

Volcano调度增强

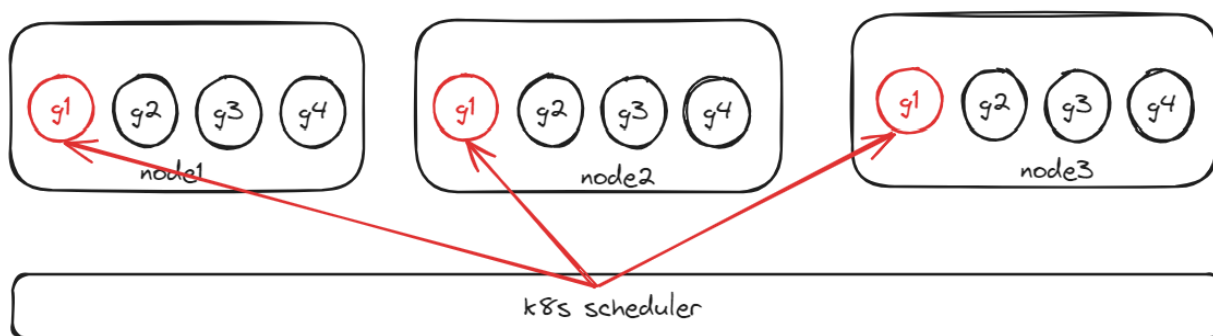
一、介绍

官方文档: [介绍](#) | Volcano

二、问题与方案

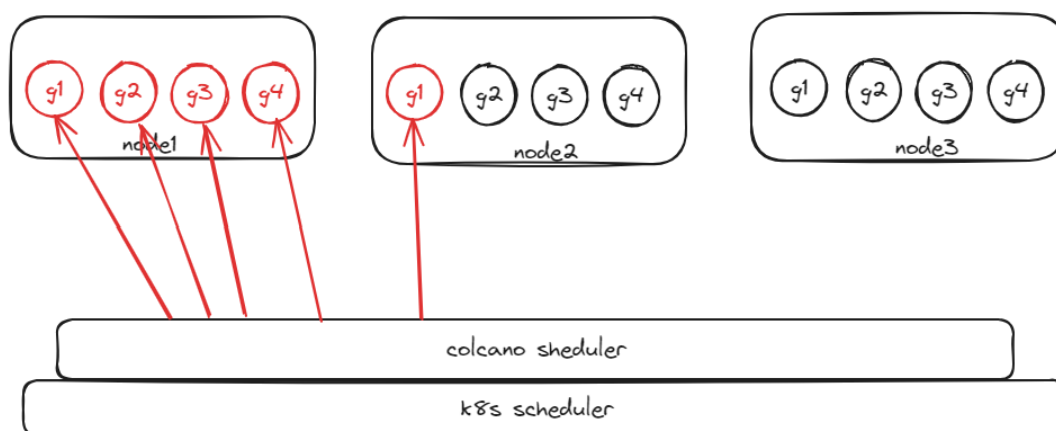
2.1 问题

在现在的 Kubernetes 调度策略中，目前分配GPU的策略比较随机，导致运行服务的Pod分布在不通的节点中，比较分散，资源管理比较困难。



2.2 目标

想借助 Volcano的运行机制，来对GPU做一个集中调度的策略。已占满一个Node为优先级， 然后进行分配第二台机器，以此类推。



从官方给出的解释来看， `Binpack` 比较符合我们的预期， 但是需要进行验证

Binpack 调度算法的目标是尽量将已有的节点填满，具体实现上，binpack调度算法是给可以投递的节点打分，分数越高表示节点的资源利用率越高。binpack算法能够尽可能填满节点，将应用负载靠拢在部分节点，这非常有利于K8S集群节点的自动扩缩容功能

引用： [装箱调度 \(Binpack\) _云容器引擎 CCE](#)

2.3 Binpack原理

Binpack在对一个节点打分时，会根据Binpack插件自身权重和各资源设置的权重值综合打分。首先，对Pod请求资源中的每类资源依次打分，以CPU为例，CPU资源在待调度节点的得分信息如下：

