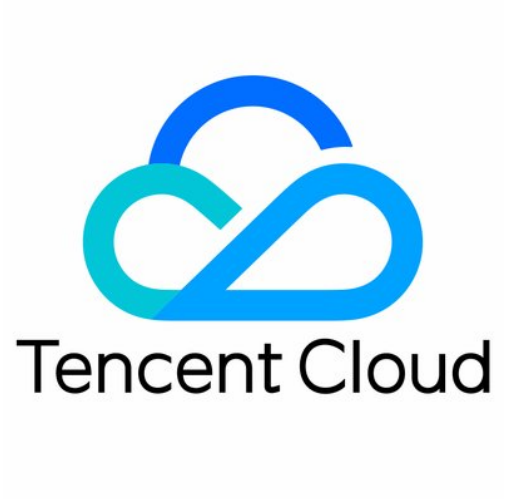
Tencent Kubernetes Engine

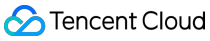
TKE Scheduling

Product Documentation



Copyright Notice

Trademark Notice



Service Statement

This document is intended to provide users with general information about Tencent Cloud's products and services only and does not form part of Tencent Cloud's terms and conditions. Tencent Cloud's products or services are subject to change. Specific products and services and the standards applicable to them are exclusively provided for in Tencent Cloud's applicable terms and conditions.

Contents

TKE Scheduling
Job Scheduling
Request Recommendation

Last updated：2024-03-04 15:26:47

Overview

Component Overview

Kubernetes can efficiently improve business orchestration capabilities and resource utilization. With no additional capabilities for support, this enhancement remains substantially limited. The average resource utilization of a TKE node is merely about 14% according to the previous statistics by the TKE team.
The main reason for the poor resource utilization rate of a Kubernetes cluster is adherence to Kubernetes' resource scheduling logic. When creating Kubernetes workloads, it's typical to configure suitable resource Requests and Limits for the workload, indicating resource concession and restriction. Among these, the Requests have the most significant impact on the utilization rate. To prevent the resources employed by their workload from being occupied by others, or to cater to the resource demands during peak traffic, users tend to set larger values for Request. The disparity between the Requests and the actual utilized resources cannot be employed by other workloads, resulting in wastage. The unreasonable setting of Request values leads to a low resource utilization rate in the Kubernetes cluster.
Tencent Kubernetes Engine (TKE) supports the installation of **Request Recommendation** component in the cluster. Request Recommendation allows for the suggestion of Request/Limit values for container-level resources in Kubernetes workloads, reducing resource wastage.

Resource objects deployed in a cluster

By enabling Request Recommendation in a cluster, it will deploy the following Kubernetes objects within a cluster:

Kubernetes Object Name	Type	Default Resource Occupation	Associated Namespace
analytics.analysis.crane.io	CustomResourceDefinition	-	-
recommendations.analysis.crane.io	CustomResourceDefinition	-	-
crane-system	Namespace	-	-
housekeeper-default	Analytics	-	crane-system

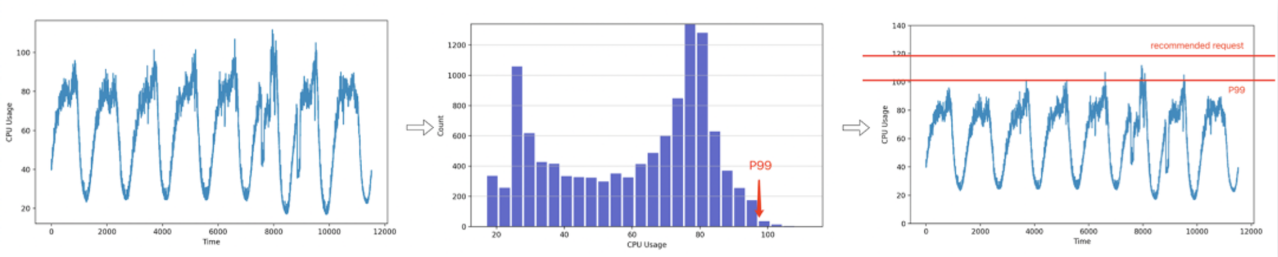
recommendation-config	ConfigMap	-	crane-system
craned	ClusterRole	-	-
craned	ClusterRoleBinding	-	-
craned	Service	-	crane-system
craned	ServiceAccount	-	crane-system
craned	Deployment	-	crane-system

Feature Overview

- It supports recommending suitable Request/Limit values of resources for each container in Deployment, StatefulSet, and DaemonSet.
- It supports one-click update of the resource values for containers in the initial workload with recommended values.
- It supports maintaining the Request/Limit ratio. The recommended Request/Limit will preserve the proportion between the Reqeust/Limit in the initial Workload Container Settings. If the Limit is not set upon Workload creation, a Limit recommendation won't be provided.
- The console's one-click update capability for Request recommendations will add a nodeSelector attribute to the workload by default. During workload updates, Pods can only be scheduled on native nodes. If native node resources are insufficient, it will lead to a pending of the Pod.

Principles of Request Recommendation

- The component creates an Analytics CR object under the crane-system Namespace, covering all native Kubernetes workloads (Deployment, DaemonSet, StatefulSet) in all clusters. It analyzes workload data for up to 14 days, updating recommended values every 12 hours.
- It then produces a Recommendation CR object for each workload within the cluster based on Analytics, purposed for data storage of recommendations.
- If recommendation CR generates recommendation data, it will inscribe this information into the corresponding workload's Annotation.



Notes

Environment Requirements

Kubernetes version: 1.10+

Node Requirements

The **One-Click Update Workload Request** feature in the Tencent Kubernetes Engine Console will migrate the workload to the native node. If your cluster's native node lacks resources, it could result in a pending of the Pod.

Requirements on the Controlled Resources

- It supports Deployment, StatefulSet, and DaemonSet.
- It does not support Job and CronJob, as well as the Pods that are not managed by a workload.

Recommended Threshold

Suggested minimum values: The recommended minimum value for CPU per container is 0.125 core, i.e. 125 m; the minimum memory is 125 Mi.

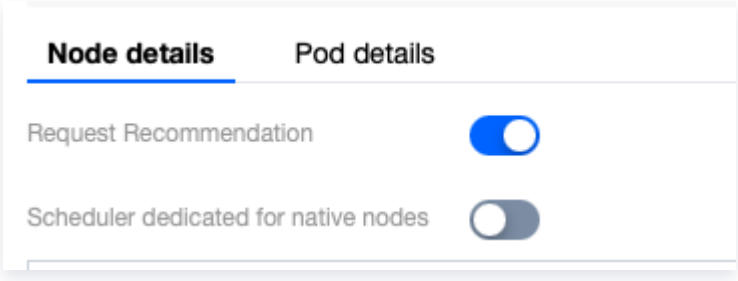
Instructions

Installing a Component

1. Log in to the Tencent Kubernetes Engine Console.
2. Select **TKE Insight > Node Map** on the left.

> **Note:**
> You can also undertake the installation in **TKE Insight > Workload Map**.

3. On the Node Map page, hover your mouse over a Node at the bottom of the page, and click **Details**.
4. In the top right corner of the Node details page, enable the **Request Recommendation** switch to configure the scheduler's parameters.



> **Note:**

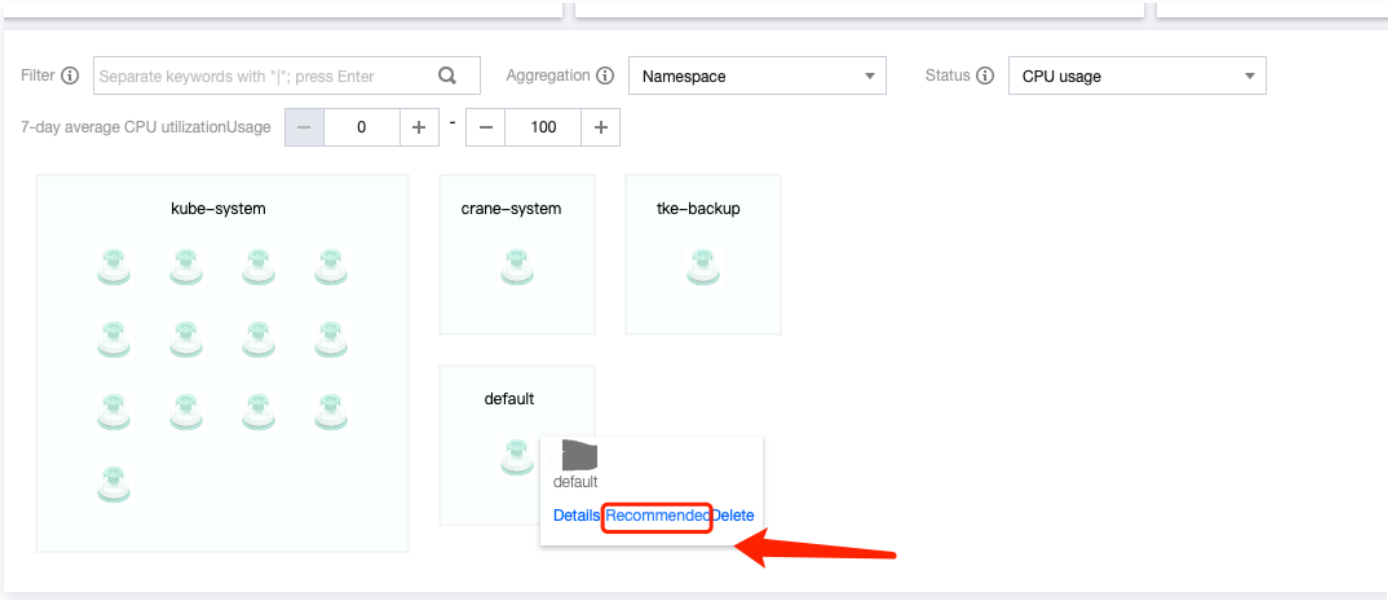
- This feature comprises a global switch at the cluster level. After the feature is enabled, it will automatically analyze the historical monitoring data of workloads to recommend appropriate Request values.
- This feature does not take effect immediately after enabling. The system will analyze the resource usage history to provide accurate recommended values.
- The period for calculation may vary for different workloads. One workload within a cluster may potentially impact another.
- After this feature is enabled, values will be recommended for the workloads that run at least for one day.
- For workloads created after this feature is enabled, it usually takes one day to recommend values.
- It is recommended to update the Workload with the recommended values after the workload has been running stably for a while.

Using a Component

1. Log in to the Tencent Kubernetes Engine Console.
2. Select **TKE Insight > Workload Map** on the left.

> **Note:**
> Workload Map mainly displays various states and metrics of workloads through a visual interface, assisting users in comprehending the current configuration volume of the workload and its actual usage, thereby helping in analyzing potential issues within the workload. For more information, see the Workload Map documentation.

3. On the Workload Map page, hover your mouse over a workload at the bottom of the page, and click **Recommended**.



4. In the pop-up window, click **Confirm** to use the suggested Request value for updating the original value in the Workload.

> **Note:**
> The **One-Click Update Workload Request** feature in the Tencent Kubernetes Engine Console will migrate the workload to the native nodes. If your native nodes in your cluster lack resources, it will result in a pending of the Pod.

Accessing Recommended Values in the Background

The Request Recommendation engine stores the recommended values in the YAML file of each workload. You can use the standard Kubernetes API to access these recommended values for each workload and then integrate them into your business's deployment system. The following demonstrates how to peek into the recommended Request amount for each container under a workload:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  annotations:
    analysis.crane.io/resource-recommendation: |
      containers:
      # If a Pod contains multiple containers, each container has
recommended values for CPU and Memory Request
      - containerName: nginx
```

```
        target:
          cpu: 125m
          memory: 125Mi #If unit is missing herein, a character string
"58243235" will be displayed, with byte as the omitted unit
```

> **Note:**
> The component itself does not recommend a Limit. When updating the Workload using the Request recommendation value in the console, it will maintain the ratio of the Workload's Request and Limit to ensure the Quality of Service (QoS) remains constant. If you access the recommended value of the Request in the background, you can consider it as a reference to update the resource configuration of the original Workload.

Component Permission Description

Permission Description

The permission of this component is the minimal dependency required for the current feature to operate.

Permission Scenarios

Feature	Involved Object	Involved Operation Permission
Recording the oom record of the pod	pod	get/list/watch
Searching and recommending idle nodes based on the node	node	get/list/watch
It is required to record the exception information in the form of events.	event	create/patch/update
Monitoring changes in related recommendation resources, and making resource recommendation	analysis.crane.io	All permissions

Permission Definition

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: craned
rules:
```

```
- apiGroups:
  - ""
  resources:
  - configmaps
  - pods
  - nodes
  verbs:
  - get
  - list
  - watch
- apiGroups:
  - analysis.crane.io
  resources:
  - "*"
  verbs:
  - "*"
- apiGroups:
  - apps
  resources:
  - daemonsets
  - deployments
  - deployments/scale
  - statefulsets
  - statefulsets/scale
  verbs:
  - get
  - list
  - watch
- apiGroups:
    - apps
  resources:
    - daemonsets/status
    - deployments/status
    - deployments/scale
    - statefulsets/status
    - statefulsets/scale
  verbs:
    - update
- apiGroups:
  - autoscaling
```

```
  resources:
  - horizontalpodautoscalers
  verbs:
  - '*'
- apiGroups:
  - autoscaling.crane.io
  resources:
  - '*'
  verbs:
  - '*'
- apiGroups:
  - ""
  resources:
  - events
  verbs:
  - create
  - patch
  - update
- apiGroups:
  - prediction.crane.io
  resources:
  - '*'
  verbs:
  - '*'
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: craned
  namespace: crane-system
rules:
  - apiGroups:
      - ""
    resources:
      - configmaps
      - secrets
    verbs:
      - create
  - apiGroups:
      - ""
```

```
    resourceNames:
      - craned
    resources:
      - configmaps
    verbs:
      - get
      - patch
      - update
  - apiGroups:
      - ""
    resourceNames:
      - clusters-secret-store
    resources:
      - secrets
    verbs:
      - get
  - apiGroups:
      - coordination.k8s.io
    resources:
      - leases
    verbs:
      - get
      - patch
      - update
      - create
```

Preemptible Job

Last updated：2024-12-24 15:20:57

Overview

In Kubernetes clusters, more businesses can run on a single node through containerization. However, this also leads to competition for resources among businesses. To address Requests are used to ensure the minimum amount of resources a Pod can claim to prevent insufficient resources during resource competition. However, the resources claimed through Requests are permanently "occupied" and cannot be used by other businesses, leading to significant waste during traffic troughs. Therefore, Tencent Kubernetes Engine (TKE) has introduced preemptible Jobs, which use idle resources in the cluster and do not occupy the actual remaining schedulable resources of the cluster/nodes. In the event of resource competition, these resources will be prioritized for reclamation to ensure the stability of services that normally use Request resources.

Resource Objects Deployed in the Cluster

Note:
To use preemptible Jobs, you need to first install the QoSAgent Component in the cluster and enable CPU Usage Priority. We will automatically assign the lowest priority (priority 7) to all preemptible Jobs, ensuring they are preempted during resource competition. For the resource objects deployed by this component in the cluster, please refer to the table below.

