(19)国家知识产权局

(12)发明专利

(10)授权公告号 CN 110888743 B
(45)授权公告日 2022.12.20

(21)申请号 201911188798.6
(22)申请日 2019.11.27
(65)同一申请的已公布的文献号
申请公布号 CN 110888743 A
(43)申请公布日 2020.03.17
(73)专利权人 中科曙光国际信息产业有限公司
地址 266000 山东省青岛市崂山区松岭路169号软件园A区211房间
(72)发明人 于润琦 郭庆 谢莹莹 于宏亮
(74)专利代理机构 北京超凡宏宇专利代理事务所(特殊普通合伙) 11463
专利代理师 张磊
(51)Int.Cl.
G06F 9/50(2006.01)
G06F 9/48(2006.01)
G06F 9/455(2006.01)

(56)对比文件
CN 106970822 A,2017.07.21
CN 108363623 A,2018.08.03
CN 108958910 A,2018.12.07
CN 109213600 A,2019.01.15
CN 109445904 A,2019.03.08
CN 103942052 A,2014.07.23

审查员 谭昭玮

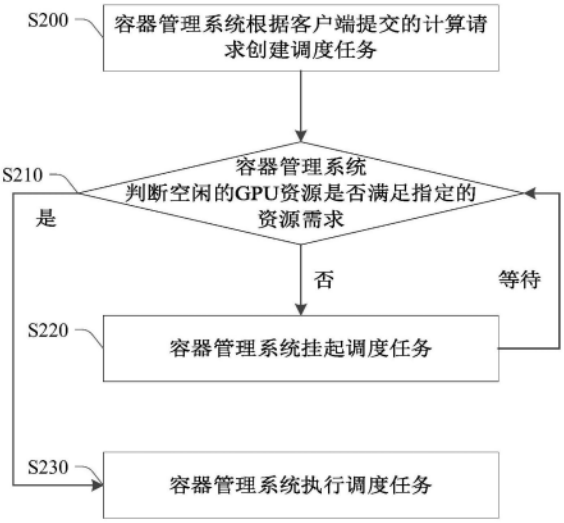
权利要求书2页 说明书10页 附图2页

(54)发明名称

一种GPU资源使用方法、装置及存储介质

(57)摘要

本申请涉及高性能计算技术领域，提供一种GPU资源使用方法、装置及存储介质。其中，GPU资源使用方法包括：容器管理系统根据客户端提交的计算请求创建调度任务；容器管理系统判断集群中空闲的GPU资源是否满足创建GPU容器的指令中指定的资源需求；若不满足需求，则容器管理系统先挂起调度任务直至满足需求时再执行；在调度任务被挂起时，容器管理系统不进行GPU容器的创建。在该方法中，容器管理系统会根据GPU资源的使用状况对调度任务进行排队处理，从而使得对于每个计算任务，都可以独占式使用GPU资源，被其使用的GPU资源不与其他计算任务共享，因此其执行进度和计算结果都可以按计划进行，不会受到其他计算任务的影响。



CN 110888743 B

1.一种GPU资源使用方法,其特征在于,包括:

部署在集群中的容器管理系统根据客户端提交的计算请求创建调度任务;其中,所述计算请求包括计算任务以及创建GPU容器的指令,所述计算任务为需要利用所述集群中的GPU资源进行计算的任务,所述GPU容器为用于运行所述计算任务的容器,所述调度任务为用于创建所述GPU容器的任务;

所述容器管理系统判断所述集群中空闲的GPU资源是否满足所述创建GPU容器的指令中指定的资源需求;其中,所述集群中空闲的GPU资源是指所述集群中未被其他计算任务占用的GPU资源;

若不满足所述资源需求,则所述容器管理系统先挂起所述调度任务直至满足需求时再执行所述调度任务;其中,在所述调度任务被挂起时,所述容器管理系统不进行GPU容器的创建;

若满足所述资源需求,则所述容器管理系统执行所述调度任务,所述调度任务在被执行时,所述容器管理系统根据所述调度任务创建所述GPU容器,将所述GPU容器调度到所述集群中包含有空闲的GPU资源的节点上运行,并在所述GPU容器内执行所述计算任务;

其中,执行所述计算任务需要的数据集以及执行所述计算任务后产生的结果集均保存在所述集群的共享存储中,所述容器管理系统在创建所述GPU容器时还将所述共享存储挂载到所述GPU容器下;

所述方法还包括:

所述容器管理系统创建CPU容器,并将所述共享存储挂载到所述CPU容器下,所述结果集中的数据在所述CPU容器中被进一步处理,所述CPU容器具有供集群用户访问的网络地址。

2.根据权利要求1所述的GPU资源使用方法,其特征在于,所述方法还包括:

在所述计算任务执行完后,所述容器管理系统销毁所述GPU容器。

3.根据权利要求1所述的GPU资源使用方法,其特征在于,所述方法还包括:

所述容器管理系统根据预配置的重启策略在所述调度任务执行失败时进行重试。

4.根据权利要求3所述的GPU资源使用方法,其特征在于,所述重启策略被配置为不断重试,所述方法还包括:

所述容器管理系统根据所述客户端发送的中断执行指令终止对所述调度任务的重试操作,若所述容器管理系统已经根据所述调度任务创建了所述GPU容器,则销毁所述GPU容器。

5.根据权利要求1-4中任一项所述的GPU资源使用方法,其特征在于,所述容器管理系统包括Kubernetes系统。

6.一种GPU资源使用装置,其特征在于,包括:

请求处理模块,用于部署在集群中的容器管理系统根据客户端提交的计算请求创建调度任务;其中,所述计算请求包括计算任务以及创建GPU容器的指令,所述计算任务为需要利用所述集群中的GPU资源进行计算的任务,所述GPU容器为用于运行所述计算任务的容器,所述调度任务为用于创建所述GPU容器的任务;