$\text{CPU.weight} * (\text{request} + \text{used}) / \text{allocatable}$

即CPU权重值越高，得分越高，节点资源使用量越满，得分越高。Memory、GPU等资源原理类似。其中：

- CPU.weight为用户设置的CPU权重
- request为当前pod请求的CPU资源量
- used为当前节点已经分配使用的CPU量
- allocatable为当前节点CPU可用总量

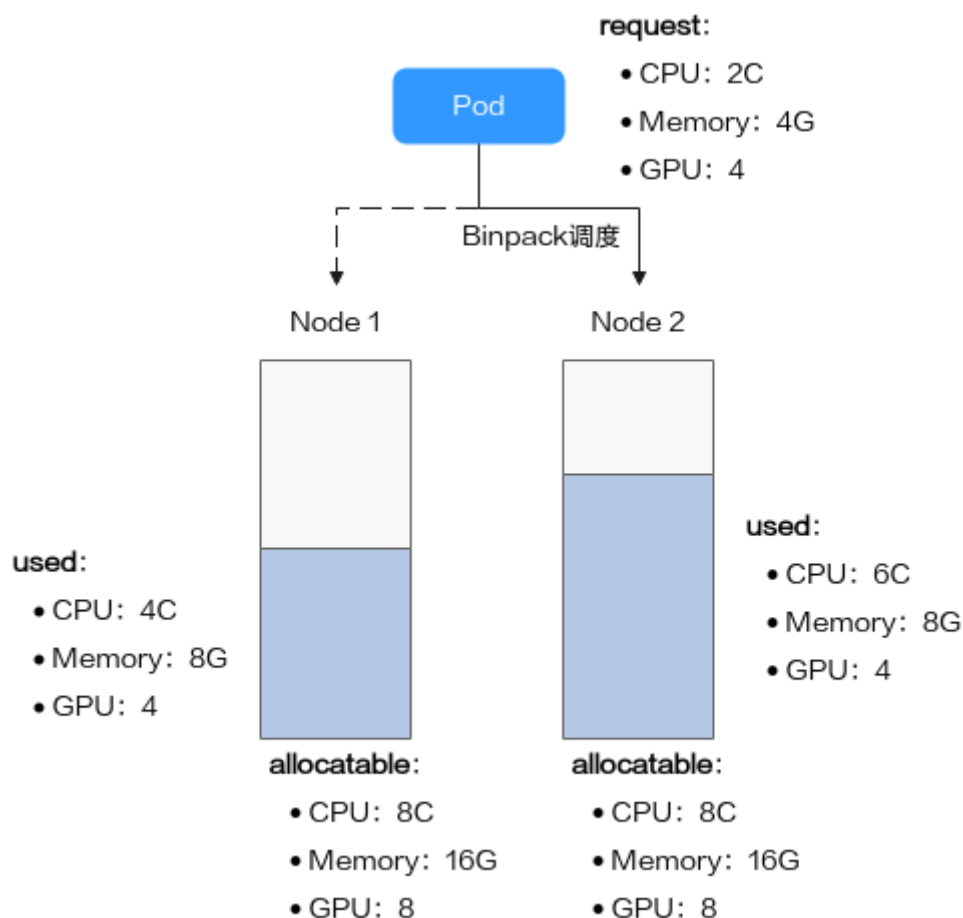
通过Binpack策略的节点总得分如下：

$\text{binpack.weight} (\text{CPU.score} + \text{Memory.score} + \text{GPU.score}) / (\text{CPU.weight} + \text{Memory.weight} + \text{GPU.weight}) 100$

即binpack插件的权重值越大，得分越高，某类资源的权重越大，该资源在打分时的占比越大。其中：

- binpack.weight为用户设置的装箱调度策略权重
- CPU.score为CPU资源得分，CPU.weight为CPU权重
- Memory.score为Memory资源得分，Memory.weight为Memory权重
- GPU.score为GPU资源得分，GPU.weight为GPU权重