Kubernetes Object Name	Type	Default Resource Occupation	Associated Namespace
avoidanceactions.ensurance.crane.io	CustomResourceDefinition	–	–
nodeqoss.ensurance.crane.io	CustomResourceDefinition	–	–
podqoss.ensurance.crane.io	CustomResourceDefinition	–	–
timeseriespredictions.prediction.crane.io	CustomResourceDefinition	–	–
kube-system	Namespace	–	–
all-be-pods	PodQOS	–	kube-system

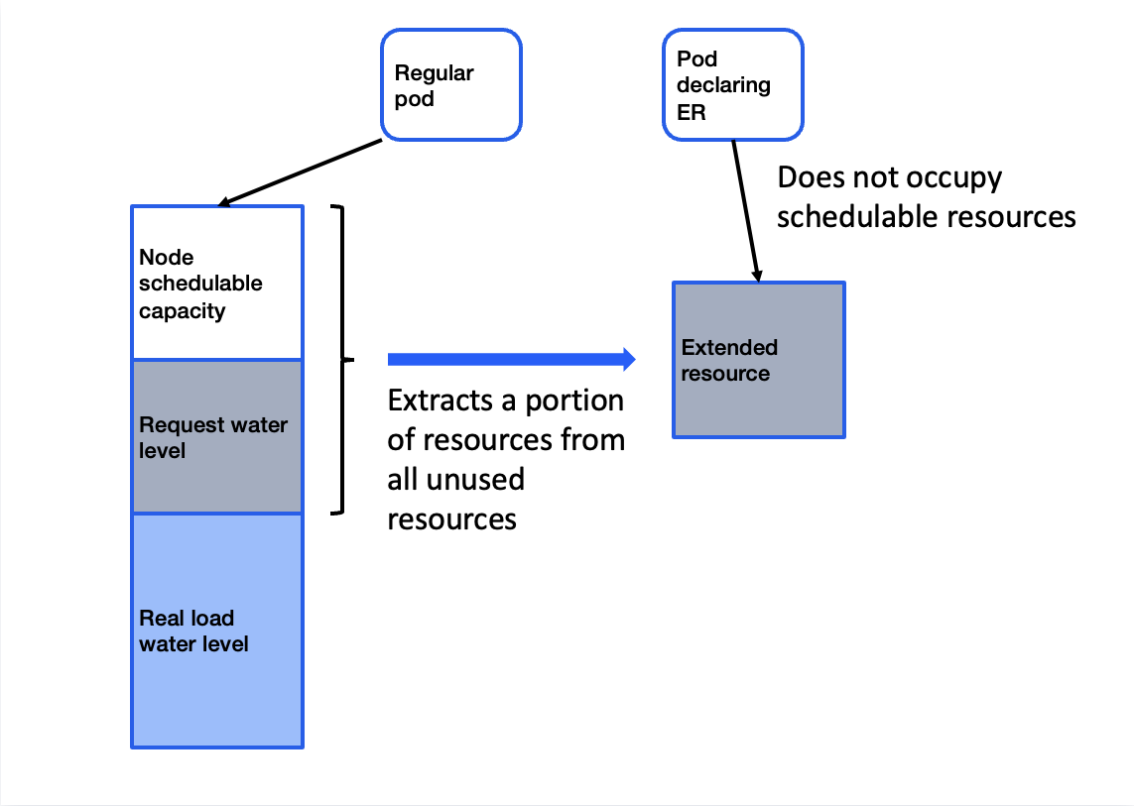
qos-agent	ClusterRole	-	-
qos-agent	ClusterRoleBinding	-	-
crane-agent	Service	-	kube-system
qos-agent	ServiceAccount	-	kube-system
qos-agent	Daemonset	-	kube-system

Overview

- Since only native nodes have idle resources, this feature is only supported on Native Nodes.
- Supports allocating idle resources in the cluster for Jobs and CronJobs.
- These idle resources do not occupy the remaining schedulable resources in the cluster.
- In the event of resource competition, idle resources will be prioritized for reclamation, hence they are called Preemptible Jobs.
- The values of idle resources for nodes and clusters are dynamically changing based on actual load. When the idle resources of a cluster/node are less than the resources requested by the Job, the Pod will be in a Pending state.

Principles of Request Recommendation

- The allocation of idle resources is implemented using Kubernetes' native Extended Resource. By recycling the remaining available resources of the nodes, a shared resource pool is formed. The shared resource pool does not occupy the actual schedulable CPU/RAM resources of the cluster/node.
- As shown in the diagram below, the QoS Agent component extracts a portion of resources from all unused resources in the cluster. Based on the historical load profile of the nodes, it predicts a safe value of Extended Resources that can be used by Job/CronJob that declare the use of Extended Resources.



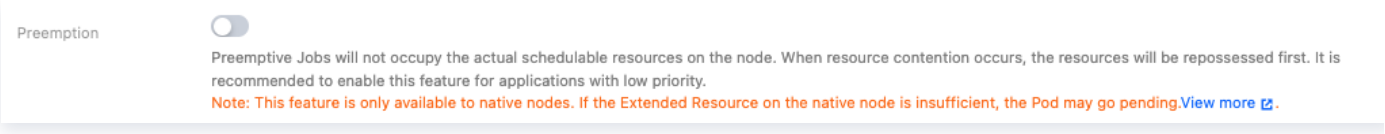
Using Preemptible Jobs

Installing Add-ons

1. Log in to the Tencent Kubernetes Engine console, select **Cluster** in the left sidebar.
2. On the cluster management page, click the ID of the target cluster to go to the cluster details page.
3. In the left sidebar, select **Add-On Management** to go to the add-on list page.
4. On the add-on list page, select **Create**, and on the Create Add-On page, check **QoS Agent**.
5. Click on **Complete** to install the add-on.

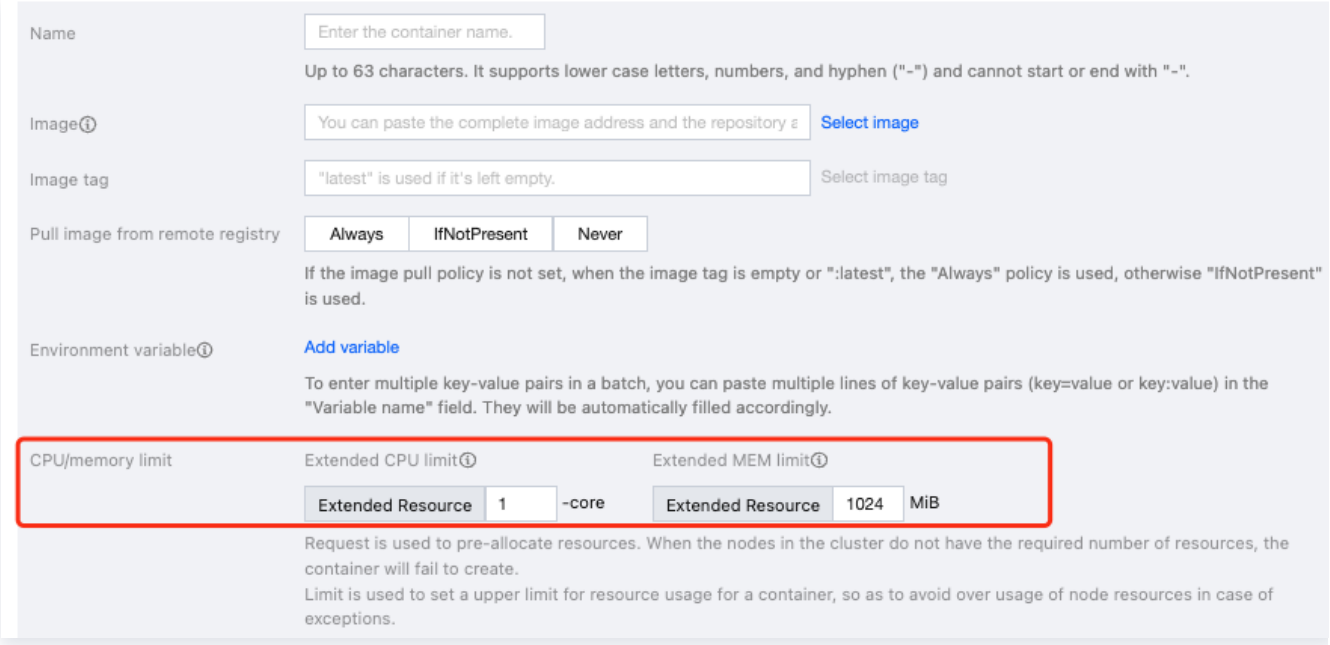
Deploying Preemptible Jobs

1. Log in to the Tencent Kubernetes Engine console, select **Cluster** in the left sidebar.
2. On the cluster management page, click the ID of the target cluster to go to the cluster details page.
3. Select **Workload > Job/CronJob**, and on the Job/CronJob resource list page, click **Create**.
4. Enable the **Preemptible** feature on the Create Add-On page. See the figure below:

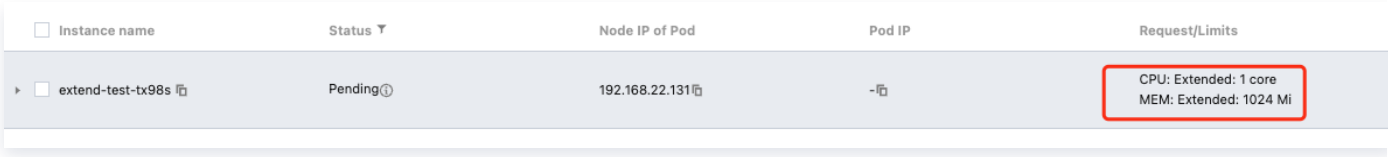


After the preemptible capability is enabled, the way resources input for Job/CronJob has changed. It is no longer necessary to input Request/Limit. Instead, it uses Extended CPU and Extended Memory:

- **Extended CPU**: Apply for idle CPU resources in native nodes. The Extended CPU limit can only be an integer.
- **Extended Memory**: Apply for idle memory resources in native nodes. The Extended Memory limit can only be an integer. See the figure below:



5. A preemptible type of Job created by declaring the use of Extended Resource's idle resources:



> **Note:**
> If you do not have native nodes or if the native nodes do not have enough idle resources, you might encounter the following errors: "Insufficient gocrane.io/memory", indicating not enough idle memory resources; or "Insufficient gocrane.io/cpu", indicating not enough idle CPU resources. In such cases, you need to increase the number of native nodes.

Native Node Dedicated Scheduler Overview

Last updated：2024-04-24 15:55:36

Overview

Component Overview

Kubernetes' scheduling logic operates **based on the Pod's Request**. The schedulable resources on the node are occupied by the Pod's Request amount and cannot free up. The native node dedicated scheduler is a scheduling plugin developed by Tencent Kubernetes Engine (TKE) based on the native Kube-scheduler Extender mechanism of Kubernetes, which can virtually magnify the capacity of the node, resolving the issue of the node's resources being occupied while maintaining a low utilization rate.

Kubernetes objects deployed in a cluster

Kubernetes Object Name	Type	Requested Resource	Belonging Namespace
crane-scheduler-controller	Deployment	Each instance is endowed with 200m CPU and 200Mi memory, totaling one instance	kube-system
crane-descheduler	Deployment	Each instance is endowed with 200m CPU and 200Mi memory, totaling one instance	kube-system
crane-scheduler	Deployment	Each instance is endowed with 200m CPU and 200Mi memory, totaling three instances	kube-system
crane-scheduler-controller	Service	-	kube-system
crane-scheduler	Service	-	kube-system
crane-scheduler	ClusterRole	-	kube-system

crane-descheduler	ClusterRole	-	kube-system
crane-scheduler	ClusterRoleBinding	-	kube-system
crane-descheduler	ClusterRoleBinding	-	kube-system
crane-scheduler-policy	ConfigMap	-	kube-system
crane-descheduler-policy	ConfigMap	-	kube-system
ClusterNodeResourcePolicy	CRD	-	-
CraneSchedulerConfiguration	CRD	-	-
NodeResourcePolicy	CRD	-	-
crane-scheduler-controller-mutating-webhook	MutatingWebhookConfiguration	-	-

Application Scenarios

Scenario 1: Resolving the issue of high node box rate but low utilization

> **Note:**
> The fundamental concepts are as follows.
> **Box Rate**: It refers to the ratio of the sum of Requests of all Pods on a node to the actual specifications of the node.
> **Utilization**: It refers to the ratio of the total actual usage of all Pods on a node to the actual specifications of the node.

The native Kubernetes scheduler schedules based on the Request resources of Pod. Therefore, even if the actual usage on the node is low at this time, if the sum of Requests of all Pods on the node is close to the actual specifications of the node, new Pods cannot be scheduled, resulting in substantial resource waste. Moreover, businesses tend to apply for surplus resources to ensure the stability of their services, that is, a large Request, leading to the occupation of node resources, unable to free up. At this point, the node's box rate is substantial, but the actual resource utilization is comparatively low.
At such times, you can use the dedicated native node scheduler to virtually enhance the specifications of CPU and memory on a node, thus amplifying its scheduler resources. More pods can thereby be scheduled.

Scenario 2: Setting the watermark of the nodes

The watermark setting of the node is to ensure the stability of the node and set the node's target utilization rate:

- Control of the watermark during scheduling: This step determines the native node's target resource utilization rate to guarantee stability. While scheduling the Pods, nodes with resources above this watermark will not be selected. Moreover, from nodes meeting the watermark requirements, as shown in the following figure, nodes with lower actual load watermarks have priority to balance the cluster node's utilization distribution.
- Control of the watermark during runtime: This step determines the current target resource utilization rate for native nodes to guarantee stability. At runtime, nodes with resources above this watermark could trigger evictions. Given that eviction is a high-risk action, bear in mind the following notes.

Notes

1. To avoid draining important Pods, this feature is set not to evict Pods by default. For Pods that can be safely drained, it is essential for users to explicitly determine the workload to which the Pod belongs. For example, StatefulSet, Deployment, and other objects can be set as drainable annotations:

   ```
   descheduler.alpha.kubernetes.io/evictable: 'true'
   ```

2. It is recommended to enable event persistence for the cluster, to better monitor component abnormalities and troubleshoot. When evicting a Pod, corresponding events will be generated. You can observe if the Pod is being repeatedly evicted based on the Descheduled event.
3. The eviction action has requirements for nodes: a cluster is required to have 3 or more low-load native nodes, where a low-load definition refers to a Node's load that is lesser than its operational water-level control.
4. After filtering at the node dimension, evacuation begins on the workload on the Node. This necessitates the constraint that the replica count of the workload should be equal to or greater than 2, or at least half of the Workload spec replicas.
5. At the Pod dimension level, if a Pod's load exceeds the eviction watermark of the node, eviction is forbidden to prevent the overloading of other nodes by relocating them there.

Scenario 3: Pods under specified Namespace shall be allocated only to native nodes upon the subsequent scheduling

Native nodes, the newly-launched node types, are introduced by the TKE Tencent Kubernetes Engine team of Tencent Cloud. They are built upon the technical excellence derived from Tencent Cloud's tens of millions of core container operations, thereby delivering native-like, high-stability, and rapid-response K8s node management capabilities. Native nodes, with amplifiable node specifications and recommended Request capabilities, are hence highly advisable for exploiting its advantages fully by scheduling your workload to

them. While enabling the native node scheduler, you can opt for Namespace. Consequently, Pods under the specified Namespace shall be scheduled exclusively to native nodes in the following scheduling.

> **Note:**
> If the native node resources are insufficient at this stage, it would result in Pod Pending.

