

GPU 多场景混部介绍

在涂鸦，GPU的主要使用场景

1. 实验场景

- 该场景特点是用户会长期占有GPU资源，但仅偶尔调试使用，资源利用率低。
- 我们主要的训练场景为单机多卡，所以用户调试的时候也需要多卡环境，即我们期望的是能够分配n个0.x卡。
- 目前开源的GPU虚拟化方案仅支持分配单个0.x卡，不支持分配n个0.x卡，所以我们需要对开源方案进行改造。

2. 训练场景

- 该场景负载高，希望能够独占整张显卡。

3. 推理场景

- 即允许单张显卡上部署多个应用。

GPU虚拟化的常见方案

分资源隔离和不隔离两种

- 资源不隔离：代表项目为阿里的GPU-Share，即简单的将单个GPU分配给多个POD，每个POD实际都能使用全部的GPU资源，需要应用自身根据环境变量中的值自己限制显存的使用。
- 资源隔离：
 - 驱动劫持，劫持libcudart.so中的API来实现资源隔离，代表项目为腾讯的vcuda，每个POD只能使用预先分配的算力和显存。
 - 内核劫持，腾讯和阿里目前主推的模式，但不开源，只能在各自的云环境中才能使用。
 - 硬件层隔离，包括MIG和vGPU等，英伟达官方提供的虚拟化方案，但仅部分高端型号支持，且需要额外付费。

我们的改进目标

我们希望达成以下目标：

- 允许用户分配两张半卡，即两张GPU可以同时给两个用户使用，且各自只能使用一半显存，算力不做限制。
- 支持三种常见的虚拟化方案，且能够混部。
- 优先将实验和推理场景的任务分配到算力低的GPU节点上。

如此不仅可以提高资源利用率，还能大幅降低运维难度。实现上，包括三个部分：

- vcuda 用于劫持 libcudart.so，以实现资源隔离。
- DevicePlugin 实现资源注册和分配。
- Scheduler 扩展插件实现 POD 的正确和优化调度。

两个半卡的实现

1. 资源隔离的实现

- 我们基于腾讯开源的 `vcuda-controller` 进行改造，移除对算力的限制，在显存分配相关的 `API` 中插入对显存的检查，会获取当前 `POD` 所有的进程 `ID`，以及显卡中所有进程的 `ID` 和已使用显存量，对比得出当前 `POD` 所占用的全部显存资源，从而确定是否已会超额。
- 原项目中始终假设为第1块 `GPU`，而我们改为会查询当前实际的 `GPU` 设备 `ID`，从而支持多 `GPU` 的显存隔离。
- `vcuda-controller` 项目自20年初就很少更新了，我们进行了 `API` 更新、加入缓存、直接通过 `socket` 与 `deviceplugin` 通信等改造。

2. DevicePlugin的实现

- 对每张物理 `GPU`，`DevicePlugin` 会向 `kubelet` 注册两份名为 `tuya.com/sgpu` 的资源。
- 分别两张半卡时，会向两张物理 `GPU` 各取一份 `tuya.com/sgpu` 资源。
- 如果一个节点只剩2份 `tuya.com/sgpu` 资源，且在同一张物理 `GPU` 上，这时分配两张半卡会失败，为避免这种情况，需要添加 `scheduler` 插件，对节点进行过滤。

混部方案的实现

- 假如一个节点有4张物理 `GPU`，`DevicePlugin` 会向 `kubelet` 同时注册以下三种资源：

```

1 Capacity:
2   nvidia.com/gpu:      4
3   tuya.com/sgpu:       8
4   tuya.com/vcuda-memory: 344
5 Allocatable:
6   nvidia.com/gpu:      4
7   tuya.com/sgpu:       8
8   tuya.com/vcuda-memory: 344
9 Allocated resources:
10  Resource              Requests      Limits
11  -----
12  nvidia.com/gpu         0             0
13  tuya.com/sgpu          0             0
14  tuya.com/vcuda-memory  0             0

```

- 假如这时有一个声明了 `tuya.com/sgpu: 2` 的 `POD` 分配到了该节点，则 `DevicePlugin` 会将其中2张物理 `GPU` 分配给该 `POD`，同时这2张物理 `GPU` 锁定为只允许分配 `tuya.com/sgpu` 资源，而从 `nvidia.com/gpu` 和 `tuya.com/vcuda-memory` 中剔除，变成：

```

1 Capacity:
2   nvidia.com/gpu:      2
3   tuya.com/sgpu:       8
4   tuya.com/vcuda-memory: 172

```

```

5 Allocatable:
6   nvidia.com/gpu:      2
7   tuya.com/sgpu:      8
8   tuya.com/vcuda-memory: 172
9 Allocated resources:
10  Resource              Requests      Limits
11  -----
12  nvidia.com/gpu        0            0
13  tuya.com/sgpu         2            2
14  tuya.com/vcuda-memory 0            0

```

这里 `nvidia.com/gpu` 和 `tuya.com/vcuda-memory` 都减半了，`tuya.com/sgpu` 不变还是等于8，所以这里的 `Allocatable` 是指所有可分配的资源，包括已分配的。

