K8S学习笔记之如何让多个Pod均匀部署到各个节点上

Jettytomcat 2023-03-23 15:42:27 博主文章分类：K8S学习笔记

Kubernetes中kube-scheduler组件负责Pod的调度，对每一个新创建的 Pod 或者是未被调度的 Pod，kube-scheduler 会选择一个最优的节点去运行这个 Pod。

kube-scheduler 给一个 Pod 做调度选择包含过滤和打分两个步骤。

过滤阶段会将所有满足 Pod 调度需求的节点选出来，在打分阶段 kube-scheduler 会给每一个可调度节点进行优先级打分，最后kube-scheduler 会将 Pod 调度到得分最高的节点上，如果存在多个得分最高的节点，kube-scheduler 会从中随机选取一个。

打分优先级中节点调度均衡（BalancedResourceAllocation）只是其中一项，还有其他打分项会导致分布不均匀。详细的调度说明请参见 Kubernetes 调度器和 调度策略。

想要让多个Pod尽可能的均匀分布在各个节点上，可以考虑使用工作负载反亲和特性，让Pod之间尽量“互斥”，这样就能尽量均匀的分布在各节点上。

示例如下：

```yaml
kind: Deployment
apiVersion: apps/v1
metadata:
  name: nginx
  namespace: default
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: container-0
          image: nginx:alpine
          resources:
            limits:
              cpu: 250m
              memory: 512Mi
            requests:
              cpu: 250m
              memory: 512Mi
      affinity:
        podAntiAffinity:                    # 工作负载反亲和
          preferredDuringSchedulingIgnoredDuringExecution:    # 尽量满足如下条件
            - podAffinityTerm:
                labelSelector:                        # 选择Pod的标签，与工作负载本身反亲和
                  matchExpressions:
                    - key: app
                      operator: In
                      values:
                        - nginx
                namespaces:
```

```
                  - default
              topologyKey: kubernetes.io/hostname     # 在节点上起作用
      imagePullSecrets:
        - name: default-secret
```

k8s解决pod调度不均衡的问题

发布于 2020-07-31 15:29:29

问题及原因

k8s是通过sceduler来调度pod的，在调度过程中，由于一些原因，会出现调度不均衡的问题，例如：

节点故障

新节点被加到集群中

节点资源利用不足

这些都会导致pod在调度过程中分配不均，例如会造成节点负载过高，引发pod触发OOM等操作造成服务不可用

其中，节点资源利用不足时是最容易出现问题的，例如，设置的requests和limits不合理，或者没有设置requests/limits都会造成调度不均衡

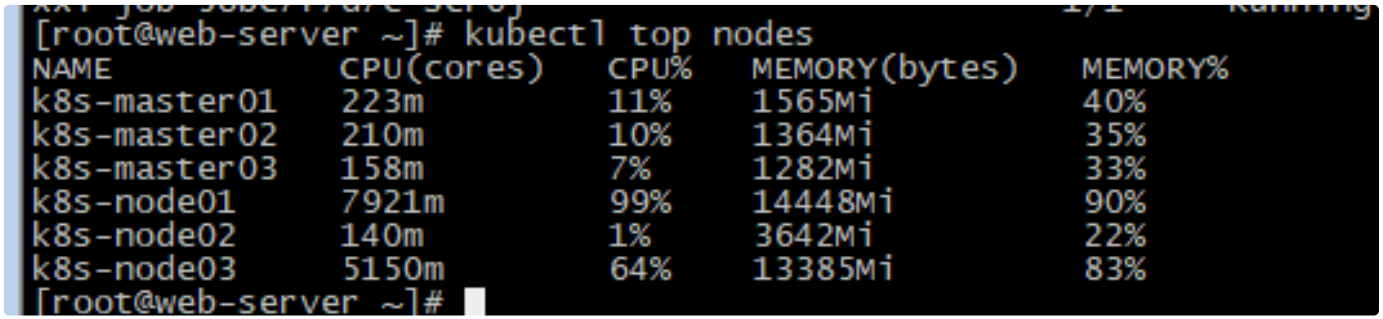
解决办法及分析

在这之前，我们需要先装一个metrics，安装方法可参考：k8s的metrics部署

Scheduler在调度过程中，经过了预选阶段和优选阶段，选出一个可用的node节点，来把pod调度到该节点上。那么在预选阶段和优选阶段是如何选出node节点的呢?