图1 Binpack策略示例



如图所示，集群中存在两个节点，分别为Node 1和Node 2，在调度Pod时，Binpack策略对两个节点分别打分。

1. Binpack对Node 1的打分信息如下：

各资源按照公式 **$\text{CPU.weight} * (\text{request} + \text{used}) / \text{allocatable}$** 计算得分，具体信息如下：

- CPU Score: $1 * (2 + 4) / 8 = 0.75$
- Memory Score: $1 * (4 + 8) / 16 = 0.75$
- GPU Score: $2 * (4 + 4) / 8 = 1$

节点总得分按照 **$\text{binpack.weight} (\text{CPU.score} + \text{Memory.score} + \text{GPU.score}) / (\text{CPU.weight} + \text{Memory.weight} + \text{GPU.weight}) 100$** 公式进行计算，具体如下：

Node 1在Binpack策略下的得分： $5 * (0.75 + 0.75 + 1) / (1 + 1 + 2) 100 = 312.5$

2. Binpack对Node 2的打分信息如下：

- CPU Score: $1 * (2 + 6) / 8 = 1$

- Memory Score: $1 * (4 + 8) / 16 = 0.75$
- GPU Score: $2 * (4 + 4) / 8 = 1$

Node 2在Binpack策略下的得分: $5 * (1 + 0.75 + 1) / (1 + 1 + 2) * 100 = 343.75$

综上, Node 2得分大于Node 1, 按照Binpack策略, Pod将会优先调度至Node 2。

2.4 Binpack 配置项

名称	说明	默认值	key
装箱调度策略权重	增大该权重值, 可提高装箱策略在整体调度中的影响力。	10	weight
CPU权重	增大该权重值, 优先提高集群CPU利用率。	1	cpu
内存权重	增大该权重值, 优先提高集群Memory利用率。	1	memory
自定义资源类型	指定Pod请求的其他自定义资源类型, 例如nvidia.com/gpu。增大该权重值, 优先提高指定资源的利用率。	-	resources

```
- name: binpack
  arguments:
    binpack.weight: 10
    binpack.cpu: 5
    binpack.memory: 1
    binpack.resources: nvidia.com/gpu, nvidia.com/gpu-a800
    binpack.resources.nvidia.com/gpu: 2
    binpack.resources.nvidia.com/gpu-a800: 3
```

binpack.resources 用于自定义资源的参数 比如 `nvidia.com/gpu, nvidia.com/gpu-a800` 使用逗号分隔

在自定义 resource 之后, 我们可以对自定义资源进行配置权重, 比如:

```
binpack.resources.nvidia.com/gpu: 2
```

三、部署

采用离线部署的方式, 版本 `v1.9.0`

```
volcano
|__ volcano.tar.gz
|__ deploy.yaml
```

安装 volcano

```
tar -zxvf volcano.tar.gz
docker load -i volcano.tar
kubectl apply -f deploy.yaml
```

注意：安装过程中有一个 job 任务，需要执行，但是需要重新拉取镜像，注意镜像地址可达

我们采用 binpack 的方式对 pod 进行集中化处理

```
kubectl edit cm volcano-scheduler-configmap -n volcano-system
```

配置一下 volcano 调度的策略

```
- name: binpack
  arguments:
    binpack.weight: 10
    binpack.cpu: 5
    binpack.memory: 1
    binpack.resources: nvidia.com/gpu, nvidia.com/gpu-a800
    binpack.resources.nvidia.com/gpu: 2
    binpack.resources.nvidia.com/gpu-a800: 3
```

四、验证

4.1 场景一：调度增强

场景描述：针对 deployment 使用 volcano 调度策略，可以在满足条件的基础上进行集中调度，对比 k8s 调度结果

4.1.1 开启 Volcano 调度增强

创建一个 deployment 选用 volcano 进行调度

```
# test.yml
apiVersion: apps/v1
kind: Deployment
metadata:
```

```

name: volcano-demo
spec:
  replicas: 10
  selector:
    matchLabels:
      app: volcano-demo
  template:
    metadata:
      labels:
        app: volcano-demo
    spec:
      schedulerName: volcano # 选择 volcano 进行调度
      containers:
        - name: nginx
          image: docker.m.daocloud.io/nginx:latest
          imagePullPolicy: IfNotPresent
          ports:
            - containerPort: 80
      resources:
        requests:
          cpu: 300m