空闲资源判断模块,用于所述容器管理系统判断所述集群中空闲的GPU资源是否满足所述创建GPU容器的指令中指定的资源需求;其中,所述集群中空闲的GPU资源是指所述集

群中未被其他计算任务占用的GPU资源；

调度任务处理模块，用于若不满足所述资源需求，则所述容器管理系统先挂起所述调度任务直至满足需求时再执行所述调度任务；其中，在所述调度任务被挂起时，所述容器管理系统不进行GPU容器的创建；以及，用于若满足所述资源需求，则所述容器管理系统执行所述调度任务，所述调度任务在被执行时，所述容器管理系统根据所述调度任务创建所述GPU容器，将所述GPU容器调度到所述集群中包含有空闲的GPU资源的节点上运行，并在所述GPU容器内执行所述计算任务；

其中，执行所述计算任务需要的数据集以及执行所述计算任务后产生的结果集均保存在所述集群的共享存储中，所述容器管理系统在创建所述GPU容器时还将所述共享存储挂载到所述GPU容器下；

所述装置还包括：CPU容器创建模块，用于所述容器管理系统创建CPU容器，并将所述共享存储挂载到所述CPU容器下，所述结果集中的数据在所述CPU容器中被进一步处理，所述CPU容器具有供集群用户访问的网络地址。

7.一种计算机可读存储介质，其特征在于，所述计算机可读存储介质上存储有计算机程序指令，所述计算机程序指令被处理器读取并运行时，执行如权利要求1-5中任一项所述的方法。

一种GPU资源使用方法、装置及存储介质

技术领域

[0001] 本发明涉及图形处理器(Graphics Processing Unit,简称GPU)计算技术领域,具体而言,涉及一种GPU资源使用方法、装置及存储介质。

背景技术

[0002] 容器技术(例如,Docker)是一种新型虚拟化技术,使用容器技术,开发者可以轻松的在容器上部署和运行应用,并通过配置文件轻松实现应用的自动化安装、部署和升级,也可以很方便的将生产环境和开发环境分离,互不影响。

[0003] GPU容器(例如,Nvidia-Docker)是一种可以使用GPU设备的容器。目前在科研领域,大量的计算任务(例如,人工智能领域的深度学习任务)使用GPU容器作为运行环境,以便利用高性能的GPU设备加快运行速度。然而,由于GPU设备本身价格昂贵,出于成本考虑,在集群中通常只配置有数量十分有限的GPU设备供不同的计算任务使用。

[0004] 为共享有限的GPU资源,现有技术中通常采用对GPU设备进行虚拟化的方式,将GPU设备的时间片分配给运行有不同计算任务的GPU容器使用。然而,对于一些特定的计算任务,在运行过程中被其他任务打断(由于切换到执行其他计算任务的时间片)可能延误其执行进度和/或导致计算结果不可用,因此并不适于采用虚拟化的方式分配集群中的GPU资源。

发明内容

[0005] 本申请实施例的目的在于提供一种GPU资源使用方法、装置及存储介质,以改善上述技术问题。

[0006] 为实现上述目的,本申请提供如下技术方案:

[0007] 第一方面,本申请提供一种GPU资源使用方法,包括:部署在集群中的容器管理系统根据客户端提交的计算请求创建调度任务;其中,所述计算请求包括计算任务以及创建GPU容器的指令,所述计算任务为需要利用所述集群中的GPU资源进行计算的任务,所述GPU容器为用于运行所述计算任务的容器,所述调度任务为用于创建所述GPU容器的任务;所述容器管理系统判断所述集群中空闲的GPU资源是否满足所述创建GPU容器的指令中指定的资源需求;其中,所述集群中空闲的GPU资源是指所述集群中未被其他计算任务占用的GPU资源;若不满足所述资源需求,则所述容器管理系统先挂起所述调度任务直至满足需求时再执行所述调度任务;其中,在所述调度任务被挂起时,所述容器管理系统不进行GPU容器的创建。

[0008] 在上述方法中,容器管理系统会根据集群中GPU资源的使用状况判断是否要执行调度任务,若GPU资源不足,则先将调度任务挂起直至有足够多的空闲GPU资源时再执行,即对调度任务进行排队处理。其中,调度任务为用于创建GPU容器的任务,而GPU容器为用于运行计算任务的容器,所以上述方法实际上也可以视为根据集群中GPU资源的使用状况对计算任务进行排队处理,从而使得对于每个计算任务,都可以按照排队的顺序独占式地使用

GPU资源，被其使用的GPU资源并不与其他计算任务共享，所以在GPU资源有限时，若容器管理系统接收到多个计算任务，其中的一些可能被挂起(调度任务被挂起相应的计算任务也不会执行，因此可以视为挂起)。

[0009] 由于计算任务可以独占式地使用GPU资源，因此其执行进度和计算结果都可以按计划进行，不会受到其他计算任务的影响，有利于保障科研工作的正常进行。此外，现有技术中对GPU资源进行虚拟化的解决方案实现复杂，实施成本也较高，而本申请实施例提供的上述方法对GPU资源的使用方式简单高效，实施成本也较低。

[0010] 在第一方面的一种实现方式中，所述方法还包括：若满足所述资源需求，则所述容器管理系统执行所述调度任务，所述调度任务在被执行时，所述容器管理系统根据所述调度任务创建所述GPU容器，将所述GPU容器调度到所述集群中包含有空闲的GPU资源的节点上运行，并在所述GPU容器内执行所述计算任务。

[0011] 这里涵盖了两种情况，一种是客户端提交计算任务后集群中有足够的空闲GPU资源，此时调度任务可以立即执行，实现GPU容器的创建并在GPU容器中执行计算任务；另一种是客户端提交计算任务后集群中没有足够的空闲GPU资源，但调度任务挂起一段时间后集群中有足够的空闲GPU资源，并且也轮到该调度任务执行，则此时调度任务可以执行，实现GPU容器的创建并在GPU容器中执行计算任务。GPU容器在创建后，执行任务所需的GPU资源被GPU容器所占据，从而计算任务可以独占式地使用GPU容器中地GPU资源。

