEdge的灵活性与容错性:第一,针对gEdge通过增加Migrator组件,对gEdge容器后端使用的资源进行资源拷贝与迁移,实现在云端服务器实时迁移的能力;第二,在gEdge容器运行时与gEdge后端之间增加缓存与错误重试机制,通过在gEdge内部实现缓存与错误重试机制,可以有效降低节点切换造成的抖动影响,从而满足边缘节点对于移动性与灵活性的要求.其次,目前gEdge还没有将边缘节点的设备信息,如功耗,响应延迟、网络带宽等参数放入框架的考量范围,未来应当提出一个设备感知的容器调度算法,通过当前节点的任务负载,来灵活将任务调度到云端或者边缘端,实现灵活的云边协同.最后,目前由于GPU设备特性限制,GPU算力资源是基于硬性限制,未来,我们将利用新型可抢占GPU硬件来实现弹性资源分配的能力,进一步提高GPU设备的利用率.

参 考 文 献

[1] Shi W, Cao J, Zhang Q, Li Y, Xu L. Edge computing: Vision and challenges. IEEE Internet of Things Journal, 2016, 3(5): 637-646

[2] Mao Y, You C, Zhang J, Huang K, Letaief K. B. A survey on mobile edge computing: The communication perspective. IEEE Communications Surveys & Tutorials, 2017, 19(4): 2322-2358

[3] Yang Zheng, He Xiao-Wu, Wu Jia-Hang, et al. Edge computing technologies for streaming video analytics. Scientia Sinica Informationis, 2022, 52: 1-53 (in Chinese)
(杨铮, 贺骁武, 吴家行等. 面向实时视频流分析的边缘计算技术. 中国科学: 信息科学, 2022, 52: 1-53)

[4] Shi Xiao-Nan, Xiong Chun-Shan, Ni Hui, et al. Analysis on 5G XR and enhancement on media service. Telecommunications Science, 2022, 38(3): 57-64 (in Chinese)
(史晓楠, 熊春山, 倪慧等. 5G XR及多媒体增强技术分析. 电信科学, 2022, 38(3): 57-64)

[5] Dastjerdi A V, Buyya R. Fog computing: Helping the internet of things realize its potential. Computer, 2016, 49(8): 112-116

[6] Zhou Z, Chen X, Li E, et al. Edge intelligence: Paving the last mile of artificial intelligence with edge computing. Proceedings of the IEEE, 2019, 107(8): 1738-1762

[7] Morabito R, Kjällman J, Komu M. Hypervisors vs. lightweight virtualization: A performance comparison//Proceedings of the 2015 IEEE International Conference on Cloud Engineering. Tempe, USA, 2015: 386-393

[8] Zhang De-Gan, Yan Hao-Ran, Zhang Jie, et al. ApproxECIoT: New edge computing architecture based on adaptive stratified sampling. Journal of Software, 2022, 33(9): 3437-3452 (in Chinese)
(张德干, 颜浩然, 张捷等. ApproxECIoT:基于自适应分层采样的边缘计算新架构. 软件学报, 2022, 33(9): 3437-3452)

[9] Yousefpour A, Patil A, Ishigaki G, et al. Fogplan: A lightweight qos-aware dynamic fog service provisioning framework. IEEE Internet of Things Journal, 2019, 6(3): 5080-5096

[10] Merlino G, Dautov R, Distefano S, Bruneo D. Enabling workload engineering in edge, fog, and cloud computing through OpenStack-based middleware. ACM Transactions on Internet Technology., 2019, 19(2): 1-22

[11] An J, Li W, Gall F L, et al. EiF: Toward an elastic IoT fog framework for AI services. IEEE Communications Magazine, 2019, 57(5): 28-33.