```

```
kubectl apply -f test.yml
```

预期结果: 在部署节点没有到使用资源超限之前，该node会一直被调用应用上去

实际结果: 新建的pod 都在master 这个节点中，符合预期

```

[root@master volcano]# kubectl get pod -o wide
NAME                                READY    STATUS    RESTARTS   AGE    IP              NODE    NOMINATED NODE    READINESS GATES
volcano-demo-f67c44675-22xm5        1/1      Running   0           6m17s  10.233.70.59    master  <none>             <none>
volcano-demo-f67c44675-6z5wh        1/1      Running   0           5m45s  10.233.70.64    master  <none>             <none>
volcano-demo-f67c44675-ffjh6        1/1      Running   0           5m45s  10.233.70.61    master  <none>             <none>
volcano-demo-f67c44675-jmrgm        1/1      Running   0           5m45s  10.233.70.62    master  <none>             <none>
volcano-demo-f67c44675-k2tgp        1/1      Running   0           5m45s  10.233.70.65    master  <none>             <none>
volcano-demo-f67c44675-qht5f        1/1      Running   0           5m45s  10.233.70.66    master  <none>             <none>
volcano-demo-f67c44675-rl5hz        1/1      Running   0           5m45s  10.233.70.60    master  <none>             <none>
volcano-demo-f67c44675-rst94        1/1      Running   0           5m45s  10.233.70.63    master  <none>             <none>
volcano-demo-f67c44675-ww8hm        1/1      Running   0           5m45s  10.233.70.67    master  <none>             <none>
volcano-demo-f67c44675-xglls        1/1      Running   0           6m17s  10.233.70.58    master  <none>             <none>

```

4.1.2 禁用Volcano仅使用默认调度

```

# test-k8s.yml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: volcano-k8s-demo
spec:
  replicas: 10
  selector:
    matchLabels:
      app: volcano-k8s-demo
  template:
    metadata:

```

```

labels:
  app: volcano-k8s-demo
spec:
  containers:
  - name: nginx
    image: docker.m.daocloud.io/nginx:latest
    imagePullPolicy: IfNotPresent
    ports:
    - containerPort: 80
  resources:
    requests:
      cpu: 300m

```

```
kubectl apply -f test-k8s.yml
```

验证结果: 将 `sheduleName:volcano` 去掉，直接使用k8s自带的调度，从图可以看出，调度随机到了 master 和 node01 上

```

[root@master volcano]# kubectl get pod -o wide | grep k8s
volcano-k8s-demo-656d46795-2dsfr    0/1    ContainerCreating    0        17s    <none>    node01    <none>    <none>
volcano-k8s-demo-656d46795-4mrjg    0/1    ContainerCreating    0        17s    <none>    node01    <none>    <none>
volcano-k8s-demo-656d46795-55zcf    0/1    ContainerCreating    0        17s    <none>    node01    <none>    <none>
volcano-k8s-demo-656d46795-9cjlt    1/1    Running              0        2m21s   10.233.70.69    master    <none>    <none>
volcano-k8s-demo-656d46795-cdqwh    0/1    ContainerCreating    0        17s    <none>    node01    <none>    <none>
volcano-k8s-demo-656d46795-cgqr2    1/1    Running              0        2m21s   10.233.70.71    master    <none>    <none>
volcano-k8s-demo-656d46795-dqb7v    0/1    ContainerCreating    0        17s    <none>    node01    <none>    <none>
volcano-k8s-demo-656d46795-kxt7p    1/1    Running              0        2m22s   10.233.70.68    master    <none>    <none>
volcano-k8s-demo-656d46795-n7cqf    1/1    Running              0        2m21s   10.233.70.70    master    <none>    <none>
volcano-k8s-demo-656d46795-pbqdf    1/1    Running              0        17s     10.233.70.72    master    <none>    <none>