[0012] 在第一方面的一种实现方式中，所述方法还包括：在所述计算任务执行完后，所述容器管理系统销毁所述GPU容器。

[0013] 在这种实现方式中，及时销毁GPU容器可以释放被GPU容器占据的GPU资源，使得容器管理系统可以继续进行调度，执行后续的计算任务。

[0014] 在第一方面的一种实现方式中，所述方法还包括：所述容器管理系统根据预配置的重启策略在所述调度任务执行失败时进行重试。

[0015] 在这种实现方式中，通过配置重启策略使得调度任务尽可能得到执行，避免一些偶然因素导致的调度任务执行失败，有利于保障计算任务的顺利执行。

[0016] 在第一方面的一种实现方式中，所述重启策略被配置为不断重试，所述方法还包括：所述容器管理系统根据所述客户端发送的中断执行指令终止对所述调度任务的重试操作，若所述容器管理系统已经根据所述调度任务创建了所述GPU容器，则销毁所述GPU容器。

[0017] 在一些容器管理系统(如Kubernetes)中，并不能自由指定重试次数，而只能指定不断重试模式，即若调度任务一直无法成功执行，重试执行的过程会一直持续下去，此时可以由客户端根据重试结果确定是否要继续重试。例如，客户端可以在调度任务重试了数次均失败后发出中断执行的指令，避免影响其他调度任务的执行。

[0018] 在第一方面的一种实现方式中，执行所述计算任务需要的数据集以及执行所述计算任务后产生的结果集均保存在所述集群的共享存储中，所述容器管理系统在创建所述GPU容器时还将所述共享存储挂载到所述GPU容器下。

[0019] 在这种实现方式中，将数据集保存至共享存储，方便计算任务使用，而将结果集保存至共享存储，则可以避免因为GPU容器被销毁导致结果集丢失，并且也可以方便其他应用程序对结果集进行访问以及进一步处理。

[0020] 在第一方面的一种实现方式中，所述方法还包括：所述容器管理系统创建CPU容

器，并将所述共享存储挂载到所述CPU容器下。

[0021] CPU容器可以视为集群提供给用户的一个访问接口，用户可以远程访问CPU容器从而使用集群中的计算资源(主要指CPU资源，不包括GPU资源)，这里由于共享存储也挂载到CPU容器下，所以用户可以访问到计算任务保存到共享存储上的结果集，从而可以在CPU容器中对结果集中的数据进行进一步处理。

[0022] 在第一方面的一种实现方式中，所述容器管理系统包括Kubernetes系统。

[0023] Kubernetes系统中的基本操作单元为Pod，每个Pod中包含一个或多个紧密相关的容器，这些容器共享存储和网络资源。上面提到的GPU容器和CPU容器在Kubernetes集群(指部署有Kubernetes系统的集群)中都包含在Pod内。

[0024] 第二方面，本申请实施例提供一种GPU资源使用装置，包括：请求处理模块，用于部署在集群中的容器管理系统根据客户端提交的计算请求创建调度任务；其中，所述计算请求包括计算任务以及创建GPU容器的指令，所述计算任务为需要利用所述集群中的GPU资源进行计算的任务，所述GPU容器为用于运行所述计算任务的容器，所述调度任务为用于创建所述GPU容器的任务；空闲资源判断模块，用于所述容器管理系统判断所述集群中空闲的GPU资源是否满足所述创建GPU容器的指令中指定的资源需求；其中，所述集群中空闲的GPU资源是指所述集群中未被其他计算任务占用的GPU资源；调度任务处理模块，用于若不满足所述资源需求，则所述容器管理系统先挂起所述调度任务直至满足需求时再执行所述调度任务；其中，在所述调度任务被挂起时，所述容器管理系统不进行GPU容器的创建。

[0025] 第三方面，本申请实施例提供一种计算机可读存储介质，所述计算机可读存储介质上存储有计算机程序指令，所述计算机程序指令被处理器读取并运行时，执行第一方面或第一方面的任意一种可能的实现方式提供的方法。

[0026] 第四方面，本申请实施例提供一种电子设备，包括：存储器以及处理器，所述存储器中存储有计算机程序指令，所述计算机程序指令被所述处理器读取并运行时，执行第一方面或第一方面的任意一种可能的实现方式提供的方法。

附图说明

[0027] 为了更清楚地说明本申请实施例的技术方案，下面将对本申请实施例中所需要使用的附图作简单地介绍，应当理解，以下附图仅示出了本申请的某些实施例，因此不应被看作是对范围的限定，对于本领域普通技术人员来讲，在不付出创造性劳动的前提下，还可以根据这些附图获得其他相关的附图。

[0028] 图1示出了一种Kubernetes集群的架构图；

[0029] 图2示出了本申请实施例提供的一种GPU资源使用方法的流程图；

[0030] 图3示出了本申请实施例提供的一种GPU资源使用装置的功能模块图；

[0031] 图4示出了本申请实施例提供的一种电子设备的结构图。

具体实施方式

[0032] 下面将结合本申请实施例中的附图，对本申请实施例中的技术方案进行描述。应注意到：相似的标号和字母在下面的附图中表示类似项，因此，一旦某一项在一个附图中被定义，则在随后的附图中不需要对其进行进一步定义和解释。术语“包括”、“包含”或者其任

何其他变体意在涵盖非排他性的包含，从而使得包括一系列要素的过程、方法、物品或者设备不仅包括那些要素，而且还包括没有明确列出的其他要素，或者是还包括为这种过程、方法、物品或者设备所固有的要素。在没有更多限制的情况下，由语句“包括一个……”限定的要素，并不排除在包括所述要素的过程、方法、物品或者设备中还存在另外的相同要素。