Limits

- This feature is only supported by the native node. For more information, see Native Node Overview.
- It is required to ensure that the Kubernetes version is v1.22.5-tke.8, v1.20.6-tke.24, v1.18.4-tke.28,v1.16.3-tke.30 or higher. For cluster versions upgrade, see Upgrading a Cluster.

Risk Control

- After the uninstallation of this component, only the scheduling logic associated with the native node-dedicated scheduler will be eliminated, leaving the scheduling capability of the native Kube-Scheduler untouched. The already scheduled Pods on the native node will not be affected due to their pre-set schedule. However, a reboot of kubelet on the native node might trigger Pod eviction as the sum of Pods' Requests on the native node could exceed the genuine specifications of the native node.
- In the event of the amplification coefficient being adjusted downwards, the existing Pods on the native node, due to their already prescribed schedule, will remain unaffected. Nonetheless, if the kubelet on the native node restarts, it might trigger Pod eviction since the aggregate of Pods' Requests on the native node could surpass the amplified specifications of the native node after the amplification.
- Users witness the inconsistency between Node resources in the Kubernetes cluster and corresponding CVM node resources.
- In the future, issues related to excessive load and instability could possibly arise.
- After the amplification of the node specifications, the node kubelet layer and the resource QoS-related modules might be affected. For instance, kubelet's binding cores, when a 4-core node is treated as an 8-core node for scheduling, the Pods' binding cores could possibly be impacted.

Component Permission Description

Crane Scheduler Permission

Permission Description

The permission of this component is the minimal dependency required for the current feature to operate.

Permission Scenarios

Feature	Involved Object	Involved Operation

		Permission
It is required to keep track of the updates and changes to the node, as well as the utilization of the access node.	nodes	get/watch/list
Track the updates and changes of pods, and determine the scheduling priority of nodes based on the recent scheduling situation of pods within the cluster.	pods/namespaces	get/watch/list
It is required to update node utilization to node resources, thereby achieving the decoupling of scheduling and query logic.	nodes/status	patch
It is required to support multiple replicas to ensure component availability.	leases	create/get/update
It is required to track the updates and changes of the configmap, implementing the feature of scheduling specified pods to native nodes.	configmap	get/list/watch

Permission Definition

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: crane-scheduler
rules:
- apiGroups:
  - ""
  resources:
  - pods
  - nodes
  - namespaces
  verbs:
  - list
  - watch
  - get
- apiGroups:
  - ""
  resources:
  - nodes/status
  verbs:
  - patch
```

```
- apiGroups:
  - ""
  resources:
  - configmaps
  verbs:
  - get
  - list
  - watch
- apiGroups:
  - extensions
  - apps
  resources:
  - deployments/scale
  verbs:
  - get
  - update
- apiGroups:
  - coordination.k8s.io
  resources:
  - leases
  verbs:
  - create
  - get
  - update
- apiGroups:
  - "scheduling.crane.io"
  resources:
  - clusternoderesourcepolicies
  - noderesourcepolicies
  - craneschedulerconfigurations
  verbs:
  - get
  - list
  - watch
  - update
  - create
  - patch
```

Crane Descheduler Permission

Permission Description

The permission of this component is the minimal dependency required for the current feature to operate.

Permission Scenarios

Feature	Involved Object	Involved Operation Permission
It is required to keep track of the updates and changes to the node, as well as the utilization of the access node.	nodes	get/watch/list
Track the updates and changes of the pods, determining the pods to be evicted first based on the information of the pods within the clusters.	pods	get/watch/list
Drain the pod.	pods/eviction	create
It is required to determine whether the number of ready workloads where the pod resides constitutes half or more of the total requirements to decide whether to drain the pod.	replicasets/deployments/statefulsets/statefulsetpluses/job	get
Report events when draining Pods.	create	events

Permission Definition

```
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: crane-descheduler
  namespace: kube-system
rules:
  - apiGroups: [""]
    resources: ["nodes"]
    verbs: ["get", "watch", "list"]
  - apiGroups: [""]
    resources: ["pods"]
    verbs: ["get", "watch", "list"]
  - apiGroups: [""]
    resources: ["nodes/status"]
    verbs: ["patch"]
```

```
  - apiGroups: [""]
    resources: ["pods/eviction"]
    verbs: ["create"]
  - apiGroups: ["*"]
    resources: ["replicasets"]
    verbs: ["get"]
  - apiGroups: ["*"]
    resources: ["deployments"]
    verbs: ["get"]
  - apiGroups: ["apps"]
    resources: ["statefulsets"]
    verbs: ["get"]
  - apiGroups: ["platform.stke"]
    resources: ["statefulsetpluses"]
    verbs: ["get"]
  - apiGroups: [""]
    resources: ["events"]
    verbs: ["create"]
  - apiGroups: ["*"]
    resources: ["jobs"]
    verbs: ["get"]
  - apiGroups: [ "coordination.k8s.io" ]
    resources: [ "leases"
```

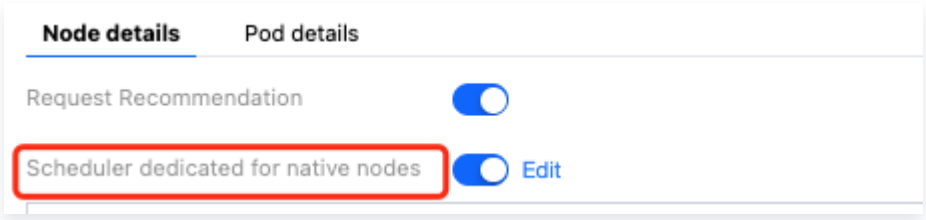
User Guide

Last updated：2024-12-24 15:26:27

This document describes how to use the native node-specific scheduler.

Installing the Native Node-specific Scheduler

1. Log in to the Tencent Kubernetes Engine console.
2. On the left side, select **TKE Insight > Node Map**.
3. On the **Node Map** page, hover over the mouse on a Node at the bottom of the page and click **Details**.
4. In the top right corner of the detail page for this node, enable the "Native Node-specific Scheduler" function.

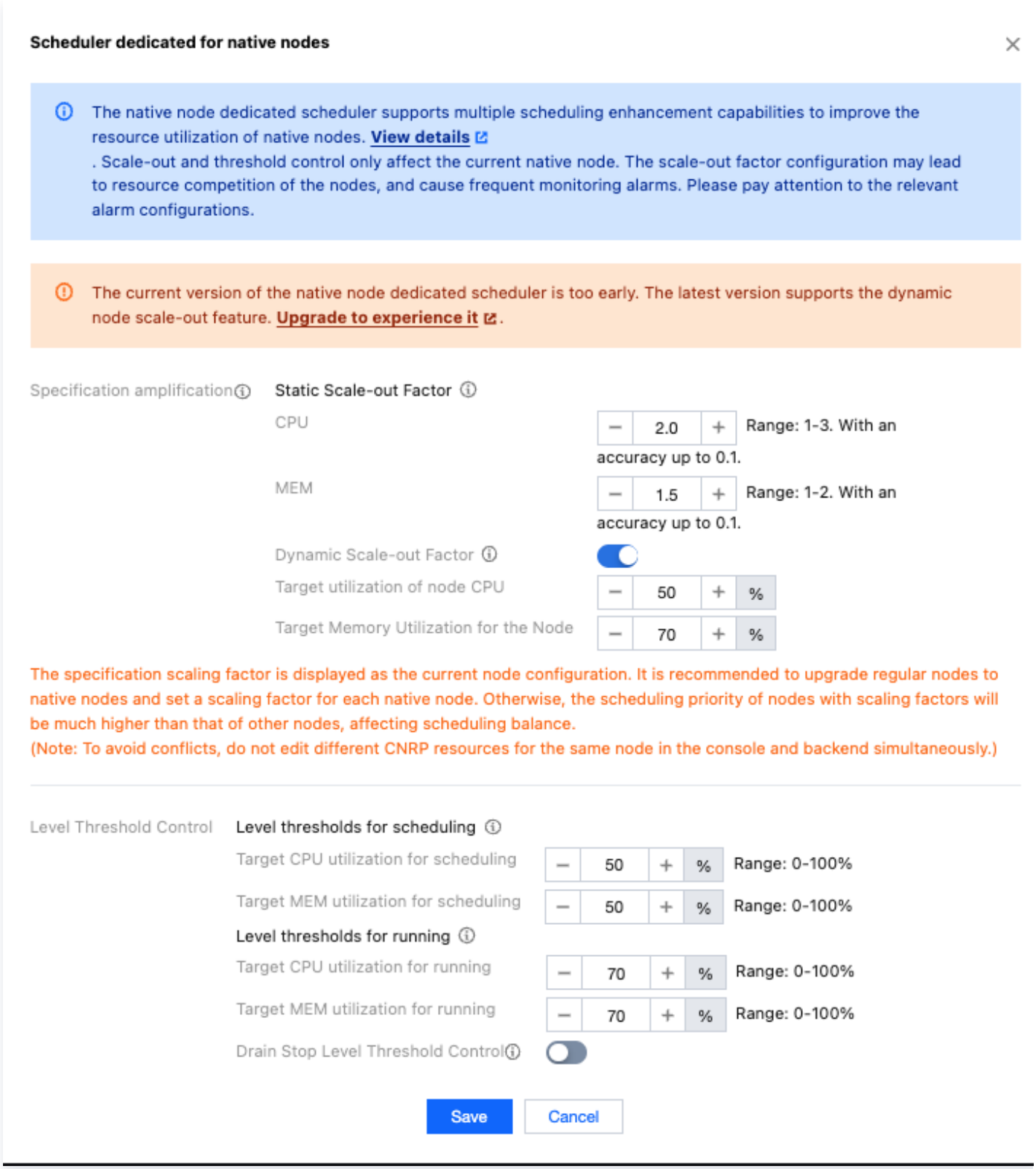


> **Note:**
> This capability is a global one, meaning it only needs to be enabled once for a cluster. The functions in other node detail pages will be enabled/disabled synchronously. This component is only effective for the Native Node. If the native node resources are insufficient, it can easily lead to Pod Pending.

Using the Native Node-specific Scheduler

Setting the Node Amplification Factor and the Threshold

1. Log in to the Tencent Kubernetes Engine console.
2. On the left sidebar, select **TKE Insight > Node Map**.
3. On the **Node Map** page, hover over the mouse on a native node that needs amplification at the bottom of the page, and click **Details**.
4. In the top right corner of the detail page for this node, click **Edit** on the right of the native node-specific scheduler.
5. On the editing page of the Native Node-specific Scheduler, set the node amplification factor and the threshold, as shown below:



Field Name	Description
Specification amplification	Virtually amplifies the native node specifications to schedule more Pods, allowing the current node packing rate to exceed 100%.

	Note: The current node specification amplification only virtually amplifies the current Native Node specifications. This feature carries certain risks: if too many Pods are scheduled on a node and the total actual resource usage of Pods exceeds the node specifications, it will trigger Pod evicting and rescheduling.
Static amplification factor – CPU	Supports the entering of numbers between 1 and 3, keeping one decimal place.
Static amplification factor – Memory	Supports the entering of numbers between 1 and 2, keeping one decimal place.
Scheduling threshold control	Sets the target resource utilization for the current native node to ensure stability. When scheduling Pods, nodes with a utilization above this threshold will not be selected. The scheduling threshold is also a default for: 1. The evicting stop threshold: When the node utilization reaches the runtime threshold and evicting occurs, it will be evicted to the target threshold. For example: If the runtime threshold is 80 and the scheduling threshold is 60, start evicting after the node utilization reaches 80. Pods are evicted in sequence based on the utilization of Pods that can be evicted until the threshold is below 60. 2. Low-load node threshold: When the node utilization reaches the runtime threshold and evicting occurs, it needs to be checked whether the utilization of **At Least Three Native Nodes** is lower than the **Scheduling Threshold** of these nodes. For example: If there are 5 native nodes, each with a runtime threshold of 80 and a scheduling threshold of 60, then when the node utilization reaches 80, it starts to check for evicting. To find the low-load native nodes that can accommodate Pods, at least three native nodes shall have the utilization lower than 60. **Note:** If different native nodes have different scheduling thresholds, it is necessary to check the utilization and scheduling threshold of the specific native node.

Target CPU utilization during scheduling	Supports the entering of integers between 0 and 100.
Target memory utilization during scheduling	Supports the entering of integers between 0 and 100.
Runtime threshold control	Sets the target resource utilization for the current native node to ensure stability. When the node is running, nodes with the utilization above this threshold may trigger evicting, provided that the tag indicating the evicting status is specified on the designated workload. **Note:** 1. Business Pods are not evicted by default. To avoid evicting critical Pods, this feature does not evict Pods by default. For Pods that can be evicted, users need to indicate on the workloads of such Pods, then the component will perform evicting when the threshold is triggered. For example, set the evictable annotation of: `descheduler.alpha.kubernetes.io/evictable: 'true'` for the objects of statefulset and deployment. 2. The eviction can occur only when there are enough low-load nodes. To ensure that the Pods have a node to accommodate after the eviction, the scheduler requires the utilization of **At Least Three Native Nodes** to be below the **Scheduling Threshold** of these nodes. Therefore, if there are many native nodes in the cluster, but fewer than three native nodes have the threshold enabled, or fewer than three native nodes are below the **Scheduling threshold**, the eviction cannot be performed.
Runtime target CPU utilization	Supports the entering of integers between 0 and 100. Note: The runtime target CPU utilization shall be greater than the scheduling target CPU utilization.
Runtime target memory utilization	Supports the entering of integers between 0 and 100. Note: The runtime target memory utilization shall be greater than the scheduling target memory utilization.
Eviction-stop threshold control	Starting from v1.4.0 and later for the native node scheduler, the eviction-stop threshold can be visually configured via the console. This means that, when a

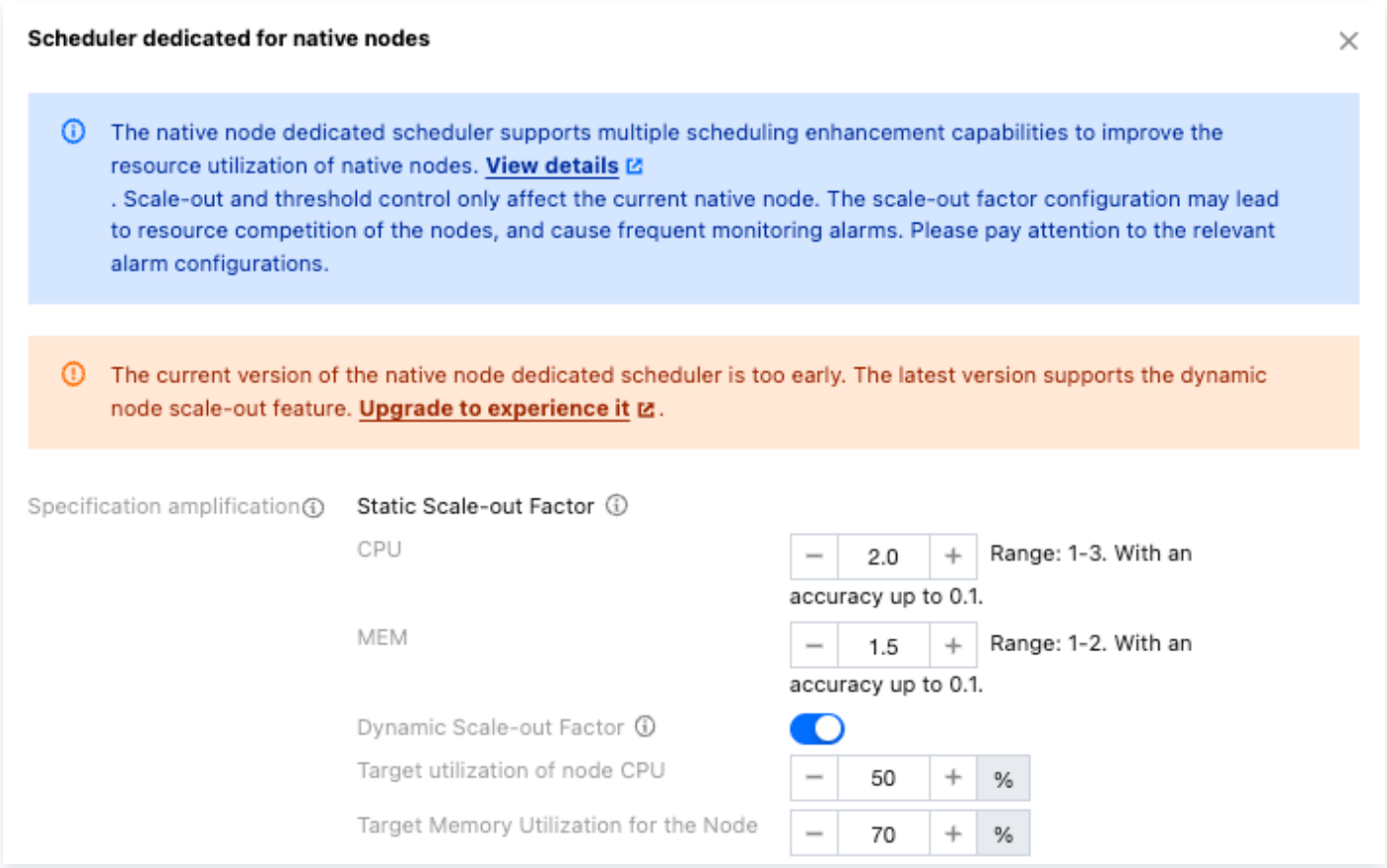
	node is under a high load, it will continuously evict the Pods until the eviction–stop threshold is reached.
Eviction–stop target CPU utilization	Supports the entering of integers between 0 and 100. Note: The runtime target CPU utilization shall be greater than the scheduling target CPU utilization.
Eviction–stop target memory utilization	Supports the entering of integers between 0 and 100. Note: The runtime target CPU utilization shall be greater than the scheduling target CPU utilization.

Description of Node Dynamic Amplification Capability

To address stability issues caused by overly aggressive configuration factor configuration by customers and to prevent mindless node amplification, the smaller value of the target node utilization/current node utilization and the static amplification rate will be selected as the dynamic amplification factor. The native node scheduler of v1.4.0 and later supports the dynamic amplification capability, which can be enabled as needed.

Configuration of Node Dynamic Amplification

On the editing page of the native node–specific scheduler, enable dynamic amplification, and set the target CPU and memory utilization for the node.



Calculation Methods for Node Dynamic Amplification

1. Complete the configuration for the node static amplification factor maxRatio and the dynamic amplification factor target utilization targetUsage.
2. Obtain the node peak utilization maxUsage for the recent 7 days.
3. The dynamic amplification factor is the smaller value of maxUsage/targetUsage and maxRatio.
4. To ensure cluster stability, the dynamic amplification factor shall be greater than the actual packing rate of the node currently.

Enabling Load-Awareness Scheduling

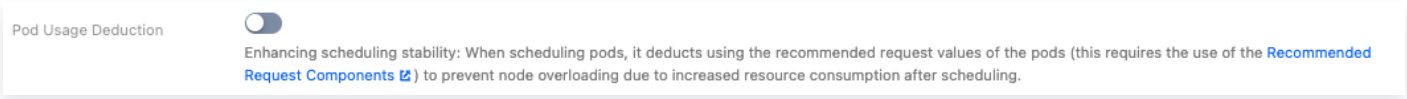
To address the issue where non-CNRP selected native nodes do not have any pod scheduling, the native node scheduler of v1.5.0 and later supports the enabling of load-awareness scheduling. Once enabled, all native nodes in the cluster will be scheduled based on the load awareness. Even if a node is not managed by CNRP, it can still be scheduled based on the actual load. This feature is disabled by default and can be manually enabled in the console.

> **Note:**
> Enabling the load-awareness scheduling does not affect the inventory. This feature is only available for the public cloud clusters.

Enabling Pod Usage Deduction Scheduling

To ensure the stability of cluster scheduling and to avoid the scheduling of high-request Pods for nodes close to the scheduling threshold, the native node scheduler of version 1.3 and later supports the "Pod usage deduction" scheduling. Once enabled, the resource usage of Pods will be deducted during the pre-selection and preferential selection stages of the scheduler, calculating whether it meets the pre-selection rules, and participating in the preferential scoring.
The method for calculating the Pod resource usage is as follows:

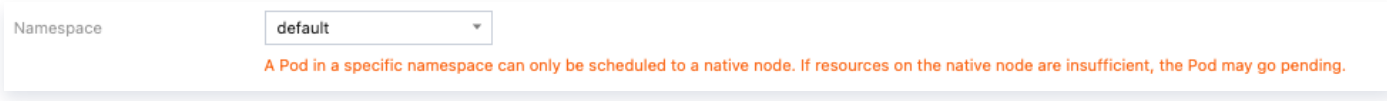
1. If the cluster has Request Intelligent Recommendation enabled and the Pod already has a recommended value, the load will be directly deducted based on this recommended value.
2. If there is no recommended value for the workload of the Pod, the deduction will be performed based on the request value set for the Pod.
3. If the Pod itself does not have a recommended value configured, the deduction will be performed based on CPU:100m and Mem:200Mi by default.



Setting the Scheduling of Pods in the Specified Namespace to Native Nodes

In addition to setting the amplification factor and the threshold control parameters on the current native nodes, the native node-specific scheduler also supports the scheduling of Pods from a specified namespace to native nodes. This helps users automate Pod migration and fully utilize the advantages of native nodes. The steps are as follows:

1. Log in to the Tencent Kubernetes Engine console, select **Cluster** in the left sidebar.
2. On the **Cluster Management** page, click the target cluster ID to enter the page of cluster details.
3. Select **Component Management**. On the component list page, select "craneScheduler (native node-specific scheduler)", and click **Update Configuration**.
4. Select the namespace. Pods in the specified namespace can only be scheduled to the native nodes. If the Native Node has insufficient resources, it will cause Pod Pending.



Description of Node Specification Amplification Capability

If the CPU and memory amplification factors of the nodes are specified, you can view the annotation of: `expansion.scheduling.crane.io/cpu` and `expansion.scheduling.crane.io/memory` related to the amplification factors on the native nodes. An example is as follows:

```
kubectl describe node 10.8.22.108

...
Annotations:        expansion.scheduling.crane.io/cpu: 1.5   # CPU
amplification factor
                    expansion.scheduling.crane.io/memory: 1.2    # Memory
amplification factor
...
Allocatable:
  cpu:                1930m # Original schedulable resource amount of
the node
  ephemeral-storage:  47498714648
  hugepages-1Gi:      0
  hugepages-2Mi:      0
  memory:             1333120Ki
  pods:               253
...
Allocated resources:
  (Total limits may be over 100 percent, i.e., overcommitted.)
```

```
  Resource           Requests         Limits
  --------           --------         ------
  cpu                960m (49%)       8100m (419%)   # Occupancy of
Request and Limit for this node
  memory             644465536 (47%)  7791050368 (570%)
  ephemeral-storage  0 (0%)           0 (0%)
  hugepages-1Gi      0 (0%)           0 (0%)
  hugepages-2Mi      0 (0%)           0 (0%)
...
```

Note:

- Original CPU schedulable amount of current node: 1930m
- Total CPU Request application volume of all Pods on the node: 960m
- Under normal circumstances, this node can only schedule up to 1930m–960m=970m CPUs.
- But in reality, the CPU schedulable amount of this node has been virtually amplified to: 1930m*1.5=2895m, and the actual remaining CPU schedulable resource is: 2895–960=1935m.

At this time, create a workload with only one Pod, the CPU application volume is 1500m. If there is no node amplification capability, this node cannot be scheduled to.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  namespace: default
  name: test-scheduler
  labels:
    app: nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      nodeSelector: # Specify node scheduling
        kubernetes.io/hostname: 10.8.20.108 # Specify the use of native
nodes in the example
```

```
      containers:
      - name: nginx
        image: nginx:1.14.2
        resources:
          requests:
            cpu: 1500m  # The application node is greater than the
schedulable amount before amplification but less than the schedulable
amount after amplification.
        ports:
        - containerPort: 80
```

The workload is successfully created:

```
kubectl get deployment
NAME                READY   UP-TO-DATE   AVAILABLE   AGE
test-scheduler      1/1     1            1           2m32s
```

Check the resource occupancy of the node again:

```
kubectl describe node 10.8.22.108

...
Allocated resources:
  (Total limits may be over 100 percent, i.e., overcommitted.)
  Resource           Requests         Limits
  --------           --------         ------
  cpu                2460m (127%)     8100m (419%)  # Occupancy of
Request and Limit for this node. It can be seen that, the total Request
exceeds the original schedulable amount. The node specification is
successfully amplified.
  memory             644465536 (47%)  7791050368 (570%)
  ephemeral-storage  0 (0%)           0 (0%)
  hugepages-1Gi      0 (0%)           0 (0%)
  hugepages-2Mi      0 (0%)           0 (0%)
```

Custom Setting of Node Amplification Factor and Threshold

> ⚠ Note:

> The custom setting of the node amplification factor and the threshold has high flexibility. You can select a node, a node pool, a batch of nodes or node pools, or even configure the same node specification amplification factor and threshold for the entire cluster.
> You can create multiple configuration files to apply to different nodes. If a node is declared by multiple configuration files and the declared parameters declared in different files are different, only the latest created configuration file will take effect.

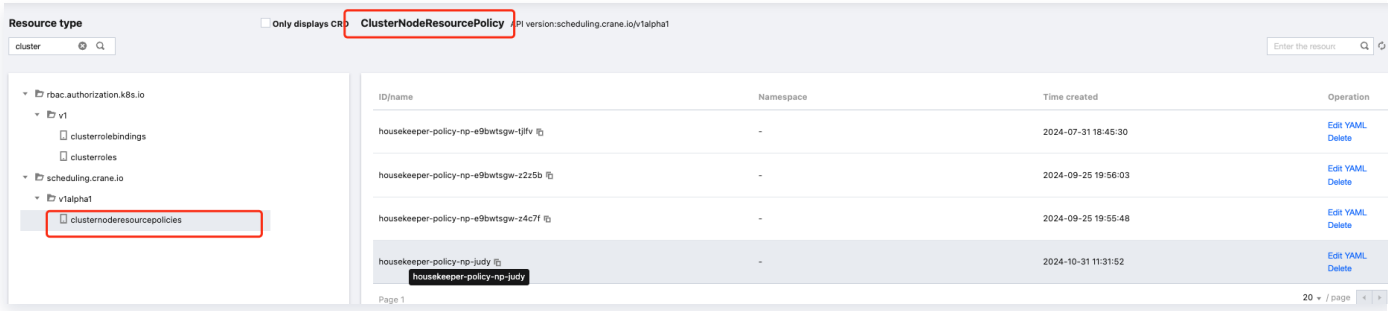
In addition to the console operation, the parameters of the native node-specific scheduler also support the configuration through YAML. Each configuration file is saved in a resource object named clusternoderesourcepolicies (CNRP). You can directly create this type of resource object:

```
apiVersion: scheduling.crane.io/v1alpha1
kind: ClusterNodeResourcePolicy
metadata:
  name: housekeeper-policy-np-88888888-55555 # name cannot be
duplicated.
spec:
  applyMode: annotation  # Default value. Other values are not supported
for now.
  nodeSelector:  # It is used to select a group of nodes with the same
configuration. For example, below is used to select a native node with
the ID of np-88888888-55555. You can also use a common tag for a batch
of nodes to achieve batch selection of nodes.
    matchLabels:
      cloud.tencent.com/node-instance-id: np-88888888-55555
  template:
    spec:
      # evictLoadThreshold is the runtime threshold control of native
nodes. It is used for setting the eviction resource utilization for the
current batch of native nodes to ensure stability. When the Pods run on
native nodes, the native nodes above this threshold may be evicted. To
avoid evicting critical Pods, this feature does not evict Pods by
default. For Pods that can be evicted, users need to explicitly judge
the workload of the Pod. For example, set the evictable annotation of:
descheduler.alpha.kubernetes.io/evictable: 'true' for the objects of
statefulset and deployment.
      # evictLoadThreshold shall be greater than targetLoadThreshold,
otherwise the node may be continuously scheduled with new Pods after
eviction to cause jitter.
```

```
      evictLoadThreshold:
        percents:
          cpu: 80
          memory: 80
      resourceExpansionStrategyType: static # default value. Other
values are not supported for now.
      staticResourceExpansion:
        ratios: # Node specification amplification factor. It is used
for setting the amplification factor for the current batch of native
nodes.
          cpu: "3" # The amplification factor of the node CPU. It is
recommended not to set it too high, otherwise it may cause stability
risks. The maximum value is limited to 3 in the console.
          memory: "2" # The amplification factor of the node memory. It
is recommended not to set it too high, otherwise it may cause stability
risks. The maximum value is limited to 2 in the console.
      # targetLoadThreshold is the scheduling threshold control of
native nodes. It is used for setting the amplification factor for the
current batch of native nodes to ensure stability. Native nodes above
this threshold will not be selected during Pod scheduling.
      # targetLoadThreshold shall be less than evictLoadThreshold,
otherwise the node may be continuously scheduled with new Pods after
eviction to cause jitter.
      targetLoadThreshold:
        percents:
          cpu: 70
          memory: 70
```

You can modify or create this type of resource object. As shown below, you can view the inventory of clusternoderesourcepolicies (CNRP) in the console.

1. Log in to the Tencent Kubernetes Engine console, and select Cluster in the left sidebar.
2. Click the cluster ID of the native node-specific scheduler to be viewed, and enter the resource object browser page. Search for clusternoderesourcepolicies (CNRP) in the top-left box, as shown below:



If you configure the scheduling parameters for native nodes via the console page, these files will be designated as: housekeeper-policy-np-88888888-55555, wherein np-88888888 is the node pool ID of this native node, and np-88888888-55555 is the node ID of this native node.