3. 假如这时再有一个声明了 `tuya.com/vcuda-memory: 10` 的 `POD` 分配到该节点，则会再有一个物理 GPU 被锁定，变成：

```

1 Capacity:
2   nvidia.com/gpu:      1
3   tuya.com/sgpu:      6
4   tuya.com/vcuda-memory: 172
5 Allocatable:
6   nvidia.com/gpu:      1
7   tuya.com/sgpu:      6
8   tuya.com/vcuda-memory: 172
9 Allocated resources:
10  Resource              Requests      Limits
11  -----
12  nvidia.com/gpu        0            0
13  tuya.com/sgpu         2            2
14  tuya.com/vcuda-memory 10           10

```

`nvidia.com/gpu` 和 `tuya.com/sgpu` 减少，而 `tuya.com/vcuda-memory` 保持不变。

4. 这时假如再分配一个独占 GPU，则变成：

```

1 Capacity:
2   nvidia.com/gpu:      1
3   tuya.com/sgpu:      4
4   tuya.com/vcuda-memory: 86
5 Allocatable:
6   nvidia.com/gpu:      1
7   tuya.com/sgpu:      4
8   tuya.com/vcuda-memory: 86
9 Allocated resources:
10  Resource              Requests      Limits
11  -----
12  nvidia.com/gpu        1            1
13  tuya.com/sgpu         2            2
14  tuya.com/vcuda-memory 10           10

```

5. 当 `POD` 被删除时，与上同理把相关资源加回去即可。

调度器扩展插件

扩展插件主要实现两个接口：

1. 过滤节点：需要对符合要求的节点进行过滤，尤其是资源看起来够，实际无法分配的节点。
2. 节点打分：包括以下规则等
 - 优先将 `tuya.com/sgpu` 和 `tuya.com/vcuda-memory` 分配到算力低的节点
 - 优先将 `tuya.com/sgpu` 分配到已分配过 `tuya.com/sgpu` 的节点
 - 优先凑整

而后在 `kube-scheduler-config.yaml` 中加入该插件的地址即可

```
1  apiVersion: kubescheduler.config.k8s.io/v1beta1
2  kind: KubeSchedulerConfiguration
3  clientConnection:
4    kubeconfig: /etc/kubernetes/scheduler.conf
5  leaderElection:
6    leaderElect: true
7  extenders:
8    - urlPrefix: "http://nvidia-device-scheduler.kube-system.svc/scheduler"
9      filterVerb: "filter"
10     prioritizeVerb: "prioritize"
11     weight: 1
12     enableHTTPS: false
13     nodeCacheCapable: false
14     ignorable: false
15     managedResources:
16       - name: "nvidia.com/gpu"
17         ignoredByScheduler: false
18       - name: "tuya.com/sgpu"
19         ignoredByScheduler: false
20       - name: "tuya.com/vcuda-core"
21         ignoredByScheduler: false
22       - name: "tuya.com/vcuda-memory"
23         ignoredByScheduler: false
```

最终分配案例：

~ ai21

GPU-POD					
device	GPU-Util	model	namespace	resource_name	pod
nvidia0	<div><div></div></div> 0%	NVIDIA GeForce GTX 1080 Ti	kebi-hi	tuya.com/sgpu	workspace-cuda10-5746465cbb-cs...
nvidia1	<div><div></div></div> 0%	NVIDIA GeForce GTX 1080 Ti	/	/	/
nvidia2	<div><div></div></div> 0%	NVIDIA GeForce GTX 1080 Ti	kebi-hi	tuya.com/sgpu	workspace-cuda10-5746465cbb-cs...
nvidia2	<div><div></div></div> 0%	NVIDIA GeForce GTX 1080 Ti	yu	tuya.com/sgpu	yu-platform-6659b66bc8-g8swf
nvidia3	<div><div></div></div> 0%	NVIDIA GeForce GTX 1080 Ti	/	/	/

~ ai22

GPU-POD					
device	GPU-Util	model	namespace	resource_name	pod
nvidia0	<div><div></div></div> 0%	NVIDIA GeForce GTX 1080 Ti	ren	tuya.com/sgpu	stylegan3-0
nvidia0	<div><div></div></div> 0%	NVIDIA GeForce GTX 1080 Ti	yu	tuya.com/sgpu	ubuntu18-cuda10-py3-59565884dd-...
nvidia1	<div><div></div></div> 100%	NVIDIA GeForce GTX 1080 Ti	ren	tuya.com/sgpu	stylegan3-0
nvidia1	<div><div></div></div> 100%	NVIDIA GeForce GTX 1080 Ti	yu	tuya.com/sgpu	ubuntu18-cuda10-py3-59565884dd-...
nvidia2	<div><div></div></div> 0%	NVIDIA GeForce GTX 1080 Ti	liuch	tuya.com/vcud...	clickhouse-server-54846dc7f7-ls5m7
nvidia2	<div><div></div></div> 0%	NVIDIA GeForce GTX 1080 Ti	zha	tuya.com/vcud...	triton-server-cy-5db67dbb49-rgqp7

实验和推理任务都分配在比较旧的GPU上，并尽可能优先分配满一个节点，再分配下一个节点