最根本的一个调度策略就是判断节点是否有**可分配**的资源，我们可以通过以下 `kubectl describe node node名` 来查看，现在按照这个调度策略来分析下

查看当前的节点资源占用情况



可以看到，当前的k8s集群共有三个node节点，但是节点的资源分布情况极其不均匀，而实际上，k8s在进行调度时，计算的就是requests的值，不管你limits设置多少，k8s都不关心。所以当这个值没有达到资源瓶颈时，理论上，该节点就会一直有pod调度上去。所以这个时候就会出现调度不均衡的问题。有什么解决办法?

给每一个pod设置requests和limits，如果资源充足，最好将requests和limits设置成一样的，提高Pod的QoS

重平衡，采取人为介入或定时任务方式，根据多维度去对当前pod分布做重平衡

重平衡工具Descheduler

工具简介

Descheduler 的出现就是为了解决 Kubernetes 自身调度（一次性调度）不足的问题。它以定时任务方式运行，根据已实现的策略，重新去平衡 pod 在集群中的分布。

截止目前，Descheduler 已实现的策略和计划中的功能点如下：

已实现的调度策略

RemoveDuplicates 移除重复 pod

LowNodeUtilization 节点低度使用

RemovePodsViolatingInterPodAntiAffinity 移除违反pod反亲和性的 pod

RemovePodsViolatingNodeAffinity

路线图中计划实现的功能点

Strategy to consider taints and tolerations 考虑污点和容忍

Consideration of pod affinity 考虑 pod 亲和性

Strategy to consider pod life time 考虑 pod 生命周期

Strategy to consider number of pending pods 考虑待定中的 pod 数量

Integration with cluster autoscaler 与集群自动伸缩集成

Integration with metrics providers for obtaining real load metrics 与监控工具集成来获取真正的负载指标

Consideration of Kubernetes’s scheduler’s predicates 考虑 k8s 调度器的预判机制

策略介绍

RemoveDuplicates 此策略确保每个副本集(RS)、副本控制器(RC)、部署(Deployment)或任务(Job)只有一个 pod 被分配到同一台 node 节点上。如果有多个则会被驱逐到其它节点以便更好的在集群内分散 pod。

LowNodeUtilization a. 此策略会找到未充分使用的 node 节点并在可能的情况下将那些被驱逐后希望重建的 pod 调度到该节点上。 b. 节点是否**利用不足**由一组可配置的 **阈值**(`thresholds`) 决定。这组阈值是以百分比方式指定了 CPU、内存以及 pod数量 的。只有当所有被评估资源都低于它们的阈值时，该 node 节点才会被认为处于**利用不足**状态。 c. 同时还存在一个 **目标阈值**(`targetThresholds`)，用于评估那些节点是否因为超出了阈值而应该从其上驱逐 pod。任何阈值介于 `thresholds` 和 `targetThresholds` 之间的节点都被认为资源被合理利用了，因此不会发生 pod 驱逐行为（无论是被驱逐走还是被驱逐来）。 d. 与之相关的还有另一个参数 `numberOfNodes` ，这个参数用来激活指定数量的节点是否处于资源**利用不足**状态而发生 pod 驱逐行为。

RemovePodsViolatingInterPodAntiAffinity 此策略会确保 node 节点上违反 pod 间亲和性的 pod 被驱逐。比如节点上有 podA 并且 podB 和 podC（也在同一节点上运行）具有禁止和 podA 在同一节点上运行的反亲和性规则，则 podA 将被从节点上驱逐，以便让 podB 和 podC 可以运行。

RemovePodsViolatingNodeAffinity 此策略会确保那些违反 node 亲和性的 pod 被驱逐。比如 podA 运行在 nodeA 上，后来该节点不再满足 podA 的 node 亲和性要求，如果此时存在 nodeB 满足这一要求，则 podA 会被驱逐到 nodeB 节点上。

遵循机制

当 Descheduler 调度器决定于驱逐 pod 时，它将遵循下面的机制：

Critical pods (with annotations scheduler.alpha.kubernetes.io/critical-pod) are never evicted 关键 pod（带注释 scheduler.alpha.kubernetes.io/critical-pod）永远不会被驱逐。