```

4.2 场景二: 调度增强、超限验证

4.2.1 基础环境准备

创建一个指定node 部署的基础环境, 在节点 node03 中部署 app=overload-demo的服务

节点	pod
node03	overload-demo

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: overload-demo
spec:
  replicas: 6
  selector:
    matchLabels:
      app: overload-demo
  template:
    metadata:
      labels:
        app: overload-demo

```

```
spec:
  nodeSelector:
    kubernetes.io/hostname: node03
  schedulerName: volcano
  containers:
  - name: nginx
    image: docker.m.daocloud.io/nginx:latest
    imagePullPolicy: IfNotPresent
    ports:
      - containerPort: 80
  resources:
    requests:
      cpu: "1000m"
      memory: "2Gi"
    limits:
      cpu: "1000m"
      memory: "2Gi"
```

```
kubectl apply -f base-env.yml
```

此时我们创建的服务都在node03中，此时node03的整体的利用率拉到最高，无法满足再次分配的条件

名称	状态	角色	CPU 用量	内存用量	容器组	已分配 CPU	已分配内存
node03 10.1.205.162	运行中	工作节点	2% 0.18/8 核	8% 1.28/15.48 GiB	13% 14/110	6.962 核 (91%) 资源预留	12.703 GiB (89%) 资源预留
node02 10.1.205.161	运行中	工作节点	2% 0.16/8 核	7% 1.1/15.48 GiB	7% 8/110	0.962 核 (12%) 资源预留	0.703 GiB (4%) 资源预留
node01 10.1.205.157	运行中	工作节点	3% 0.26/8 核	15% 2.36/15.48 GiB	24% 26/110	1.567 核 (20%) 资源预留	1.357 GiB (9%) 资源预留
master 10.1.205.158	运行中	控制平面节点	9% 0.7/8 核	26% 4.01/15.48 GiB	21% 23/110	6.652 核 (87%) 资源预留	10.09 GiB (70%) 资源预留

4.2.2 节点资源超限的情况下，开启Volcano

验证目标： 开启volcano调度增强后，原本的资源分配策略仍然起到作用

```
# out-volcano.yml

apiVersion: apps/v1
kind: Deployment
metadata:
  name: out-volcano-demo
spec:
  replicas: 2
  selector:
```



```

matchLabels:
  app: out-volcano-demo
template:
  metadata:
    labels:
      app: out-volcano-demo
  spec:
    schedulerName: volcano
    containers:
    - name: nginx
      image: docker.m.daocloud.io/nginx:latest
      imagePullPolicy: IfNotPresent
      ports:
      - containerPort: 80
    resources:
      requests:
        cpu: "1000m"
        memory: "2Gi"

```

部署服务