[0033] 本申请主要针对集群中GPU资源较少、而需要利用GPU资源的计算任务较多的情况，提出一种排队使用GPU资源的方法，该方法使得每个计算任务可以独占式地使用GPU资源，并且多个计算任务也能够依次得到处理，实现了对GPU资源简单、高效地利用，方法实施成本较低。当然，也不是要排除在其他情况下使用该方法，即该方法是一种使用GPU资源的通用方法。在本文中，GPU资源泛指各类可用于GPU计算的GPU设备，如GPU卡等。

[0034] 该方法用于集群环境，集群可以理解为大量具有运算处理能力的电子设备的集合，这些电子可以协作完成特定的工作。例如，上述电子设备可以是服务器，并且不限于物理设备，也可以是虚拟机。

[0035] 进一步的，在该集群环境中，应用程序以容器化的方式进行部署和运行，集群中部署有容器管理系统对容器进行管理。一个典型的这样的集群是Kubernetes集群，该集群中部署的Kubernetes是一个开源的容器管理系统，该系统采用分布式架构，为容器化的应用提供部署运行、资源调度、服务发现和动态伸缩等一系列完整功能，提高了大规模容器集群管理的便捷性。图1示出了一种Kubernetes集群的架构图，应当理解，Kubernetes系统仅仅是示例，本申请的方法实际上并不限定容器管理系统具体采用何种系统，但为简化阐述，后文在提到容器管理系统时，仍然主要以Kubernetes为例进行介绍。

[0036] 参照图1，Kubernetes集群100主要包括管理节点110(Master)和服务节点120(Node)，各节点之间可以相互通信，在图1示出的例子中，管理节点110为一个，服务节点120为多个，这里的节点可以指集群中的服务器，但不限于物理设备。

[0037] 其中，管理节点110负责集群的管理和控制，管理节点110上运行Kubernetes API Server、Kubernetes Controller Manager以及Kubernetes Scheduler等关键进程，此外还可以启动Etcd服务。服务节点120为集群中的工作负载节点，管理节点110会将工作负载(例如下面要介绍的Pod)分配给服务节点120，服务节点120上运行Kubelet、Kube-Proxy、Docker Engine等关键进程。以上构成Kubernetes系统的各进程的具体功能可以参考现有技术，这里不作具体阐述。

[0038] Kubernetes集群100中的最小运行单元为Pod对象(简称Pod)，Pod被Kubernetes系统创建并调度到服务节点120上运行，例如，Kubernetes系统可以根据集群中资源的使用状况进行调度。

[0039] 每个Pod中包含一个或多个紧密相关的容器，这些容器共享存储和网络资源，容器为应用程序的运行提供环境。Kubernetes系统中容器主要指Docker容器，当然也可以是Rocket等容器，取决于不同Kubernetes系统版本的支持，但后文主要以最常见的Docker容器为例进行介绍。需要指出，在后文中，有时也会直接采用Kubernetes系统创建、调度或销毁容器这种说法，应当理解这只是一种简化的说法，其完整的含义是Kubernetes系统创建、调度或销毁Pod，由于Pod中包含容器，所以也同时完成了对容器的创建、调度或销毁。

[0040] 在本申请中，容器中要运行的应用程序主要指需要使用集群中的GPU资源的GPU计算任务(简称计算任务)，在Kubernetes集群100中，GPU设备可以配置在服务节点120上。可

以使用GPU资源的容器在本申请中称为GPU容器，而不能使用GPU资源的容器在本申请中称为CPU容器(使用CPU资源)，由于GPU设备价格昂贵，因此在Kubernetes集群中，GPU资源通常是一种比较稀缺的资源，而CPU资源通常是一种非稀缺的资源。上述计算任务在既然要使用GPU资源，因此需要在GPU容器中运行。

[0041] 为使原生的Docker容器能够支持使用GPU资源，Nvidia公司在Docker容器的基础上进行了一层封装，提出了Nvidia-Docker容器，Nvidia-Docker容器是一种GPU容器，相应地，原生的Docker容器则是一种CPU容器。Kubernetes系统提供了对Nvidia-Docker容器的创建和调度能力，但Nvidia-Docker对GPU资源的利用最小粒度为一张GPU卡(并且只独占使用一张GPU卡)，不支持更细粒度资源的分配，例如若希望某个容器占用半张GPU卡，目前的Kubernetes系统设计中无法实现，但目前的设计可以实现比较好的隔离性，确保使用GPU卡的计算任务不受其他计算任务的影响。

[0042] 在后文中，为简单起见，除非特别说明的，在提到Kubernetes集群中的节点时，一般理解为服务节点120。

[0043] 图2示出了本申请实施例提供的一种GPU资源使用方法的流程图，该方法可以应用于部署在集群中的容器管理系统，包括但不限于上面提到的Kubernetes系统。参照图2，该方法包括以下步骤：

[0044] 步骤S200：容器管理系统根据客户端提交的计算请求创建调度任务。

[0045] 客户端可以由用户使用，计算请求中至少包括两项内容，一项是计算任务，另一项是指示容器管理系统创建GPU容器的指令。其中，计算任务的概念前文已经给出，在创建GPU容器的指令中可以指定计算任务对GPU资源的需求(简称资源需求)，例如，一种常见的情况中，一个计算任务需要一张GPU卡(或一个GPU容器)来执行，这就是一种资源需求。在一些实现方式中，计算请求可以是包含上述两项内容的一个请求，在另一些实现方式中，计算请求也可以是两个请求，分别包含上述两项内容，客户端分两次向容器管理系统提交。

[0046] 作为对计算任务的响应，容器管理系统会创建调度任务，调度任务为用于创建和调度GPU容器的任务。其中，容器的创建和调度可以认为是一个一体的过程，GPU容器创建好后，即被容器管理系统调度至合适的节点上运行，例如，调度至有空闲GPU资源的节点(比如，某个节点上配置有GPU卡，且该GPU卡处于空闲状态)上运行。

[0047] 在Kubernetes集群中，客户端可以将计算请求提交到Kubernetes Job，Kubernetes Job为Kubernetes系统中的一个组件，用于管理调度任务，在Kubernetes系统中，调度任务也称为Job，Job可以由Kubernetes Job在接收到计算请求后根据请求内容创建。