Pods (static or mirrored pods or stand alone pods) not part of an RC, RS, Deployment or Jobs are never evicted because these pods won't be recreated 不属于RC，RS，部署或作业的Pod（静态或镜像pod或独立pod）永远不会被驱逐，因为这些pod不会被重新创建。

Pods associated with DaemonSets are never evicted 与 DaemonSets 关联的 Pod 永远不会被驱逐。

Pods with local storage are never evicted 具有本地存储的 Pod 永远不会被驱逐。

BestEffort pods are evicted before Burstable and Guaranteed pods QoS 等级为 BestEffort 的 pod 将会在等级为 Burstable 和 Guaranteed 的 pod 之前被驱逐。

部署方式

Descheduler 会以 Job 形式在 pod 内运行，因为 Job 具有多次运行而无需人为介入的优势。为了避免被自己驱逐 Descheduler 将会以 **关键型** pod 运行，因此它只能被创建建到 `kube-system` namespace 内。 关于 Critical pod 的介绍请参考：Guaranteed Scheduling For Critical Add-On Pods

要使用 Descheduler，我们需要编译该工具并构建 Docker 镜像，创建 ClusterRole、ServiceAccount、ClusterRoleBinding、ConfigMap 以及 Job。

yaml文件下载地址：https://github.com/kubernetes-sigs/descheduler

```
git clone https://github.com/kubernetes-sigs/descheduler.git
```

Run As A Job

```
kubectl create -f kubernetes/rbac.yaml
kubectl create -f kubernetes/configmap.yaml
kubectl create -f kubernetes/job.yaml
```

Run As A CronJob

```
kubectl create -f kubernetes/rbac.yaml
kubectl create -f kubernetes/configmap.yaml
kubectl create -f kubernetes/cronjob.yaml
```

两种方式，一种是以任务的形式启动，另一种是以计划任务的形式启动，建议以计划任务方式去启动

启动之后，可以来验证下descheduler是否启动成功

```
# kubectl get pod -n kube-system | grep descheduler
descheduler-job-6qtk2                      1/1     Running   0          158m
```

再来验证下pod是否分布均匀

```
[root@web-server kubernetes]# kubectl get pod -o wide | grep -c node01
26
[root@web-server kubernetes]# kubectl get pod -o wide | grep -c node02
20
[root@web-server kubernetes]# kubectl get pod -o wide | grep -c node03
27
```

可以看到，目前node02这个节点的pod数是20个，相比较其他节点，还是差了几个，那么我们只对pod数量做重平衡的话，可以对descheduler做如下的配置修改

```
# cat kubernetes/configmap.yaml
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: descheduler-policy-configmap
  namespace: kube-system
data:
  policy.yaml: |
    apiVersion: "descheduler/v1alpha1"
    kind: "DeschedulerPolicy"
    strategies:
      "RemoveDuplicates":
         enabled: true
      "RemovePodsViolatingInterPodAntiAffinity":
         enabled: true
      "LowNodeUtilization":
         enabled: true
         params:
           nodeResourceUtilizationThresholds:
             thresholds:    #阈值
               #"cpu" : 20    #注释掉下面这些关于cpu和内存的配置项
```

```
      #"memory": 20
      "pods": 24    #把pod的数值调高一些
    targetThresholds:    #目标阈值
      #"cpu" : 50
      #"memory": 50
      "pods": 25
```

修改完成后，重启下即可

```
kubectl delete -f kubernetes/configmap.yaml
kubectl apply -f kubernetes/configmap.yaml
kubectl delete -f kubernetes/cronjob.yaml
kubectl apply -f kubernetes/cronjob.yaml
```