Setting Node Dynamic Amplification

The native node scheduler of v1.4.0 and later supports node dynamic amplification. The parameters can also be configured via the backend YAML. The corresponding ClusterNodeResourcePolicy example is as follows:

```
apiVersion: scheduling.crane.io/v1alpha1
kind: ClusterNodeResourcePolicy
metadata:
  name: housekeeper-policy-np-88888888-55555 # name cannot be
duplicated.
spec:
  applyMode: annotation  # Default value. Other values are not supported
for now.
  nodeSelector:  # It is used to select a group of nodes with the same
configuration. For example, below is used to select a native node with
the ID of np-88888888-55555. You can also use a common tag for a batch
of nodes to achieve batch selection of nodes.
    matchLabels:
      cloud.tencent.com/node-instance-id: np-88888888-55555
  template:
    spec:
      resourceExpansionStrategyType: auto # Enable node dynamic
amplification
      autoResourceExpansion:
        crontab: 0 * * * 0-6
        decayLife: 168h
        # Setting Specific Parameters for Node Dynamic Amplification
        maxRatios:
```

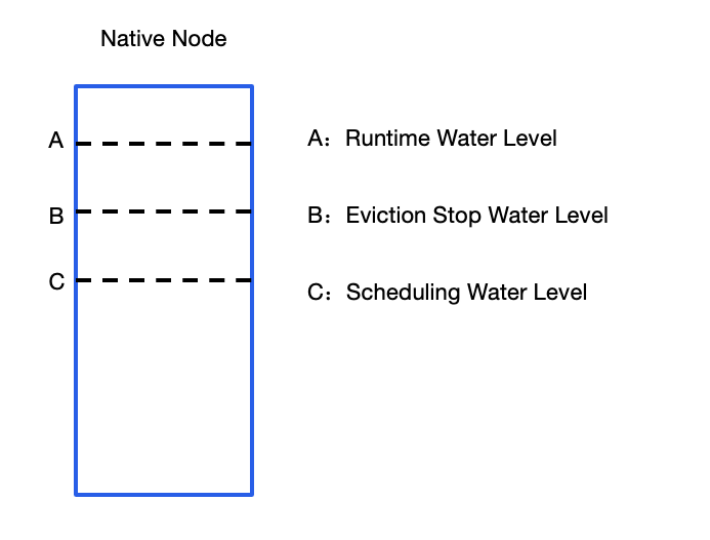
```
          cpu: "2"
          memory: "1.5"
        minRatios:
          cpu: "1.0"
          memory: "1.0"
        targetLoadThreshold:
          percents:
            cpu: 50
            memory: 70
      # evictLoadThreshold is the runtime threshold control of native
nodes. It is used for setting the eviction resource utilization for the
current batch of native nodes to ensure stability. When Pods run on
native nodes, native nodes above this threshold may be evicted. To avoid
evicting critical Pods, this feature does not evict Pods by default. For
Pods that can be evicted, users need to explicitly judge the workload of
the Pod. For example, set the evictable annotation of:
descheduler.alpha.kubernetes.io/evictable: 'true' for the objects of
statefulset and deployment.
      # evictLoadThreshold shall be greater than targetLoadThreshold,
otherwise the node may be continuously scheduled with new Pods after
eviction to cause jitter.
      evictLoadThreshold:
        percents:
          cpu: 80
          memory: 80
      # targetLoadThreshold is the scheduling threshold control of
native nodes. It is used for setting the current target resource
utilization of these native nodes to ensure stability. Native nodes
above this threshold will not be selected during Pod scheduling.
      # targetLoadThreshold shall be less than evictLoadThreshold,
otherwise the node may be continuously scheduled with new Pods after
eviction to cause jitter.
      targetLoadThreshold:
        percents:
          cpu: 70
          memory: 70
```

Custom Eviction–Stop Threshold

As mentioned above, there are two additional features of: Eviction-stop threshold and low-load node threshold for the current scheduling threshold. However, this design leads to feature binding issues, for example:

1. If users want to evict fewer Pods during business eviction, they need to raise the scheduling threshold, which will result in more Pods being scheduled on the node, increasing the probability of eviction. Additionally, since fewer Pods are evicted, the node may experience frequent scheduling and eviction, increasing node instability.
2. If users want to evict more Pods during business eviction, they need to reduce the scheduling threshold, which will result in fewer Pods being scheduled on the node, decreasing the probability of eviction. Since more Pods are evicted, the possible utilization of the node will change dramatically, and the node instability will also be increased. Additionally, since the scheduling threshold is lower, the utilization cannot be improved, leading to significant resource waste.

Therefore, in the new Crane Scheduler, TKE has introduced the third threshold of: the eviction-stop threshold, which has the two additional features of: the eviction-stop threshold and the low-load node threshold for the scheduling threshold.



Note:
This feature requires the version of the Crane Scheduler to be at least 1.1.10. For version details and upgrade information, refer to Documentation.

```
apiVersion: scheduling.crane.io/v1alpha1
kind: ClusterNodeResourcePolicy
metadata:
  name: housekeeper-policy-np-88888888-55555 # name cannot be
duplicated.
spec:
```

```
  applyMode: annotation  # Default value. Other values are not supported
for now.
  nodeSelector:  # It is used to select a group of nodes with the same
configuration. For example, below is used to select a native node with
the ID of np-88888888-55555. You can also use a common tag for a batch
of nodes to achieve batch selection of nodes.
    matchLabels:
      cloud.tencent.com/node-instance-id: np-88888888-55555
  template:
    spec:
      # evictLoadThreshold is the runtime threshold control of native
nodes. It is used for setting the eviction resource utilization for the
current batch of native nodes to ensure stability. When Pods run on
native nodes, native nodes above this threshold may be evicted. To avoid
evicting critical Pods, this feature does not evict Pods by default. For
Pods that can be evicted, users need to explicitly judge the workload of
the Pod. For example, set the evictable annotation of:
descheduler.alpha.kubernetes.io/evictable: 'true' for the objects of
statefulset and deployment.
      # evictLoadThreshold shall be greater than targetLoadThreshold,
otherwise the node may be continuously scheduled with new Pods after
eviction to cause jitter.
      evictLoadThreshold:
        percents:
          cpu: 80
          memory: 80
      ##################
      ## Eviction Stop Watermark ##
      # The value shall be lower than the runtime threshold of
evictLoadThreshold.
      evictTargetLoadThreshold:
        percents:
          cpu: 75
          memory: 75
      resourceExpansionStrategyType: static # default value. Other
values are not supported for now.
      staticResourceExpansion:
        ratios: # Node specification amplification factor. It is used
for setting the amplification factor for the current batch of native
nodes.
```

```
          cpu: "3" # The amplification factor of the node CPU. It is
recommended not to set it too high, otherwise it may cause stability
risks. The maximum value is limited to 3 in the console.
          memory: "2" # The amplification factor of the node memory. It
is recommended not to set it too high, otherwise it may cause stability
risks. The maximum value is limited to 2 in the console.
      # targetLoadThreshold is the scheduling threshold control of
native nodes. It is used for setting the current target resource
utilization of these native nodes to ensure stability. Native nodes
above this threshold will not be selected during Pod scheduling.
      # targetLoadThreshold shall be less than evictLoadThreshold,
otherwise the node may be continuously scheduled with new Pods after
eviction to cause jitter.
      targetLoadThreshold:
        percents:
          cpu: 70
          memory: 70
```

Scheduling Policy Configuration

Last updated：2024-12-24 15:27:03

Overview

The scheduling policy configuration is a feature of the native node-specific scheduler (Crane-scheduler), designed to help users flexibly adjust Pod scheduling policies based on business needs and cluster resource status. By configuring scheduling policies, users can optimize cluster resource utilization, enhance application performance, and improve availability.

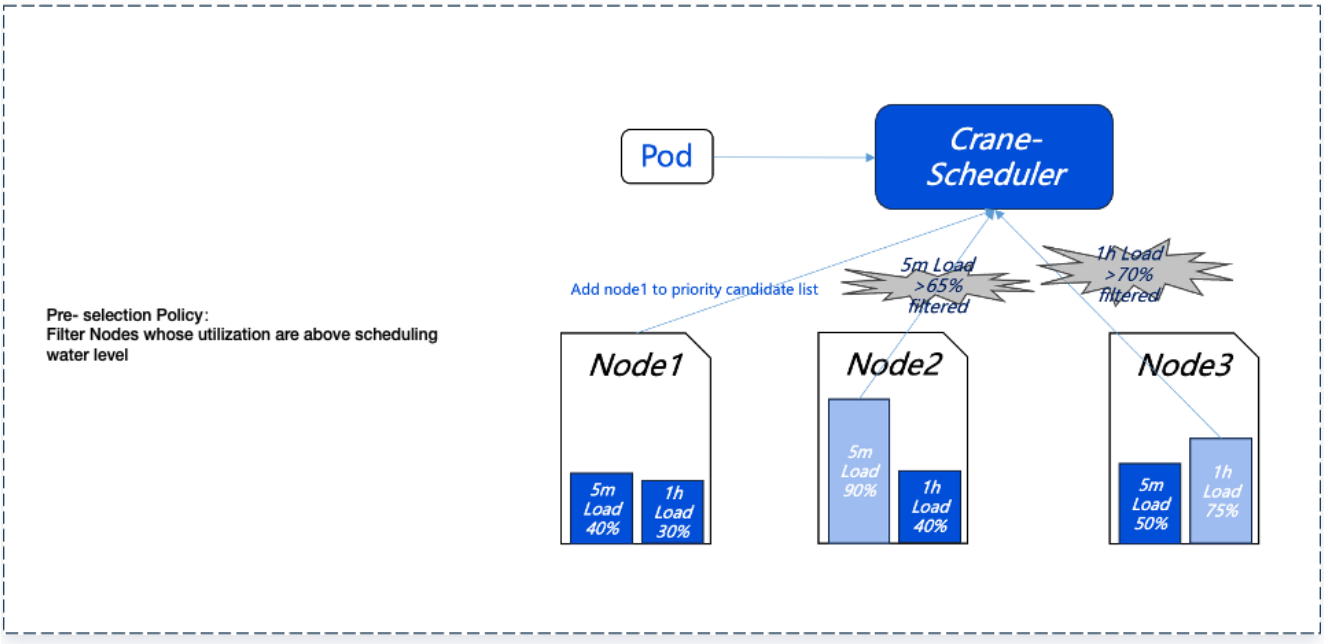
Version Limits

Only v1.4.1 and above versions of Crane-scheduler are supported.

Scheduling Policy Description

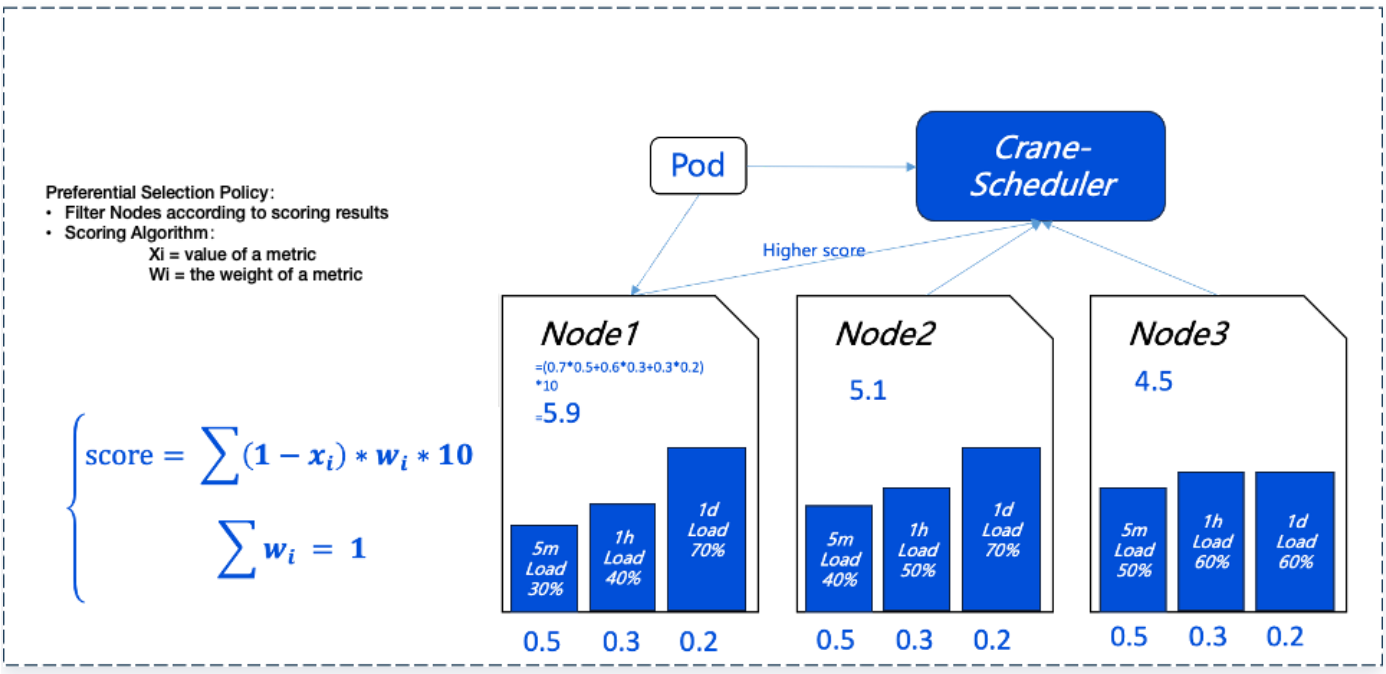
Pre-selection Policy

To avoid scheduling Pods to high-load nodes (Nodes), we first use a pre-selection policy to filter out high-load nodes. The pre-selection policy can dynamically configure filtering thresholds and proportions based on the actual load of the Nodes. For specific configuration methods, please refer to the subsequent parameter settings.
As an example, Node2's load over the past 5 minutes and Node3's load over the past hour have both exceeded the preset thresholds, so they will not enter the next phase of the preferential selection process.



Preferential Selection Policy

The preferential selection policy aims to select the best nodes from those that meet the pre-selection criteria for Pod scheduling. To achieve balanced distribution of load among nodes in the cluster, Crane-scheduler scores each node based on its real-time load data, with lower-load nodes receiving higher scores. The scoring strategy and its weight coefficients can be dynamically adjusted based on actual needs. For detailed configuration methods, please refer to the subsequent parameter configuration description.
In the diagram below; we can see that Node1, received the highest score due to its low load, so it will be prioritized by the system to run Pods.

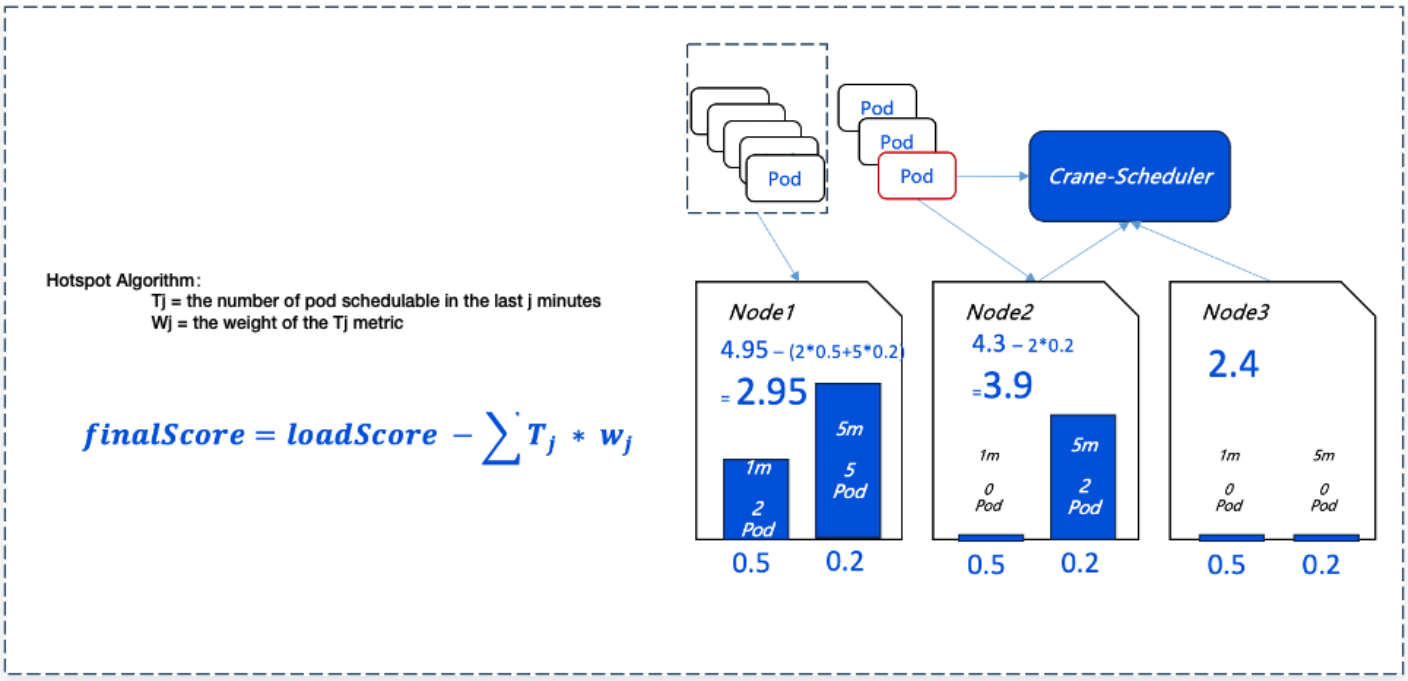


Scheduling Hotspots

To prevent low-load nodes from becoming new load hotspots due to the continuous scheduling of large numbers of Pods, Crane-scheduler introduces a hotspot scheduling policy. This policy adjusts node scores in the preferential selection phase based on the number of Pods scheduled in the past period.