[0048] 步骤S210：容器管理系统判断集群中空闲的GPU资源是否满足创建GPU容器的指令中指定的资源需求。

[0049] 若集群中空闲的GPU资源满足创建GPU容器的指令中指定的资源需求，则执行步骤S230，在步骤S230中调度任务被执行，GPU容器也会被创建；若集群中空闲的GPU资源不满足创建GPU容器的指令中指定的资源需求，则执行步骤S220，在步骤S220中调度任务被挂起，GPU容器也不会被创建。GPU容器一旦创建，则会占用集群中的GPU资源，即占用要在该容器中运行的计算任务所需要的资源(在资源需求中指定)。在本申请的方案中，GPU资源一旦被某个GPU容器占用，就不会再被其他GPU容器占用，从而在占据GPU资源的GPU容器中运行的

计算任务会独占式地使用这些GPU资源,除非GPU容器被销毁,这些被容器占用的GPU资源才会释放。

[0050] 步骤S210中空闲的GPU资源是指集群中尚未被已创建的GPU容器占用的GPU资源。例如,在一个Kubernetes集群中只配置有一张GPU卡,Kubernetes系统先后收到客户端发起的两个计算请求,其对应的资源需求都是一张GPU卡,一旦Kubernetes系统针对在先的计算请求创建了一个Nvidia-Docker容器(通过执行根据该计算请求创建的调度任务),集群中的GPU卡即已经分配给该容器独占使用,此时集群中已经没有空闲的GPU资源,Kubernetes系统不会再为在后的计算请求创建Nvidia-Docker容器(通过将根据该计算请求创建的调度任务挂起)。但若该Kubernetes集群中配置有两张GPU卡,则Kubernetes系统在处理完在先的计算请求后集群中还有空闲的GPU资源(一张GPU卡),此时Kubernetes系统可以为在后的计算请求创建Nvidia-Docker容器。

[0051] 步骤S220:容器管理系统挂起调度任务。

[0052] 调度任务被挂起即暂不执行调度任务,等待直至集群中空闲的GPU资源满足相应的资源需求再重启执行调度任务。仍然采用阐述步骤S210时提到的例子,在一个Kubernetes集群中只配置有一张GPU卡,Kubernetes系统先后收到客户端发起的两个计算请求,其对应的资源需求都是一张GPU卡,在后的计算请求对应的调度任务会被Kubernetes系统挂起,直至在先的计算请求对应的计算任务被执行完毕,相应的Nvidia-Docker容器被销毁,集群中的GPU卡也不再被容器占用,此时挂起的调度任务可以恢复执行。

[0053] 步骤S230:容器管理系统执行调度任务。

[0054] 步骤S230的执行涵盖了两种情况,一种是客户端提交计算任务后集群中有足够的空闲GPU资源,此时调度任务可以立即执行;另一种是客户端提交计算任务后集群中没有足够的空闲GPU资源,但调度任务挂起一段时间后集群中有足够的空闲GPU资源,并且也轮到该调度任务执行,则此时调度任务可以执行。

[0055] 调度任务在被执行时,容器管理系统会根据调度任务创建GPU容器,将GPU容器调度到集群中包含有空闲的GPU资源的节点上运行,并在GPU容器内执行计算任务。

[0056] 例如,在Kubernetes集群中,Kubernetes Job组件负责Job的创建及执行,Job执行后,Kubernetes系统中的控制器Kubernetes Controller根据Job的相关信息创建Pod(也即同时创建了Nvidia-Docker容器),而Kubernetes系统中的调度器Kubernetes Scheduler负责将Pod调度到有空闲GPU资源的节点上运行。为了能够在Kubernetes系统中管理和分配GPU资源,Nvidia公司提供了基于Nvidia GPU的插件Device Plugin,该插件部署在集群中的节点上,主要用于上报节点上的GPU设备数量及支持分配GPU设备的行为。节点(指Pod被调度至的节点)上的Kubelet进程启动运行Pod时,会调用DevicePlugin提供的Allocate接口,该接口能够根据Nvidia-Docker的NVIDIA_VISIBLE_DEVICES环境变量,将gpu-containers-runtime-hook转换为--devices参数,然后调用nvidia-container-cli prestart,从而将GPU设备映射到Nvidia-Docker容器中(因此在容器中可以发现并使用该GPU设备)。

[0057] 需要指出,调度任务在GPU容器创建完成后即可以结束,不必等待计算任务的执行结束。计算任务执行结束后,容器管理系统可以将GPU容器销毁,避免其继续占用GPU资源,导致其他计算任务无法使用GPU资源。

[0058] 在上述方法中，容器管理系统会根据集群中GPU资源的使用状况判断是否要执行调度任务，若GPU资源不足，则先将调度任务挂起直至有足够多的空闲GPU资源时再执行，即对调度任务进行排队处理。由于调度任务为用于创建GPU容器的任务，而GPU容器为用于运行计算任务的容器，所以上述方法实际上也可以视为根据集群中GPU资源的使用状况对计算任务进行排队处理，从而使得对于每个计算任务，都可以按照排队的顺序使用GPU资源，并且由于GPU容器对GPU资源的独占特性，每个计算任务都独占式地使用GPU资源，被其使用的GPU资源并不与其他计算任务共享。注意，如果GPU资源足够，本申请的方案并不排除同时执行多个计算任务，但每个计算任务自己占用的GPU资源在其执行期间也并不与其他计算任务共享。

[0059] 由于计算任务可以独占式地使用GPU资源，因此其执行进度和计算结果都可以按计划进行，不会受到其他计算任务的影响(例如，不会被其他计算任务打断)，有利于保障科研工作的正常进行。并且，由于采用了排队机制，容器管理系统可以对多个计算请求进行响应并依次处理这些计算请求。此外，现有技术中对GPU资源进行虚拟化的解决方案实现复杂，实施成本也较高，而上述方法对GPU资源的使用方式不涉及时间片切分，因此实现上简单高效，实施成本也较低。