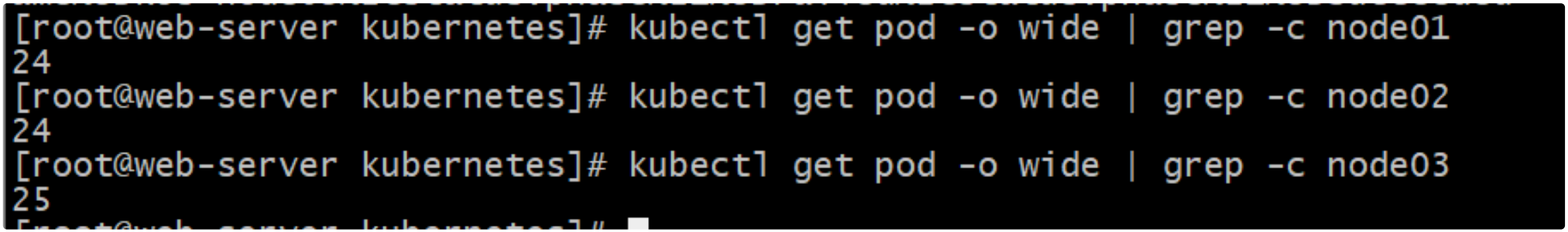
然后，看下Descheduler的调度日志

```
# kubectl logs -n kube-system descheduler-job-9rc9h
I0729 08:48:45.361655       1 lownodeutilization.go:151] Node "k8s-node02" is under utilized with usage: api.ResourceThresholds{"cpu":44.375, "
I0729 08:48:45.361772       1 lownodeutilization.go:154] Node "k8s-node03" is over utilized with usage: api.ResourceThresholds{"cpu":49.375, "m
I0729 08:48:45.361807       1 lownodeutilization.go:151] Node "k8s-master01" is under utilized with usage: api.ResourceThresholds{"cpu":50, "mer
I0729 08:48:45.361828       1 lownodeutilization.go:151] Node "k8s-master02" is under utilized with usage: api.ResourceThresholds{"cpu":40, "mer
I0729 08:48:45.361863       1 lownodeutilization.go:151] Node "k8s-master03" is under utilized with usage: api.ResourceThresholds{"cpu":40, "mer
I0729 08:48:45.361977       1 lownodeutilization.go:154] Node "k8s-node01" is over utilized with usage: api.ResourceThresholds{"cpu":46.875, "m
I0729 08:48:45.361994       1 lownodeutilization.go:66] Criteria for a node under utilization: CPU: 0, Mem: 0, Pods: 23
I0729 08:48:45.362016       1 lownodeutilization.go:73] Total number of underutilized nodes: 4
I0729 08:48:45.362025       1 lownodeutilization.go:90] Criteria for a node above target utilization: CPU: 0, Mem: 0, Pods: 23
I0729 08:48:45.362033       1 lownodeutilization.go:92] Total number of nodes above target utilization: 2
I0729 08:48:45.362051       1 lownodeutilization.go:202] Total capacity to be moved: CPU:0, Mem:0, Pods:55.2
I0729 08:48:45.362059       1 lownodeutilization.go:203] ********Number of pods evicted from each node:***********
I0729 08:48:45.362066       1 lownodeutilization.go:210] evicting pods from node "k8s-node01" with usage: api.ResourceThresholds{"cpu":46.875,
I0729 08:48:45.362236       1 lownodeutilization.go:213] allPods:30, nonRemovablePods:3, bestEffortPods:2, burstablePods:25, guaranteedPods:0
I0729 08:48:45.362246       1 lownodeutilization.go:217] All pods have priority associated with them. Evicting pods based on priority
I0729 08:48:45.381931       1 evictions.go:102] Evicted pod: "flink-taskmanager-7c7557d6bc-ntnp2" in namespace "default"
I0729 08:48:45.381967       1 lownodeutilization.go:270] Evicted pod: "flink-taskmanager-7c7557d6bc-ntnp2"
I0729 08:48:45.381980       1 lownodeutilization.go:283] updated node usage: api.ResourceThresholds{"cpu":46.875, "memory":32.25716687667426, "
I0729 08:48:45.382268       1 event.go:278] Event(v1.ObjectReference{Kind:"Pod", Namespace:"default", Name:"flink-taskmanager-7c7557d6bc-ntnp2",
I0729 08:48:45.399567       1 evictions.go:102] Evicted pod: "flink-taskmanager-7c7557d6bc-t2htk" in namespace "default"
I0729 08:48:45.399613       1 lownodeutilization.go:270] Evicted pod: "flink-taskmanager-7c7557d6bc-t2htk"
I0729 08:48:45.399626       1 lownodeutilization.go:283] updated node usage: api.ResourceThresholds{"cpu":46.875, "memory":32.25716687667426, "
```

```
I0729 08:48:45.400503       1 event.go:278] Event(v1.ObjectReference{Kind:"Pod", Namespace:"default", Name:"flink-taskmanager-7c7557d6bc-t2htk"
I0729 08:48:45.450568       1 evictions.go:102] Evicted pod: "oauth-center-tools-api-645d477bcf-hnb8g" in namespace "default"
I0729 08:48:45.450603       1 lownodeutilization.go:270] Evicted pod: "oauth-center-tools-api-645d477bcf-hnb8g"