- **Node Score Calculation Formula:** Node Score = Preferred Selection Policy Scoring – Hotspot Value
- **The Current Policy Details Are as Follows:**
  - If more than two Pods were scheduled to the node in the past minute, the node's score for preferential selection is decreased by 1.
  - If more than five Pods were scheduled to the node in the past 5 minutes, the node's score for preferential selection is also decreased by 1.

By implementing this policy, Crane-scheduler ensures that scheduling decisions are more comprehensive, taking into account both the immediate load on nodes and their recent scheduling history, thereby achieving a more balanced cluster load distribution.



Parameter Description

Pre-selection Parameters

Note:

- Default values for pre-selection parameters. If you have no special requirements, you can directly use them.
- The threshold for the default parameters of pre-selection indicators is the water level control parameter setting of the native node dedicated scheduler. For details, please refer to the Usage Instructions.

Default Parameters for Pre-selection Indicators	Description
Average CPU utilization threshold in 5 minutes	If the average CPU utilization of the node in the past 5 minutes exceeds the configured threshold, no Pods will be scheduled to the node.
Maximum CPU utilization threshold in an hour	If the maximum CPU utilization of the node in the past hour exceeds the configured threshold, no Pods will be scheduled to the node.
Average memory utilization threshold in 5 minutes	If the average memory utilization of the node in the past 5 minutes exceeds the configured threshold, no Pods will be scheduled to the node.

Maximum memory utilization threshold in an hour	If the maximum memory utilization of the node in the past hour exceeds the configured threshold, no Pods will be scheduled to the node.

Preferential Selection Parameters

> **Note:**
> Default values for preferential selection parameters. If you have no special requirements, you can directly use them.

Preferential Selection Indicators	Default Parameters	Description
Average CPU utilization weight in 5 minutes	0.2	The greater the weight, the bigger impact the average CPU utilization in the past 5 minutes has on the node score.
Maximum CPU utilization weight in an hour	0.3	The greater the weight, the bigger impact the maximum CPU utilization in the past hour has on the node score.
Maximum CPU utilization weight in a day	0.5	The greater the weight, the bigger impact the maximum CPU utilization in the past day has on the node score.
Average memory utilization weight in 5 minutes	0.2	The greater the weight, the bigger impact the average memory utilization in the past 5 minutes has on the node score.
Maximum memory utilization weight in an hour	0.3	The greater the weight, the bigger impact the maximum memory utilization in the past hour has on the node score.
Maximum memory utilization weight in a day	0.5	The greater the weight, the bigger impact the maximum memory utilization in the past day has on the node score.

Hotspot Parameters

> **Note:**
> Default values for hotspot parameters. If you have no special requirements, you can directly use them.

Hotspot Indicators	Default Parameters	Description
Number of Pods scheduled in the last 5 minutes	10.0	The greater the weight, the bigger impact the number of Pods scheduled in the last 5 minutes has on the node score.
Number of Pods scheduled in the last 1 minute	20.0	The greater the weight, the bigger impact the number of Pods scheduled in the last 1 minute has on the node score.

Operation Steps

Starting to Use

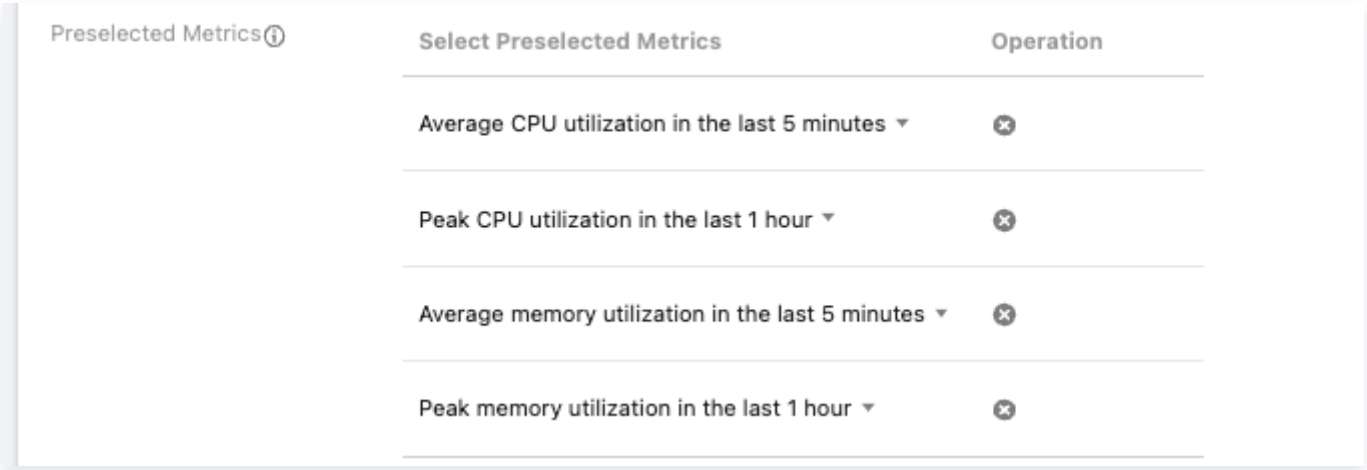
1. Log in to the Tencent Kubernetes Engine Console, and choose **Cluster** from the left navigation bar.
2. In the cluster list, click on the desired cluster ID to access its detailed page.
3. Select **Add-On Management** from the menu on the left, click **Update Configuration** in the CranesScheduler operation column to enter the update component configuration page.

Configuration Scheduling Policy

> **Note:**
> 1. When configuring the scheduling policy, please ensure you fully understand the business needs and cluster resource status to avoid unnecessary resource waste or performance degradation.
> 2. It is recommended to validate the effect in the test environment before applying the scheduling policy to the production environment.

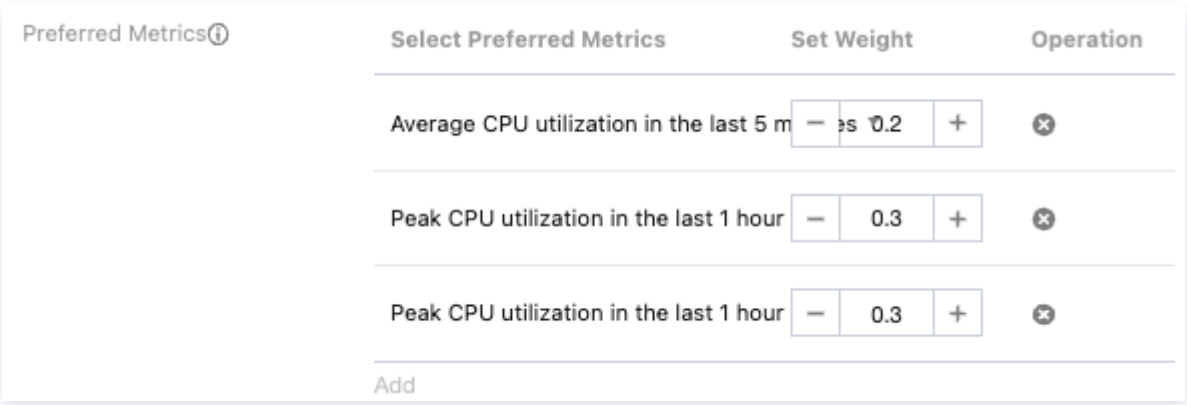
Pre-selection Indicator Configuration

- **Role of Pre-selection Indicators:** Used for preliminary node filtering. A node can only pass pre-selection if the corresponding indicator is below the set scheduling threshold.
- **Metrics Quantity Limit:** Up to four pre-selection indicators can be configured, and indicators must not be duplicated. You can also choose not to configure pre-selection indicators.



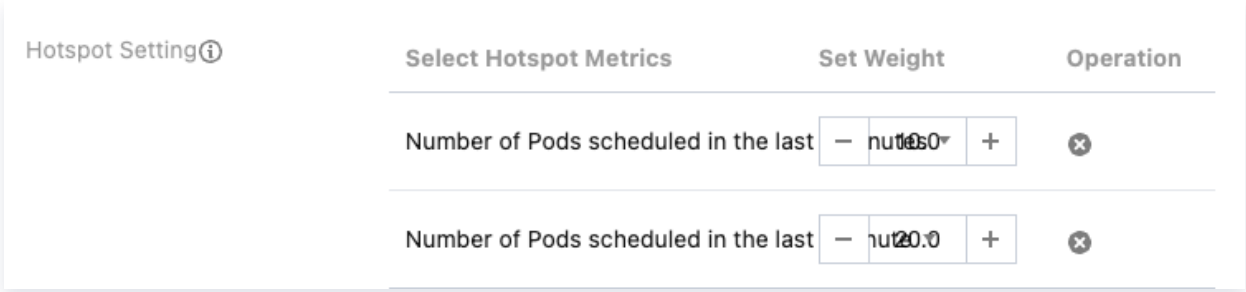
Preferred Metrics Configuration

- **Role of Preferred Metrics:** Used to further filter nodes that have passed pre-selection, displaying default weight parameters and supporting self-defined weight settings.
- **Scheduling Principles:** Pods will be scheduled to nodes with lower resource utilization first.
- **Indicator Quantity Limit:** Up to six preferential selection indicators can be configured, and indicators must not be duplicated. You can also choose not to configure preferential selection indicators.



Hotspot Settings

- **Role of Hotspot Indicators:** Considers the recent number of Pods scheduled to the node, to provide a more comprehensive assessment of the node's load status.
- **Recommended Weight Setting:** The weight for the number of Pods scheduled in the last 5 minutes is recommended to be set to 10, and the weight for the number of Pods scheduled in the last 1 minute is recommended to be set to 20.
- **Indicator Quantity Limit:** Up to two hotspot indicators can be configured, and indicators must not be duplicated. You can also choose not to configure hotspot indicators.



Fine Scheduling
QoSAgent

Last updated：2024-02-05 16:28:54

QoS Agent is an extended component enhanced by Tencent Cloud based on quality of service, offering an array of capabilities. It ensures stability while increasing the utilization rate of cluster resources.

> **Note:**
> QoS capabilities are only supported on native nodes. If your nodes are not native, or your workload does not reside on native nodes, these capabilities will not be effective.

Kubernetes objects deployed in a cluster

Kubernetes Object Name	Type	Default Resource Occupation	Associated Namespaces
avoidanceactions.ensurance.crane.io	CustomResourceDefinition	–	–
nodeqoss.ensurance.crane.io	CustomResourceDefinition	–	–
podqoss.ensurance.crane.io	CustomResourceDefinition	–	–
timeseriespredictions.prediction.crane.io	CustomResourceDefinition	–	–
kube-system	Namespace	–	–
all-be-pods	PodQOS	–	kube-system
qos-agent	ClusterRole	–	–
qos-agent	ClusterRoleBinding	–	–
crane-agent	Service	–	kube-system
qos-agent	ServiceAccount	–	kube-system
qos-agent	Daemonset	–	kube-system

Feature Overview

Feature	Description
Priority of CPU Usage	The feature of setting CPU usage priority ensures a sufficient supply of resources for high-priority tasks during resource competition, thereby suppressing low-priority tasks.
CPU Burst	CPU Burst permits temporary provision of resources beyond the limit for latency-sensitive applications, ensuring their stability.
CPU Hyperthreading Isolation	Preventing L2 Cache of high-priority container threads from being affected by low-priority threads running on the same CPU physical core.
Memory QoS Enhancement	A comprehensive enhancement of memory performance, along with the flexible limitations on the memory usage of the container.
Network QoS Enhancement	A comprehensive enhancement of network performance, along with flexible limitations on the network usage of the container.
Disk IO QoS Enhancement	A comprehensive enhancement of disk performance, along with flexible limitations on the disk usage of the container.

QoS Agent Permission

> **Note:**
> The **Permission Scenarios** section only lists the permissions related to the core features of the components, for a complete permission list, please refer to the **Permission Definition**.

Permission Description

The permission of this component is the minimal dependency required for the current feature to operate.

Permission Scenarios

Feature	Involved Object	Involved Operation Permission
Reading podqos, nodeqos, time series, and other configurations	podqoss / nodeqoss / avoidanceactions	get/list/watch/update
Viewing the pod information of the current node	pod	get/list/watch
Enabling isolation capability based on Podqos/	pod status	update/patch

Modifying node resources to increase offline resources		
Adding a taint to the node	node	get/list/watch/update
Sending events based on the status of isolation and resource interference	event	All Permissions

Permission Definition

```
rules:
 - apiGroups:
     - ""
   resources:
     - pods
   verbs:
     - get
     - list
     - watch
 - apiGroups:
     - ""
   resources:
     - pods/status
   verbs:
     - update
     - patch
 - apiGroups:
     - ""
   resources:
     - nodes
   verbs:
     - get
     - list
     - watch
     - update
 - apiGroups:
     - ""
   resources:
     - nodes/status
     - nodes/finalizers
   verbs:
```

```
    - update
    - patch
  - apiGroups:
    - ""
    resources:
    - pods/eviction
    verbs:
    - create
  - apiGroups:
    - ""
    resources:
    - configmaps
    verbs:
    - get
    - list
    - watch
  - apiGroups:
    - ""
    resources:
    - events
    verbs:
    - "*"
  - apiGroups:
    - "ensurance.crane.io"
    resources:
    - podqoss
    - nodeqoss
    - avoidanceactions
    verbs:
    - get
    - list
    - watch
    - update
  - apiGroups:
    - "prediction.crane.io"
    resources:
    - timeseriespredictions
    - timeseriespredictions/finalizers
    verbs:
    - get
```

```
      - list
      - watch
      - create
      - update
      - patch
  - apiGroups:
      - "topology.crane.io"
    resources:
      - "noderesourcetopologies"
    verbs:
      - get
      - list
      - watch
      - create
      - update
      - patch
```

Deployment Methods

1. Log into the Tencent Kubernetes Engine Console, and choose **Cluster** from the left navigation bar.
2. In the Cluster list, click the desired Cluster ID to access its detailed page.
3. Select **Add-on management** from the left-side menu, and click **Create** within the Component Management page.
4. On the **Create Add-on management** page, tick the box for **QoS Agent**.
5. Click **Complete** to install the add-on.