[0060] 在一些实现方式中，容器管理系统可以根据预配置的重启策略在调度任务执行失败时进行重试。此举主要是为了使得调度任务尽可能得到执行，避免一些临时性的因素导致的调度任务执行失败(进而计算任务也无法执行)，尽可能保障计算任务的顺利执行。

[0061] 例如，在Kubernetes中，重启策略为restartPolicy，若restartPolicy设置为Never表示不进行重试，若restartPolicy设置为Always表示不断重试，即调度任务执行不成功就一直重试下去(还有其他取值，此处从略)。当然，若一个调度任务不断重试却不成功，难免影响后续调度任务的执行，因此在一些可选方案中，客户端可以获取调度任务的执行结果(由Kubernetes记录并输出)，并在适当的时机向Kubernetes系统发送中断执行指令，收到该指令后，Kubernetes系统即不再重试当前的调度任务。上面所称的适当的时机可以是重试达到了预设的次数但调度任务仍然未成功执行，预设次数不作限定(比如可以是1次、3次等)，当然也可以是重试了预设的时长但调度任务仍然未成功执行，预设时长不作限定(比如可以是1分钟、3分钟等)。

[0062] 调度任务执行失败，多数情况下不会创建GPU容器，但如果调度任务执行失败是在创建GPU容器之后，则重试之前需要先将之前创建的GPU容器销毁掉，在重试执行调度任务时重新进行GPU容器的创建。

[0063] 在一些实现方式中，计算任务的执行需要输入一些数据(称为数据集)，计算任务执行后也会输出一些结果(称为结果集)，例如，对于深度学习中的训练任务，数据集可以是训练样本，结果集可以是训练好的模型参数。在集群中可以设置共享存储(例如，通过设置专门的I/O节点)，容器管理系统在创建GPU容器时还将共享存储的目录(例如，可以是网络文件系统NFS的共享目录)挂载到GPU容器下，从而在GPU容器中运行的计算任务既可以方便地使用共享存储中的数据集，也可以方便地将运行后产生的结果集保存到共享存储中。由于计算任务执行完成后，GPU容器可能被容器管理系统销毁以释放GPU资源，因此将结果集保存在共享存储中(而非GPU容器的本地存储中)是对结果集的一种持久化存储方案，使得在GPU容器销毁后这些数据也不会丢失，从而其他应用程序对结果集进行访问以及进一步

处理结果集中的数据，例如，基于训练好的模型参数进行测试。

[0064] 在一些实现方式中，容器管理系统还可以为每个集群用户创建CPU容器，其数量不限定，因为CPU资源在集群中通常不是稀缺资源。CPU容器可以视为集群提供给用户的一个访问接口，每个CPU容器有自己的网络地址，从而用户可以远程访问CPU容器以便使用集群提供给他的计算资源(主要指CPU资源，不包括GPU资源)，通过提供CPU容器可以避免用户直接访问集群中的物理节点。在创建CPU容器时，可以将集群中共享存储的目录也挂载到CPU容器下，使得用户可以访问到计算任务保存到共享存储上的结果集，进而可以在CPU容器中对结果集中的数据进行进一步处理。

[0065] 创建CPU容器的时机不作限定，例如，可以在用户向集群申请资源时就进行创建(时间可以早于步骤S200)，又例如，可以在容器管理系统接收到计算请求后和GPU容器一起创建，等等。可以理解的，创建CPU容器并非必须的步骤，例如，用户不打算处理结果集中的数据，则可以不创建CPU容器，又例如，用户可以直接访问共享存储中的结果集(不通过容器的方式)，也可以不创建CPU容器。

[0066] 图3示出了本申请实施例提供的GPU资源使用装置300的功能模块图。参照图3，GPU资源使用装置300包括：

[0067] 请求处理模块310，用于部署在集群中的容器管理系统根据客户端提交的计算请求创建调度任务；其中，所述计算请求包括计算任务以及创建GPU容器的指令，所述计算任务为需要利用所述集群中的GPU资源进行计算的任务，所述GPU容器为用于运行所述计算任务的容器，所述调度任务为用于创建所述GPU容器的任务；

[0068] 空闲资源判断模块320，用于所述容器管理系统判断所述集群中空闲的GPU资源是否满足所述创建GPU容器的指令中指定的资源需求；其中，所述集群中空闲的GPU资源是指所述集群中未被其他计算任务占用的GPU资源；

[0069] 调度任务处理模块330，用于若不满足所述资源需求，则所述容器管理系统先挂起所述调度任务直至满足需求时再执行所述调度任务；其中，在所述调度任务被挂起时，所述容器管理系统不进行GPU容器的创建。

[0070] 在GPU资源使用装置300的一种实现方式中，调度任务处理模块330还用于若满足所述资源需求，则所述容器管理系统执行所述调度任务，所述调度任务在被执行时，所述容器管理系统根据所述调度任务创建所述GPU容器，将所述GPU容器调度到所述集群中包含有空闲的GPU资源的节点上运行，并在所述GPU容器内执行所述计算任务。

[0071] 在GPU资源使用装置300的一种实现方式中，装置还包括：

[0072] 销毁模块，用于在所述计算任务执行完后，所述容器管理系统销毁所述GPU容器。

[0073] 在GPU资源使用装置300的一种实现方式中，调度任务处理模块330还用于所述容器管理系统根据预配置的重启策略在所述调度任务执行失败时进行重试。

[0074] 在GPU资源使用装置300的一种实现方式中，所述重启策略被配置为不断重试，调度任务处理模块330还用于所述容器管理系统根据所述客户端发送的中断执行指令终止对所述调度任务的重试操作，若所述容器管理系统已经根据所述调度任务创建了所述GPU容器，则销毁所述GPU容器。