I0729 08:48:45.450619       1 lownodeutilization.go:283] updated node usage: api.ResourceThresholds{"cpu":45.625, "memory":31.4545002047819, "p
I0729 08:48:45.451240       1 event.go:278] Event(v1.ObjectReference{Kind:"Pod", Namespace:"default", Name:"oauth-center-tools-api-645d477bcf-h
I0729 08:48:45.477605       1 evictions.go:102] Evicted pod: "dazzle-core-api-5d4c899b84-xhlkl" in namespace "default"
I0729 08:48:45.477636       1 lownodeutilization.go:270] Evicted pod: "dazzle-core-api-5d4c899b84-xhlkl"
I0729 08:48:45.477649       1 lownodeutilization.go:283] updated node usage: api.ResourceThresholds{"cpu":44.375, "memory":30.65183353288954, "
I0729 08:48:45.477992       1 event.go:278] Event(v1.ObjectReference{Kind:"Pod", Namespace:"default", Name:"dazzle-core-api-5d4c899b84-xhlkl", U
I0729 08:48:45.523774       1 request.go:557] Throttling request took 141.499557ms, request: POST:https://10.96.0.1:443/api/v1/namespaces/defau
I0729 08:48:45.569073       1 evictions.go:102] Evicted pod: "live-foreignapi-api-7bc679b789-z8jnr" in namespace "default"
I0729 08:48:45.569105       1 lownodeutilization.go:270] Evicted pod: "live-foreignapi-api-7bc679b789-z8jnr"
I0729 08:48:45.569119       1 lownodeutilization.go:283] updated node usage: api.ResourceThresholds{"cpu":43.125, "memory":29.84916686099718, "
I0729 08:48:45.569151       1 lownodeutilization.go:236] 6 pods evicted from node "k8s-node01" with usage map[cpu:43.125 memory:29.849166860997
I0729 08:48:45.569172       1 lownodeutilization.go:210] evicting pods from node "k8s-node03" with usage: api.ResourceThresholds{"cpu":49.375,
I0729 08:48:45.569418       1 lownodeutilization.go:213] allPods:27, nonRemovablePods:2, bestEffortPods:0, burstablePods:25, guaranteedPods:0
I0729 08:48:45.569430       1 lownodeutilization.go:217] All pods have priority associated with them. Evicting pods based on priority
I0729 08:48:45.603962       1 event.go:278] Event(v1.ObjectReference{Kind:"Pod", Namespace:"default", Name:"live-foreignapi-api-7bc679b789-z8jn
I0729 08:48:45.639483       1 evictions.go:102] Evicted pod: "dazzle-contentlib-api-575f599994-khdn5" in namespace "default"
I0729 08:48:45.639512       1 lownodeutilization.go:270] Evicted pod: "dazzle-contentlib-api-575f599994-khdn5"
I0729 08:48:45.639525       1 lownodeutilization.go:283] updated node usage: api.ResourceThresholds{"cpu":48.125, "memory":26.26225017097819, "
I0729 08:48:45.645446       1 event.go:278] Event(v1.ObjectReference{Kind:"Pod", Namespace:"default", Name:"dazzle-contentlib-api-575f599994-kh
I0729 08:48:45.780324       1 evictions.go:102] Evicted pod: "dazzle-datasync-task-577c46668-lltg4" in namespace "default"
I0729 08:48:45.780544       1 lownodeutilization.go:270] Evicted pod: "dazzle-datasync-task-577c46668-lltg4"
I0729 08:48:45.780565       1 lownodeutilization.go:283] updated node usage: api.ResourceThresholds{"cpu":46.875, "memory":25.45958349908583, "
I0729 08:48:45.780600       1 lownodeutilization.go:236] 4 pods evicted from node "k8s-node03" with usage map[cpu:46.875 memory:25.459583499085
I0729 08:48:45.780620       1 lownodeutilization.go:102] Total number of pods evicted: 11
```

通过这个日志，可以看到

Node “k8s-node01” is over utilized，然后就是有提示evicting pods from node “k8s-node01”，这就说明，Descheduler已经在重新调度了，最终调度结果如下：



node.js kubernetes http