[12] Yigitoglu E, Mohamed M, Liu L, et al. Foggy: A framework for continuous automated IoT application deployment in fog computing//Proceedings of the 2017 IEEE International Conference on AI & Mobile Services (AIMS). Honolulu, HI, USA, 2017: 38-45

[13] Bellavista P, Zanni A. Feasibility of fog computing deployment based on docker containerization over RaspberryPi//Proceedings of the 18th International Conference on Distributed Computing and Networking, Hyderabad, India, 2017: 1-10

[14] Noor S, Koehler B, Steenson A, et al. IoTDoc: A docker-container based architecture of IoT-enabled cloud system//Big Data, Cloud Computing, and Data Science Engineering. Springer International Publishing. 2020: 51-68

[15] Xing T, Barbalace A, Olivier P, et al. H-Container: Enabling heterogeneous-ISA container migration in edge computing. ACM Transactions on Computer Systems, 2022, 39(1-4): 1-36

[16] Barbalace A, Karaoui M. L, Wang W, Xing T, Olivier P, Ravindran B. Edge computing: The case for heterogeneous-ISA container migration//Proceedings of the 16th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE' 20. Lausanne, Switzerland, 2020: 73-87

[17] Rausch T, Rashed A, Dustdar S. Optimized container scheduling for data-intensive serverless edge computing. Future Generation Computer Systems, 2021, 114: 259-271

[18] Yang Shu, Chen Zi-Teng, Cui Lai-Zhong, et al. Function delivery network: Container-based smart edge computing platform. Journal of Software, 2021, 32(12): 3945-3959 (in Chinese)
(杨术, 陈子腾, 崔来中等. 功能分发网络: 基于容器的智能边缘计算平台. 软件学报, 2021, 32(12): 3945-3959)

[19] Lou J, Luo H, Tang Z, Jia W, and Zhao W. Efficient container assignment and layer sequencing in edge computing//Proceedings of the IEEE Transactions on Services Computing, 2023, 16(2): 1118-1131

[20] Kennedy J, Varghese B, Reaño C. AVEC: Accelerator virtualization in cloud-edge computing for deep learning libraries//Proceedings of 2021 IEEE 5th International Conference on Fog and Edge Computing, Melbourne, Australia, 2021: 37-44

[21] Kang Y, Hauswald J, Gao C, et al. Neurosurgeon: Collaborative intelligence between the cloud and mobile edge//Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems, Xi'an, China, 2017: 615 - 629

[22] Guo Kai-Cheng, Wu Cheng-Gang, Zhang Wei-Feng, et al. A heterogeneous framework to accelerate CNNs with fine-grained FPGA management. Chinese Journal of Computers, 2021, 44(12): 2529-2541 (in Chinese)
(郭开诚, 吴承刚, 张伟丰等. 面向细粒度FPGA管理的CNN异构加速框架. 计算机学报, 2021, 44(12): 2529-2541)

[23] Abramson D, Jackson J, Muthrasanallur S, et al. Intel virtualization technology for directed I/O. Intel technology journal, 2006, 10(3): 1-16

[24] Doorn L V. Hardware virtualization trends//Proceedings of the 2nd International Conference on Virtual Execution Environments, Ottawa, Canada, 2006: 45

[25] Herrera A. NVIDIA GRID: Graphics accelerated VDI with the visual performance of a workstation. Nvidia Corp, 2014: 1-18

[26] Choquette J, Gandhi W, Giroux O, Stam N, Krashinsky R. NVIDIA A100 tensor core GPU: Performance and innovation//Proceedings of the IEEE Micro, 2021, 41(2): 29-35

[27] Tian K, Dong Y, Cowperthwaite D. A full GPU virtualization solution with mediated pass-through//Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference, Philadelphia, PA, 2014: 121-132

[28] Duato J, Pena A J, Silla F, et al. rCUDA: Reducing the number of GPU-based accelerators in high performance clusters//Proceedings of the 2010 International Conference on High Performance Computing & Simulation, Caen, France, 2010: 224-231

[29] Tang D, Li L, Ma J, et al. gRemote: Cloud rendering on GPU resource pool based on API-forwarding. Journal of Systems Architecture, 2021, 116: 1-13

[30] Montella R, Giunta G, Laccetti G, et al. On the virtualization of CUDA based GPU remoting on ARM and X86 machines in the GVirtuS framework. International Journal of Parallel Programming, 2017, 45(5): 1142-1163