[0075] 在GPU资源使用装置300的一种实现方式中，执行所述计算任务需要的数据集以及执行所述计算任务后产生的结果集均保存在所述集群的共享存储中，所述容器管理系统在

创建所述GPU容器时还将所述共享存储挂载到所述GPU容器下。

[0076] 在GPU资源使用装置300的一种实现方式中,装置还包括:

[0077] CPU容器创建模块,用于所述容器管理系统创建CPU容器,并将所述共享存储挂载到所述CPU容器下。

[0078] 在GPU资源使用装置300的一种实现方式中,所述容器管理系统包括Kubernetes系统。

[0079] 本申请实施例提供的GPU资源使用装置300,其实现原理及产生的技术效果在前述方法实施例中已经介绍,为简要描述,装置实施例部分未提及之处,可参考方法施例中相应内容。

[0080] 图4示出了本申请实施例提供的电子设备400的一种可能的结构。参照图4,电子设备400包括:处理器410、存储器420以及通信接口430,这些组件通过通信总线440和/或其他形式的连接机构(未示出)互连并相互通讯。

[0081] 其中,存储器420包括一个或多个(图中仅示出一个),其可以是,但不限于,随机存取存储器(Random Access Memory,简称RAM),只读存储器(Read Only Memory,简称ROM),可编程只读存储器(Programmable Read-Only Memory,简称PROM),可擦除只读存储器(Erasable Programmable Read-Only Memory,简称EPROM),电可擦除只读存储器(Electric Erasable Programmable Read-Only Memory,简称EEPROM)等。处理器410以及其他可能的组件可对存储器420进行访问,读和/或写其中的数据。

[0082] 处理器410包括一个或多个(图中仅示出一个),其可以是一种集成电路芯片,具有信号的处理能力。上述的处理器410可以是通用处理器,包括中央处理器(Central Processing Unit,简称CPU)、微控制单元(Micro Controller Unit,简称MCU)、网络处理器(Network Processor,简称NP)或者其他常规处理器;还可以是专用处理器,包括数字信号处理器(Digital Signal Processor,简称DSP)、专用集成电路(Application Specific Integrated Circuits,简称ASIC)、现场可编程门阵列(Field Programmable Gate Array,简称FPGA)或者其他可编程逻辑器件、分立门或者晶体管逻辑器件、分立硬件组件。

[0083] 通信接口430包括一个或多个(图中仅示出一个),可以用于和其他设备进行直接或间接地通信,以便进行数据的交互。通信接口430可以是以太网接口;可以是高速网络接口(如Infiniband网络);可以是移动通信网络接口,例如3G、4G、5G网络的接口;还是可以是具有数据收发功能的其他类型的接口。

[0084] 在存储器420中可以存储一个或多个计算机程序指令,处理器410可以读取并运行这些计算机程序指令,以实现本申请实施例提供的GPU资源使用方法以及其他期望的功能。

[0085] 可以理解,图4所示的结构仅为示意,电子设备400还可以包括比图4中所示更多或者更少的组件,或者具有与图4所示不同的配置。图4中所示的各组件可以采用硬件、软件或其组合实现。于本申请实施例中,电子设备400可以是Kubernetes集群中的节点,例如图1中的管理节点110和/或服务节点120。

[0086] 本申请实施例还提供一种计算机可读存储介质,该计算机可读存储介质上存储有计算机程序指令,所述计算机程序指令被计算机的处理器读取并运行时,执行本申请实施例提供的GPU资源使用方法。例如,计算机可读存储介质可以实现为图4中电子设备400中的存储器420。

[0087] 在本申请所提供的实施例中，应该理解到，所揭露装置和方法，可以通过其它的方式实现。以上所描述的装置实施例仅仅是示意性的，例如，所述单元的划分，仅仅为一种逻辑功能划分，实际实现时可以有另外的划分方式，又例如，多个单元或组件可以结合或者可以集成到另一个系统，或一些特征可以忽略，或不执行。另一点，所显示或讨论的相互之间的耦合或直接耦合或通信连接可以是通过一些通信接口，装置或单元的间接耦合或通信连接，可以是电性，机械或其它的形式。

[0088] 另外，作为分离部件说明的单元可以是或者也可以不是物理上分开的，作为单元显示的部件可以是或者也可以不是物理单元，即可以位于一个地方，或者也可以分布到多个网络单元上。可以根据实际的需要选择其中的部分或者全部单元来实现本实施例方案的目的。

[0089] 再者，在本申请各个实施例中的各功能模块可以集成在一起形成一个独立的部分，也可以是各个模块单独存在，也可以两个或两个以上模块集成形成一个独立的部分。

[0090] 以上所述仅为本申请的实施例而已，并不用于限制本申请的保护范围，对于本领域的技术人员来说，本申请可以有各种更改和变化。凡在本申请的精神和原则之内，所作的任何修改、等同替换、改进等，均应包含在本申请的保护范围之内。