> ⚠ **Please Note:**
> With the completion of the deployment, you need to manually select the corresponding driver due to potential differences in cgroup driver of the cluster. The instructions are as follows:
>
> 1. Within the **Add-on** in your cluster, locate the successfully deployed QoS Agent, and click **Update configuration** on the right.
> 2. On the add-on configuration page of QoS Agent, select the dropdown box to the right of the cgroupDrive option, and choose cgroupDrive that matches your cluster.
> 3. Click **Complete**.

FAQs

How to confirm the cgroupDrive of a cluster?

The cgroupDrive of a cluster can only be either cgroupfs or systemd. The confirmation method is as follows:
Initially, the operation of peekcluster can be viewed in the "basic information" page of the cluster, specifically in the "operating add-on", by determining whether the current cluster serves as a docker or containerd.
If the operating cluster is docker, on any node in the cluster, execute `docker info` and view the field content of `Cgroup Driver` .
If the operating cluster is containerd, in the file of /etc/containerd/config.toml on any node in the cluster, the presence of the field: `SystemdCgroup = true` signifies a systemd, otherwise, it is a cgroup.

How to select the operating business or node?

Choosing a specific resource object via `label` or `scope` is supported.

> **Note:**
> When both of the following selectors exist concurrently, the operation used is an "and", i.e. all conditions must be met.

labelSelector

The labelSelector filters resources by associating them with the resource labels of the object. The usual method of usage is to attach a specific tag to the designated workloads on the business end. This Tag is then given to the operation team. When creating a PodQOS, the operation team associates this tag through the labelSelector field, effectively granting different QoS capabilities to different businesses.

scopeSelector

The scopeSelector is composed of multiple MatchExpressions. The relationship between these MatchExpressions is an "and". There are three fields in MatchExpressions, namely ScopeName, Operator, and Values corresponding to ScopeName;
The ScopeName includes three types: QOSClass, Priority, and Namespace;

- QOSClass refers to a desired Workload that is associated with a specific QOSClass. The Values can be one or more among Guaranteed, Burstable, and BestEffort;
- Priority refers to a desired Workload that is associated with a specific Priority. The Values can be specific priority values, such as ["1000", "2000-3000"], supporting a range of priorities;
- Namespace refers to a desired Workload that is associated with a specific Namespace. The Values can be one or more.

Operator includes two types, specifically In and NotIn. If left it blank, the default type is In.
As illustrated below, it denotes that the BestEffortPod meets a condition of app-type=offline, with a CPU priority of 7:

```
apiVersion: ensurance.crane.io/v1alpha1
```

```
kind: PodQOS
metadata:
  name: offline-task
spec:
  allowedActions:
  - eviction
  resourceQOS:
    cpuQOS:
      cpuPriority: 7
  scopeSelector:
    matchExpressions:
    - operator: In
      scopeName: QOSClass
      values:
      - BestEffort
  labelSelector:
    matchLabels:
      app-type: offline
```

CPU Usage Priority

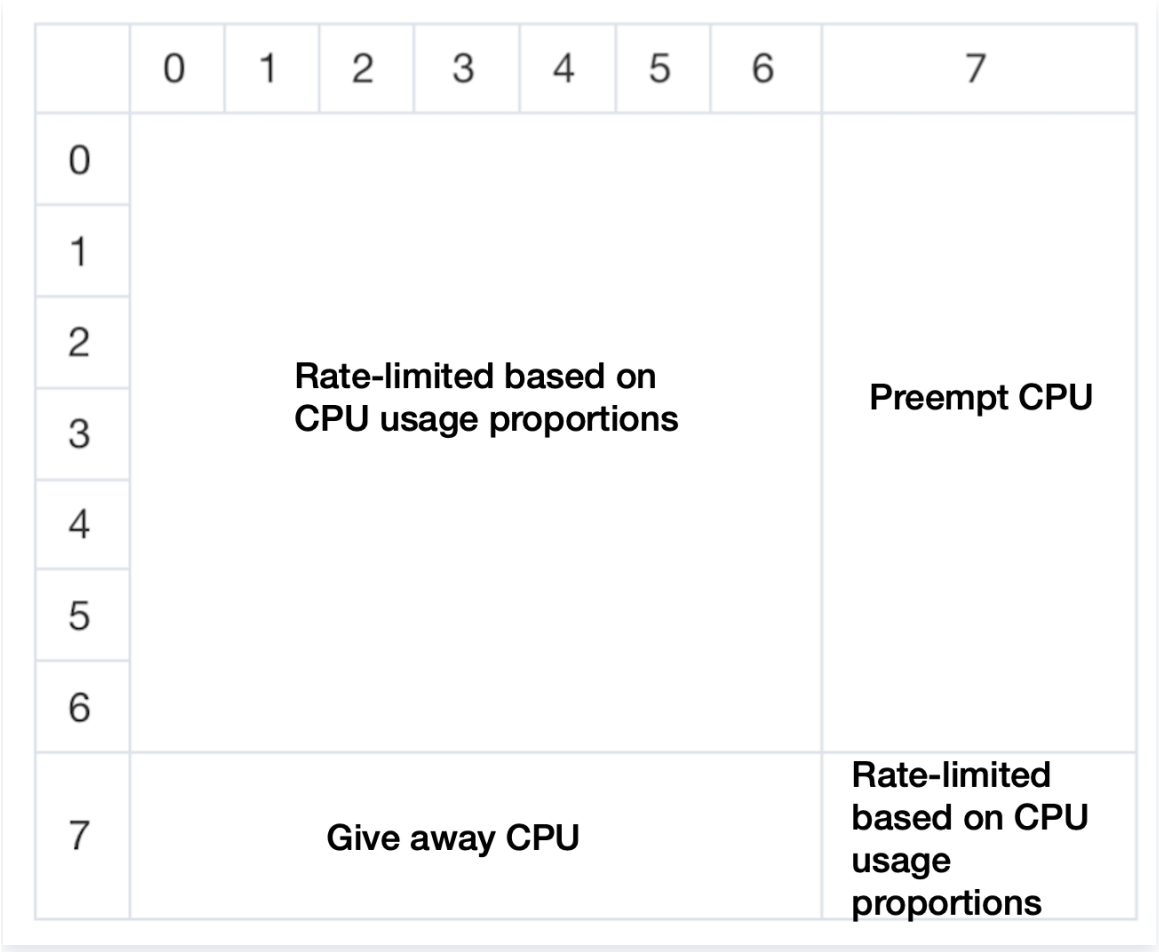
Last updated：2024-12-24 15:38:25

Overview

During the deployment of services, it's common to deploy a type of business process onto a specific server. The CPU utilization of the server depends on the pressure of the business process. Generally, the overall CPU utilization of a server tends to be relatively low. To improve the CPU utilization of the server, some tasks with lower priorities can be deployed on the server in a hybrid manner. This approach can significantly enhance the overall CPU utilization and reduce hardware costs. Here, the tasks with lower priorities deployed on the machine in a hybrid manner are referred to as "offline tasks" or "low-priority tasks".
The hybrid deployment solves the problem of low CPU utilization, but can lead to CPU competition among the hybrid tasks, thereby affecting the originally important business metrics (such as runtime duration, latency, etc.). To solve this problem, CPU QoS introduces an absolute preemption mechanism, allowing important business processes to unconditionally preempt less important business processes when they are runnable. This ensures that important business processes remain unaffected while the overall CPU utilization is improved.

Basic Principles

Users can set different priorities for tasks/containers to specify the scheduling behavior of the kernel for them. For instance, if a container with priority 0 and a container with priority 7 are running on the same CPU, once the container with priority 0 can run, it will unconditionally preempt the container with priority 7. The processes in the containers with lower priorities can be scheduled for execution only when the processes of the containers with higher priorities are either completed or blocked and asleep.
The QoS Agent allows users to set 8 different priority levels (0–7). Containers with priorities 0–6 do not have an absolute preemption relationship, and resources are divided and throttled based on the CPU usage proportion. For two businesses, if one is set to priority 0 and the other to priority 6, resources will not be divided based on priority. Resource division occurs only when there is node resource competition, where the resources of different Pods are proportionally divided according to the requested amount of resources. For more details, refer to Documentation. By setting some offline operations to priority 7, machine utilization is improved while ensuring that high-priority and latency-sensitive online business is not affected by offline operations.
The CPU QoS behavior among different priorities is as follows:



Operation Steps

1. Deploy QoS Agent.
2. Within the **Component Management** page in the cluster, locate the successfully deployed QoS Agent, and click **Update Configuration** on the right.
3. On the add-on configuration page for modifying QoS Agent, select **CPU Usage Priority**.
4. Click **Complete**.
5. Deploy offline services.
6. Deploy the PodQOS objects associated with the offline services, and select the business that needs to lower the priority, as shown below:

```
apiVersion: ensurance.crane.io/v1alpha1
kind: PodQOS
metadata:
  name: low
spec:
  labelSelector:
    matchLabels:
```

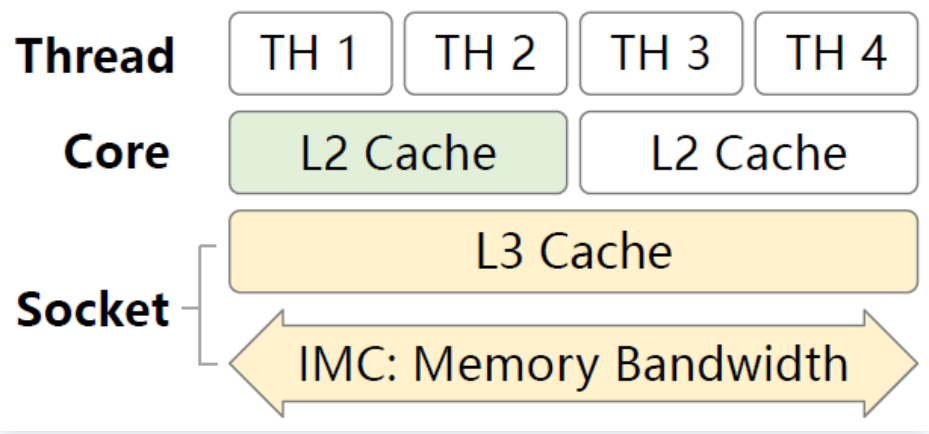
```
      k8s-app: low # select the Label of the business that needs to
lower the priority
  resourceQOS:
    cpuQOS:
      cpuPriority: 7 # Set the cpuPriority of the specified business
to 7, which means the lowest priority, and the business will be
preempted first in resource competition
```

CPU Hyper-Threading Isolation

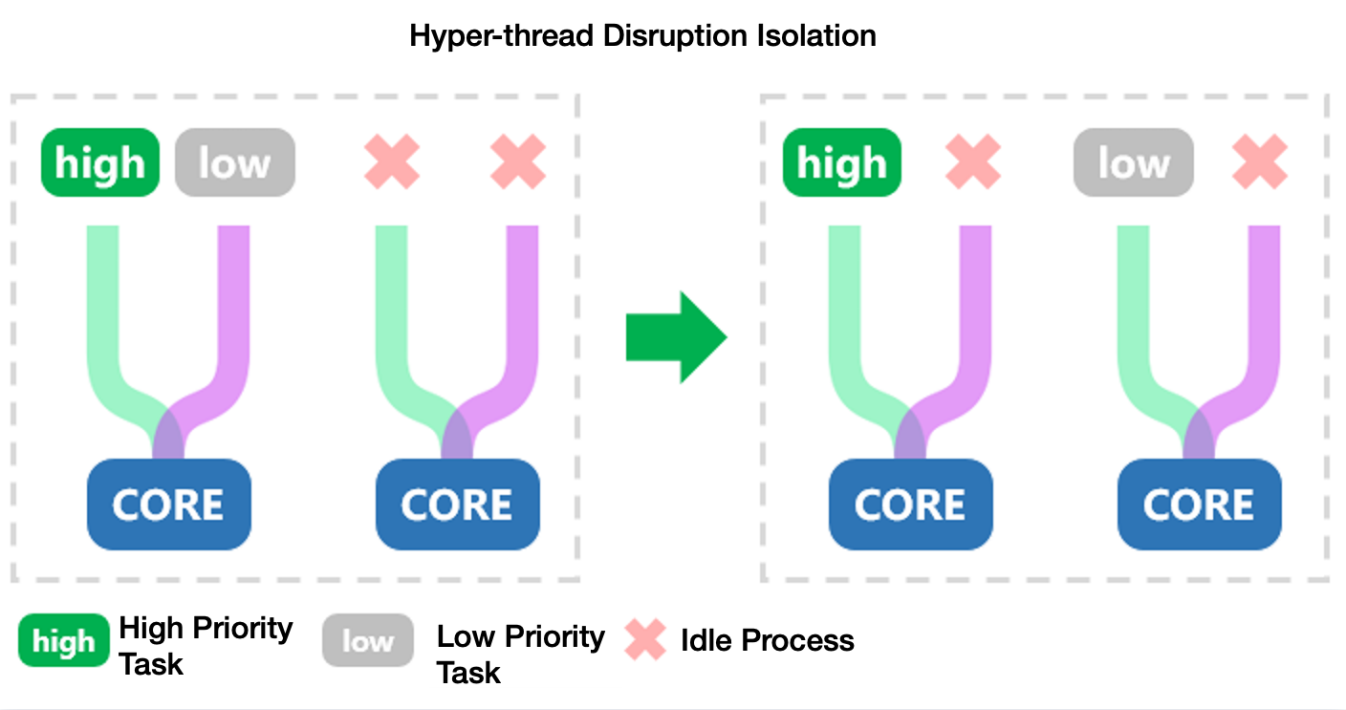
Last updated：2024-12-24 15:40:06

Overview

Common SMP processors have multiple levels, including the structures of threads, physical cores, and processors. When hyper-threading (HT) is enabled, a physical core typically contains 2 threads, and both threads on the same core share the L2 Cache.



When high-priority containers and low-priority containers run simultaneously, the threads of high-priority containers may be executed on the same physical core or even on the same HT CPU as the thread of the low-priority containers. In this case, although the CPU Usage Priority ensures that the threads of high-priority containers can always preempt the threads of low-priority containers, the threads of low-priority containers will occupy the shared L2 Cache on the physical core as long as they run, affecting the L2 Cache for high-priority container threads.



To prevent the L2 Cache for the threads of high-priority containers from being affected by the threads of low-priority containers running on the same physical core, the QoS Agent introduces the HT isolation mechanism. When there are rich processor resources, it ensures that there is no interference from threads of low-priority containers on the physical cores for the threads of high-priority containers.
HT interference isolation policies in different scenarios:

Thread 1	Thread 2	Action
High priority	High priority	Take no action
High priority	Low priority	Throttle low-priority tasks on siblings and restrict the migration of these low-priority tasks to siblings at the Cloud Load Balancer (CLB).
Low priority	Low priority	Try to pull high-priority tasks from other cores.