[31] Du D, Yu T, Xia Y, et al. Catalyzer: Sub-millisecond startup for serverless computing with initialization-less booting//Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems, New York, USA, 2020: 467-481

[32] Yu T, Liu Q, Du D, et al. Characterizing serverless platforms with serverless bench//Proceedings of the 11th ACM Symposium on Cloud Computing, Virtual Event, USA: 30-44

[33] Braun L, Nikas S, Song C, Heuveline V, Fröning H. A simple model for portable and fast prediction of execution time and power consumption of GPU kernels. ACM Transactions on Architecture and Code Optimization, 2021, 18(1): 1-25

[34] Akkus I E, Chen R, Rimac I, et al. SAND: Towards high-performance serverless computing//Proceedings of the 2018 USENIX Conference on Usenix Annual Technical Conference, Boston, USA, 2018: 923 - 935

[35] Memeti S, Li L, Pllana S, et al. Benchmarking OpenCL, OpenACC, OpenMP, and CUDA: Programming productivity, performance, and energy consumption//Proceedings of the 2017 Workshop on Adaptive Resource Management and Scheduling for Cloud Computing, Washington, USA, 2017: 1-6

[36] Che S, Boyer M, Meng J, et al. Rodinia: A benchmark suite for heterogeneous computing//Proceedings of the 2009 IEEE International Symposium on Workload Characterization (IISWC), 2009: 44-54

[37] Yu H, Peters AM, Akshintala A, Rossbach CJ. AvA: Accelerated virtualization of accelerators//Proceedings of the 25th International Conference on Architectural Support for Programming Languages and Operating Systems, 2020: 807-825

[38] He K, Zhang X, Ren S, et al. Deep residual learning for image recognition//Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 2016: 770-778

WANG Yun, Ph. D. candidate. His research interests include cloud computing and GPU virtualization.

TANG Dong-Jie, Ph.D., lecturer. Her research interests include cloud rendering and GPU virtualization.

GUO Kai-Cheng, Ph. D. candidate. His research interests include machine learning system and FPGA virtualization.

QI Zheng-Wei, Ph.D., professor. His research interests include system software, virtualization, and program analysis.

GUAN Hai-Bing, Ph. D., professor. His research interests include cloud computing and virtualization.

Background

Due to the exponential growth of smart devices, a considerable amount of data is generated by running different applications such as Augmented Reality (AR), Optical Character Recognition (OCR), and autonomous driving. As a result, the data generated by these devices need to be processed properly and timely. While traditional Cloud computing has been proven for its wide availability and high scalability, it requires significant amounts of data transmitted and processed in the data center. As an extension of Cloud computing, Edge Computing complements the current centralized Cloud computing with critical quantities of distributed nodes that provide computing and storage resources close to the data source and end-users. Thus, end-users can access edge nodes with much higher bandwidth and extremely low latency. However, owing to the form factor of the edge nodes, they often have fewer computing power and storage resources compared to their cloud counterparts. With the advance in Artificial Intelligence and cloud gaming, the computing requirement for related applications has been raised accordingly. Cloud and edge collaborative computing has been proposed to address the above problem. Graphics Processing Units (GPU), widely available in the Cloud data center, are suitable for accelerating related workloads. Accelerating workloads in the edge nodes by leveraging Cloud GPU devices has become a critical challenge. Existing work requires a rewrite of the current application to achieve GPU offloading in the Edge nodes.

In this paper, we propose gEdge, a container-based Cloud-Edge collaboration framework for Heterogeneous Computing which enables the edge nodes to facilitate the GPU devices equipped in the distant Cloud server for accelerating data processing. The main research contributions of this paper are as follows: (1) the first prototype of cloud-edge collaboration framework for Heterogeneous Computing, which provides the use of virtual GPU to the Edge nodes. (2) two isolation mechanisms for different workloads to achieve better isolation in the multi-tenet environment. (3) demonstrate the benefits of using gEdge by transparently executing two workload types in edge nodes. The gEdge framework proposed in this paper provides a reference for general hardware accelerator virtualization in cloud-edge collaboration.

This work is partially sponsored by the National Natural Science Foundation of China (61732010, 62141218).