```
kubectl apply -f out-volcano.yml
```

验证结果： 我们可以看到，node03 的使用率是最高的，本应该优先分配，但是再次分配要求的资源已经不满足了，所以分配策略顺延到第二使用了高的节点 `master` 中了，符合预期

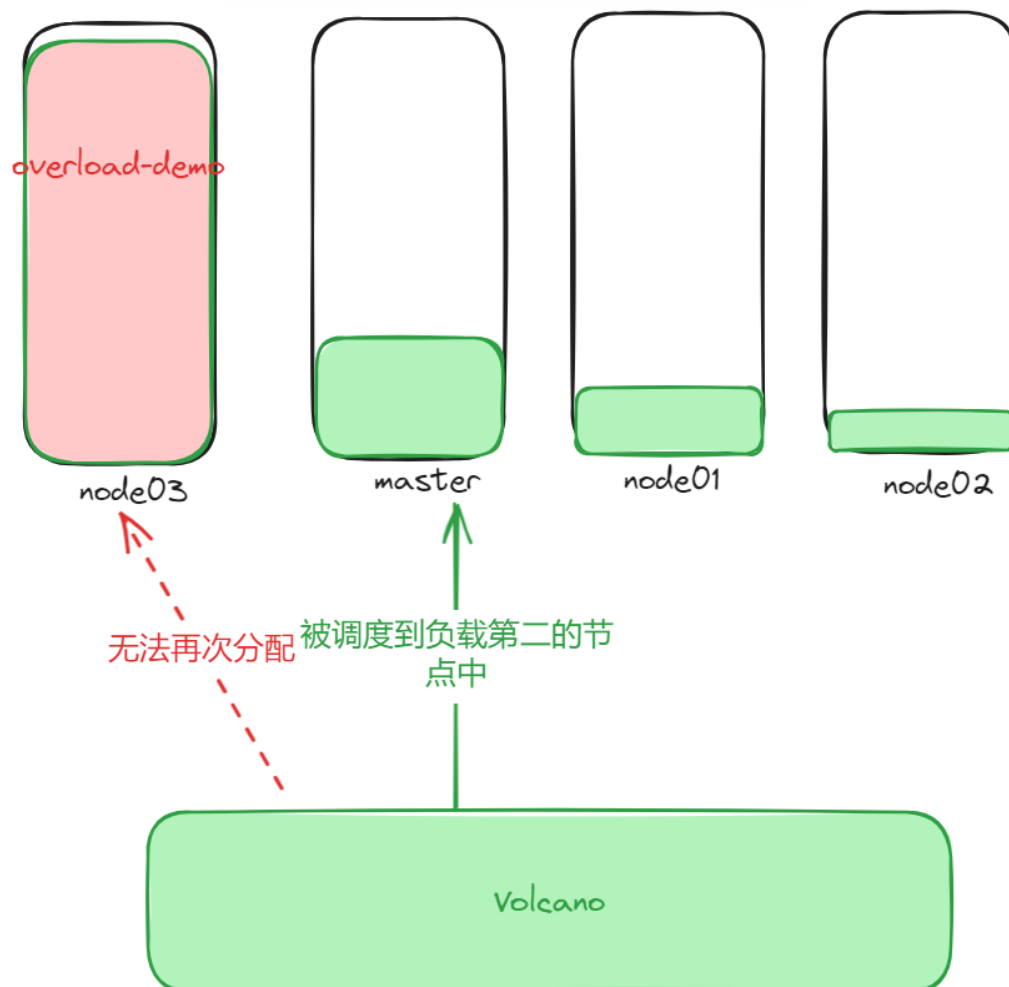
```

[root@master volcano]# kubectl get pod -o wide
NAME                                READY   STATUS    RESTARTS   AGE   IP              NODE   NOMINATED NODE   READINESS GATES
out-volcano-demo-56756f947d-2rqzm   1/1     Running   0           4m    10.233.70.79    master <none>         <none>
out-volcano-demo-56756f947d-88fqz   1/1     Running   0           4m    10.233.70.80    master <none>         <none>
overload-demo-79f6784f76-6lr7q      1/1     Running   0           4m58s 10.233.69.12    node03 <none>         <none>
overload-demo-79f6784f76-7dp2b      1/1     Running   0           5m30s 10.233.69.11    node03 <none>         <none>
overload-demo-79f6784f76-7ps8s      1/1     Running   0           5m30s 10.233.69.10    node03 <none>         <none>
overload-demo-79f6784f76-jh9th      1/1     Running   0           24m    10.233.69.9     node03 <none>         <none>
overload-demo-79f6784f76-qn5rw      1/1     Running   0           24m    10.233.69.8     node03 <none>         <none>
overload-demo-79f6784f76-xb494      1/1     Running   0           4m58s 10.233.69.13    node03 <none>         <none>

```

4.2.3 场景结论

开启Volcano之后，原本的资源分配的策略仍然起作用，Volcano只是扩展了计算方式



基础环境搭建后，各node 排序的结果为如上图所示，node03负载最高，按照 binpack的算法在不考虑资源可用率的情况下，node03 将是优先调度的节点。但是 node03 中的可用资源无法再次分配出一个部署环境，volcano 降级到第二个利用率高的节点中，所以落在了 `master` 这个node中。

4.3 场景三: 调度增强、开启反亲和验证

4.3.1 基础环境准备

创建一个指定node 部署的基础环境, 在节点 node03 中部署 app=overload-demo的服务

节点	pod
node03	overload-demo

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: overload-demo
spec:
  replicas: 2
```

```

selector:
  matchLabels:
    app: overload-demo
template:
  metadata:
    labels:
      app: overload-demo
  spec:
    nodeSelector:
      kubernetes.io/hostname: node03
    schedulerName: volcano
    containers:
      - name: nginx
        image: docker.m.daocloud.io/nginx:latest
        imagePullPolicy: IfNotPresent
        ports:
          - containerPort: 80
        resources:
          requests:
            cpu: "1000m"
            memory: "2Gi"
          limits:
            cpu: "1000m"
            memory: "2Gi"