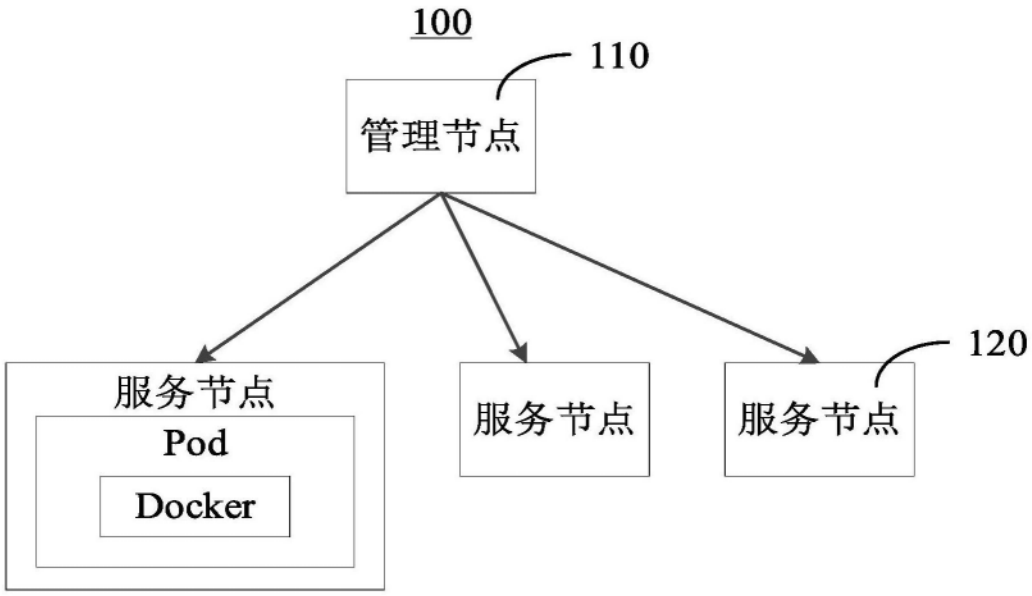

100
110
管理节点
120
服务节点
Pod
Docker
服务节点
服务节点

图1

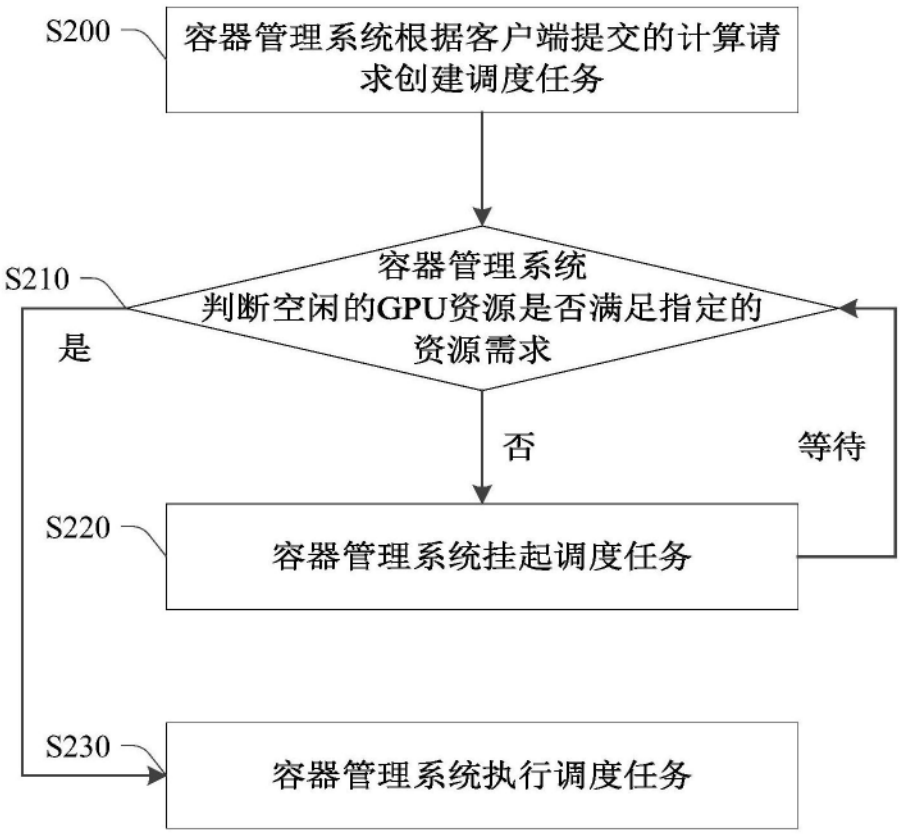

S200
容器管理系统根据客户端提交的计算请求创建调度任务
S210
容器管理系统判断空闲的GPU资源是否满足指定的资源需求
是
否
等待
S220
容器管理系统挂起调度任务
S230
容器管理系统执行调度任务

图2

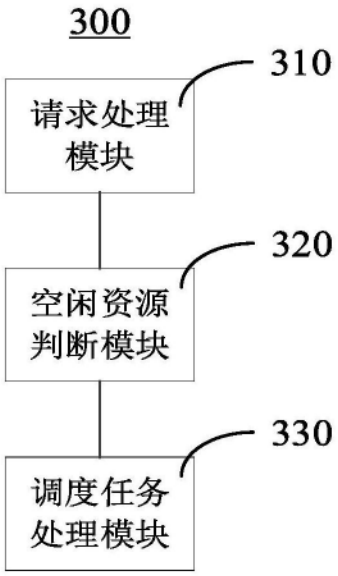

300
310
请求处理模块
320
空闲资源判断模块
330
调度任务处理模块

图3

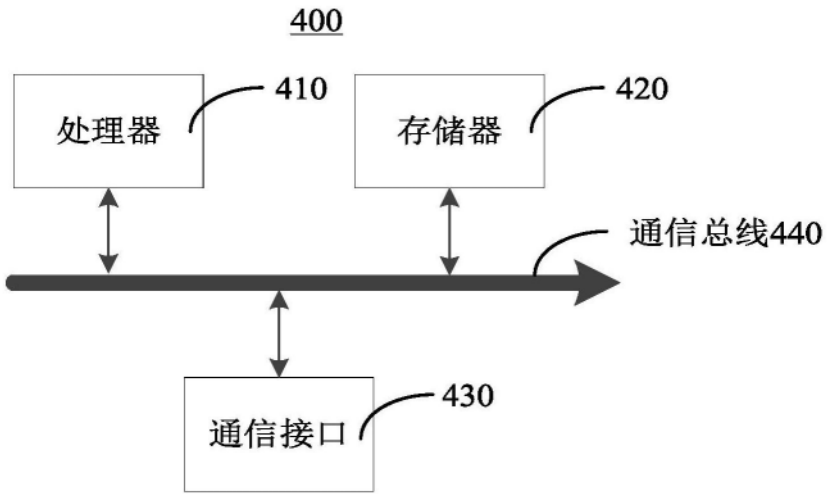

400
处理器
410
存储器
420
通信总线440
通信接口
430

图4