```

```
kubectl apply -f base-env.yml
```

此时我们创建的服务都在node03中，此时node03的整体的利用率是最高的

<input type="checkbox"/>	名称 ▾	状态 ▾	角色 ▾	CPU 用量	内存用量	容器组	已分配 CPU	已分配内存	
<input type="checkbox"/>	 node03 10.1.205.162	 运行中	工作节点	2% 0.17/8 核	7% 1.15/15.48 GiB	9% 10/110	2.962 核 (38%) 资源预留	4.703 GiB (33%) 资源预留	⋮
<input type="checkbox"/>	 node02 10.1.205.161	 运行中	工作节点	2% 0.15/8 核	7% 1.1/15.48 GiB	7% 8/110	0.962 核 (12%) 资源预留	0.703 GiB (4%) 资源预留	⋮
<input type="checkbox"/>	 node01 10.1.205.157	 运行中	工作节点	3% 0.26/8 核	15% 2.35/15.48 GiB	24% 26/110	1.567 核 (20%) 资源预留	1.357 GiB (9%) 资源预留	⋮
<input type="checkbox"/>	 master 10.1.205.158	 运行中	控制平面节点	8% 0.63/8 核	25% 3.95/15.48 GiB	19% 21/110	2.652 核 (34%) 资源预留	2.09 GiB (14%) 资源预留	⋮

4.3.2 设置反亲和性的情况下，开启Volcano

验证目标： 开启volcano调度增强后，原来的反亲和策略仍然有效

```

# anti-affinity-volcano.yml
apiVersion: apps/v1
kind: Deployment
metadata:

```

```

name: anti-overload-demo
spec:
  replicas: 2
  selector:
    matchLabels:
      app: anti-overload-demo
  template:
    metadata:
      labels:
        app: anti-overload-demo
    spec:
      schedulerName: volcano
      containers:
      - name: nginx
        image: docker.m.daocloud.io/nginx:latest
        imagePullPolicy: IfNotPresent
        ports:
        - containerPort: 80
      resources:
        requests:
          cpu: "1000m"
          memory: "2Gi"
      affinity:
        podAntiAffinity:
          preferredDuringSchedulingIgnoredDuringExecution:
          - weight: 100
            podAffinityTerm:
              labelSelector:
                matchExpressions:
                - key: app
                  operator: In
                  values:
                  - overload-demo
              topologyKey: "kubernetes.io/hostname"

```

部署服务

```
kubectl apply -f anti-affinity-volcano.yml
```

验证结果： 我们可以看到，因为node03 上存在 overload-demo 的pod，新增的容器被分配到了使用率第二的 `master` 节点中，符合预期

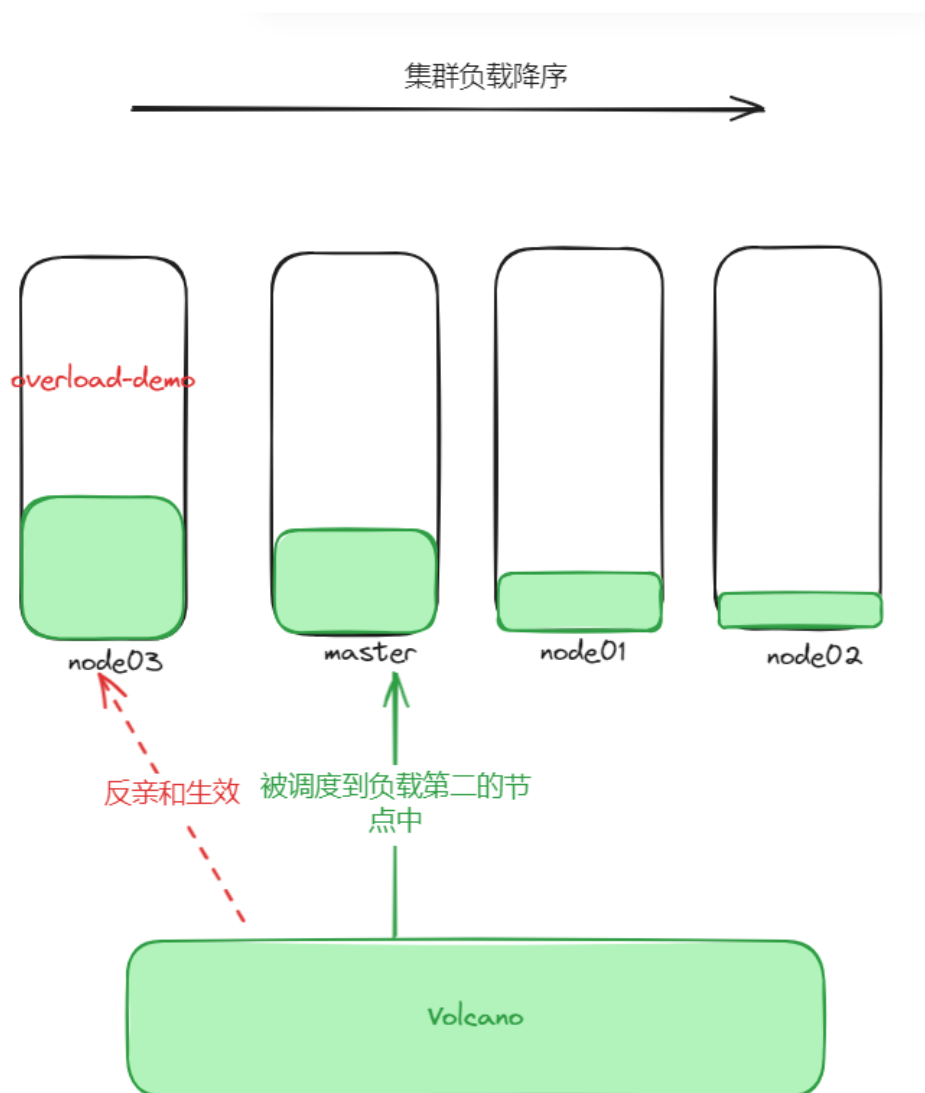
```

[root@master volcano]# kubectl get pod -o wide
NAME                                READY   STATUS    RESTARTS   AGE   IP              NODE   NOMINATED NODE   READINESS GATES
anti-overload-demo-6cb99cd8fd-lnglt 1/1     Running   0           6s    10.233.70.78    master <none>          <none>
anti-overload-demo-6cb99cd8fd-lnhtx 1/1     Running   0           6s    10.233.70.77    master <none>          <none>
overload-demo-79f6784f76-jh9th       1/1     Running   0          3m13s  10.233.69.9     node03 <none>          <none>
overload-demo-79f6784f76-qnsrw       1/1     Running   0          3m17s  10.233.69.8     node03 <none>          <none>

```

4.3.3 场景结论

在开启Volcano增强的时候，原本的K8S调度的反亲和性的策略仍然有效



基础环境搭建后，各node 排序的结果为如上图所示，node03负载最高，按照 binpack的算法在不考虑反亲和性的条件下，node03 将是优先调度的节点。但是node03中存在 `overload-demo` 的pod，触发了反亲和性策略，volcano 降级到第二个利用率高的节点中，所以落在了 `master` 这个node中。

4.4 验证总结

我们从上述的验证可以看出，开启 Volcano 调度增强后，原本的策略仍然有效，不管超限无法分配降级，还是反亲和策略的触发。不难看出 Volcano 的策略是做了 Plugin 的形式接入到 Kubernetes 中。使用方式比较优雅，侵入性小。符合我们针对NPU调度的场景。