Operation Steps

1. Deploy QoS Agent.
2. On the **Add-on Management** page in the cluster, locate the successfully deployed QoS Agent, and click **Update Configuration** on the right.
3. On the configuration page for modifying the QoS Agent component, select **CPU HT Isolation**.
4. Select **CPU Usage Priority** to identify a high-priority business.
5. Click **Complete**.

6. Deploy a business.
7. Deploy the PodQOS objects associated with this business, and select the Label of the Workload that needs to use HT Isolation, as shown below:

```
apiVersion: ensurance.crane.io/v1alpha1
kind: PodQOS
metadata:
  name: ht-1
spec:
  labelSelector:
    matchLabels:
      k8s-app: memcached # Select the Label of the business that needs
to lower the priority
  resourceQOS:
    cpuQOS:
      cpuPriority: 0 # Identify as a high-priority business
      htIsolation:
        enable: true # Enable the CPU HT Isolation capability.
'enable' can be true/false, indicating whether HT isolation is enabled
for the business associated with PodQOS
```

CPU Surge During Application Startup

Last updated：2024-12-24 15:40:35

Overview

Usually, during the release or restart of an application, there might be a significant increase in CPU utilization, potentially consuming all cluster resources. This is mainly because the Java Virtual Machine (JVM) needs to perform class loading and object initialization during application startup, leading to the CPU handling more compilation tasks throughout the process.
In a Kubernetes scenario, applications are deployed through containers within Pods, and each container can set Requests and Limits to control the minimum and maximum resource usage. If the Limit is set too low, it may result in slow application startup or even failure to start due to insufficient resources. Conversely, if the Limit is set too high, the application's operation may excessively utilize resources, creating resource competition with other applications.

Overview

Operation Steps

Set the startup time of the application and the multiplier for increasing the Limit, thereby increasing the upper limit of resources used by the application during the initial startup phase.

Notes

- The capability of resource surge at business startup depends on the kernel of the native nodes. This feature is only supported in native nodes.
- Before using this feature, you need to install the QoS Agent in advance. Ensure the QoS Agent component version is 1.1.4 or above.

How the Feature Works

In Kubernetes, resource limits (Limits) are controlled by the CPU cgroup control module's cpu.cfs_period_us and cpu.cfs_quota_us configurations. Kubernetes configures these settings for the container's cgroup. By adjusting the cpu.cfs_quota_us value on the node where the Pod is located, you can achieve temporary resource surges, thereby increasing the resource usage limit during the application startup phase.

Operation Steps

Installing Add-ons

1. Log in to the Tencent Kubernetes Engine Console, and choose **Cluster** from the left navigation bar.
2. In the cluster list, click on the desired cluster ID to access its detailed page.

3. Select **Add-on management** from the left-side menu, and click on **Create** within the component management page.
4. On the **New Add-on Management** page, check **QoS Agent**.
5. Click on **Done** to install the add-on.

Resource Surge Capability at Application Startup

1. On the **Component Management** page, check the QoS Agent component version. Ensure the QoS Agent component version is 1.1.4 or above. If the version is below this requirement, click **Upgrade** on the right side of the component to upgrade.
2. Create a PodQoS CRD object via kubectl and apply it to the workload that requires a surge. Example code is as follows:

```
apiVersion: ensurance.crane.io/v1alpha1
kind: PodQOS
metadata:
  name: exceed
spec:
  labelSelector:
    matchLabels:
      app: 0kusa3 # Match the tag on the workload that needs a surge.
  resourceQOS:
    cpuQOS:
      limitExceed:
        time: 5    # Time after which a surge is required (in mins)
        value: "1.2"    # Multiplication for the Limit surge
```

3. Create the workload. Example code is as follows:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: 0kusa3    # Must be consistent with the matchLabels in the
PodQoS object.
spec:
  replicas: 3
```

```
  selector:
    matchLabels:
      app: 0kusa3
  template:
    metadata:
      labels:
        app: 0kusa3
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

4. Log in to the node where the Pod is located and check the cfs_quota value corresponding to the container on the node.

   Path:

```
/sys/fs/cgroup/cpu/kubepods/burstble/pod<id>/<containerid>/cpu.cfs_qu
ota_us
```

   Example:

```
/sys/fs/cgroup/cpu/kubepods/burstable/pod44ad741a-cbe6-48a2-a5b7-
b6920df2fe40/645a0a88f23c2aca4b207ec2c3d83bbe760a5f2a4db812938042df5cb
5054e18/cpu.cfs_quota_us
```

   Where:

   - pod<id>:<id> comes from the UID field in the Pod YAML.
   - <containerid>: comes from the containerID field in the Pod YAML.

   Before creating PodQoS, the value of cpu.cfs_quota_us is cpu.cfs_period_us *Limit. For example: 100000* 0.5 cores = 50000.

```
[[root@VM-21-7-centos 645a0a88f23c2aca4b207ec2c3d83bbe760a5f2a4db812938042df5cb5054e18]# cat cpu.cfs_quota_us
50000
```

   After creating PodQoS, within the first 5 minutes after the Pod is recreated, the new cpu.cfs_quota_us value is the old value multiplied by 1.2, meaning the Limit is increased.

> **Note:**
> The Pod ID will change after the Pod is recreated.

```
[root@VM-21-7-centos 197dbd54d23a17ebded1dea9c21e644c2a1bb17b26a727a8701848d612aa179b]# cat cpu.cfs_quota_us
60000
```

Fine-Grained Memory Scheduling

Last updated：2024-12-24 15:40:59

The fine-grained scheduling capability of memory provides a series of features to ensure the service quality for business memories. It comprehensively enhances the memory performance and flexibly limits the memory usage of a container.

Feature 1: Asynchronous Reclamation

Overview

During the memory asynchronous reclamation, a threshold is allowed to be set within the container. Once this threshold is exceeded, backend asynchronous reclamation will be triggered, ensuring a certain amount of idle memory to be used and maintained within the corresponding memcg. It provides assurance for the subsequent memory allocation and reduces the frequency of direct memory reclamation.

Operation Steps

1. Deploy QoS Agent.
2. On the Add-on management page in the cluster, locate the successfully deployed QoS Agent, and click **Update Configuration** on the right.
3. On the add-on configuration page for modifying QoS Agent, select **Memory QoS Enhancement**.
4. Click **Complete**.
5. Deploy a business.
6. Deploy the PodQOS object associated with the business, and select the business to apply, as shown below:

```
apiVersion: ensurance.crane.io/v1alpha1
kind: PodQOS
metadata:
  name: sql
spec:
  labelSelector:
    matchLabels:
      k8s-app: low # Select the business label to apply
  resourceQOS:
    memoryQOS:
      memAsyncReclaim:
```

```
        asyncRatio: 90 # asyncRatio represents the threshold for
asynchronous reclamation. Reclamation starts when memory usage exceeds
this proportion in the cgroup. The value range is [0-100]. It is
recommended to set above 90. It is 0 by default, which means off
        asyncDistanceFactor: 200 # asyncDistanceFactor controls the
total number of pages to be reclaimed each time asynchronous
reclamation is triggered. It is 1 by default, with the value range of
[1, 150000]
```

Feature 2: Classification of Global Threshold

Overview

Global threshold classification of the memory means that the cgroups with different priorities have different global memory thresholds; high-priority containers have lower thresholds and are more likely to obtain the memory under the same conditions; and low-priority containers have higher thresholds and are more likely to trigger reclamation to enter the direct reclamation process under the same conditions.

Operation Steps

1. Deploy QoS Agent.
2. On the Add-on management page in the cluster, locate the successfully deployed QoS Agent, and click the **Update Configuration** on the right.
3. On the add-on configuration page, select the **Memory QoS Enhancement**.
4. Click **Complete**.
5. Deploy a business.
6. Deploy the PodQOS object associated with the business, and select the business to apply, as shown below:

```
apiVersion: ensurance.crane.io/v1alpha1
kind: PodQOS
metadata:
  name: test
spec:
  labelSelector:
    matchLabels:
      k8s-app: low # Select the business label to apply
  resourceQOS:
    memoryQOS:
      memWatermark:
```

```
        watermarkRatio: 50 # The value range of watermarkRatio is
[-75,75]; negative values indicate the reduced threshold, mainly for
online containers; and positive values indicate the elevated
threshold, mainly for offline containers;
```

Create a PodQOS object for an offline business:

```
apiVersion: ensurance.crane.io/v1alpha1
kind: PodQOS
metadata:
  name: sql
spec:
  labelSelector:
    matchLabels:
      k8s-app: mysql
  resourceQOS:
    memoryQOS:
      memWatermark:
        watermarkRatio: -50
```

Feature 3: pagecache limit

Overview

Isolate the pagecache at the container level.

Operation Steps

1. Deploy QoS Agent.
2. On the Add-on management page in the cluster, locate the successfully deployed QoS Agent, and click the **Update Configuration** on the right.
3. On the add-on configuration page, select **Memory QoS Enhancement**.
4. Click **Complete**.
5. Deploy a business.
6. Deploy the PodQOS object associated with the business, and select the business to apply, as shown below:

```
apiVersion: ensurance.crane.io/v1alpha1
kind: PodQOS
metadata:
```

```
  name: sql
spec:
  labelSelector:
    matchLabels:
      k8s-app: mysql-pi-000006 # Select the business label to apply
  resourceQOS:
    memoryQOS:
      memPageCacheLimit:
        pageCacheMaxRatio: 20 # pageCacheMaxRatio represents the
maximum ratio of pagecache usage to memory quota, based on the current
memory limit. Therefore, to use this feature, memory thresholds must
be set in the limits. For example, if the Pod memory threshold is 10
GB and pageCacheMaxRatio is 20%, then the pagecache is limited to
using a maximum of 2 GB.
        pageCacheReclaimRatio: 5 # pageCacheReclaimRatio represents
the reclamation ratio after pagecache exceeds its limit, specifically
referring to the ratio of the maximum pagecache usage.
```

Fine-Grained Network Scheduling

Last updated：2024-12-24 15:47:21

The fine-grained scheduling capability of network provides a series of features to ensure the service quality of business networks, improving network performance in all aspects and flexibly limiting the use of network by a container.

Feature 1: Throttling of Inbound and Outbound Directions

Overview

Limit the inbound and outbound bandwidth of the container.

Operation Steps

1. Deploy QoS Agent.
2. Within the **Add-on Management** page in the cluster, locate the successfully deployed QoS Agent, and click **Update Configuration** on the right.
3. On the add-on configuration page for modifying QoS Agent, tick the **Network QoS Enhancement**.
4. Click **Complete**.
5. Deploy a business.
6. Deploy the PodQOS object associated with the business, and select the business to apply, as shown below:

```
apiVersion: ensurance.crane.io/v1alpha1
kind: PodQOS
metadata:
  name: a
spec:
  labelSelector:
    matchLabels:
      k8s-app: a # select the Label of the business that needs to
lower the priority
  resourceQOS:
    netIOQOS:
      netIOLimits:
        rxBps: 50 # rxBps represents the maximum inbound bandwidth,
in Mbps. 0 means no limit. The maximum input value is a 13-digit
integer.
```

```
        txBps: 50 # txBps represents the maximum outbound bandwidth,
in Mbps. 0 means no limit. The maximum input value is a 13-digit
integer.
```

Feature 2: Priority (Absolute Bandwidth Preemption)

Overview

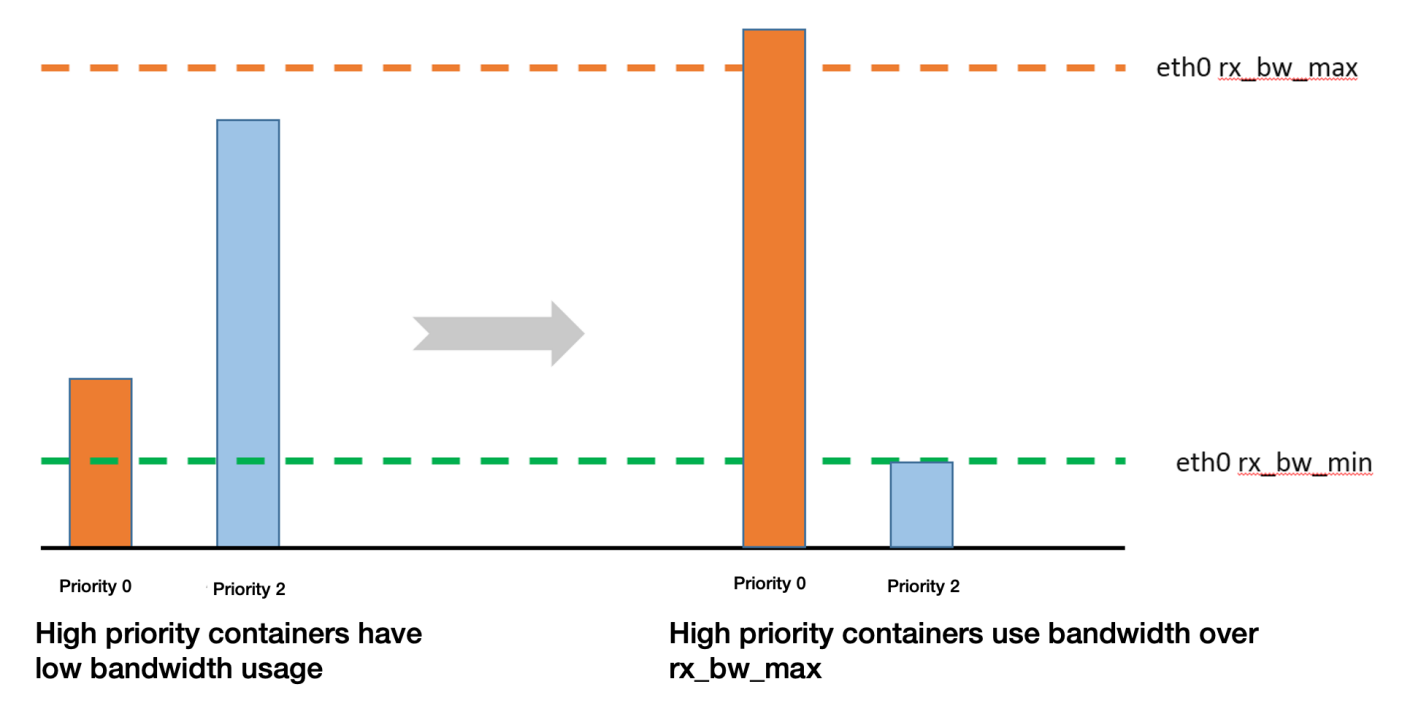
Users can set different priorities for different containers and allocate bandwidth resources of network interfaces according to the priorities. An example of scheduling policy for inbound bandwidth with three priority levels is provided, which can be extended to more priority levels:

- Containers with the highest priority (priority 0) are not subject to network QoS limitations and can use bandwidth resources freely.
- When the total bandwidth used by all containers with the lower priorities (priorities 1 and 2) is less than rx_bps_max minus the bandwidth of the highest priority, and the bandwidth which can be used by each container can is not subject to limitations and can exceed rx_bps_min.
- When the total bandwidth used by all containers with the lower priorities (priorities 1 and 2) exceeds rx_bps_max, the containers can use up to the difference between rx_bps_max and the total traffic of the containers with a higher priority. If the difference is less than rx_bps_min, then the containers with the lower priority are guaranteed to use the bandwidth of at least rx_bps_min.
- The bandwidth of the container with the highest priority (priority 0) can exceed rx_bps_max/tx_bps_max, while the containers with lower priorities cannot exceed this parameter.

Scenarios

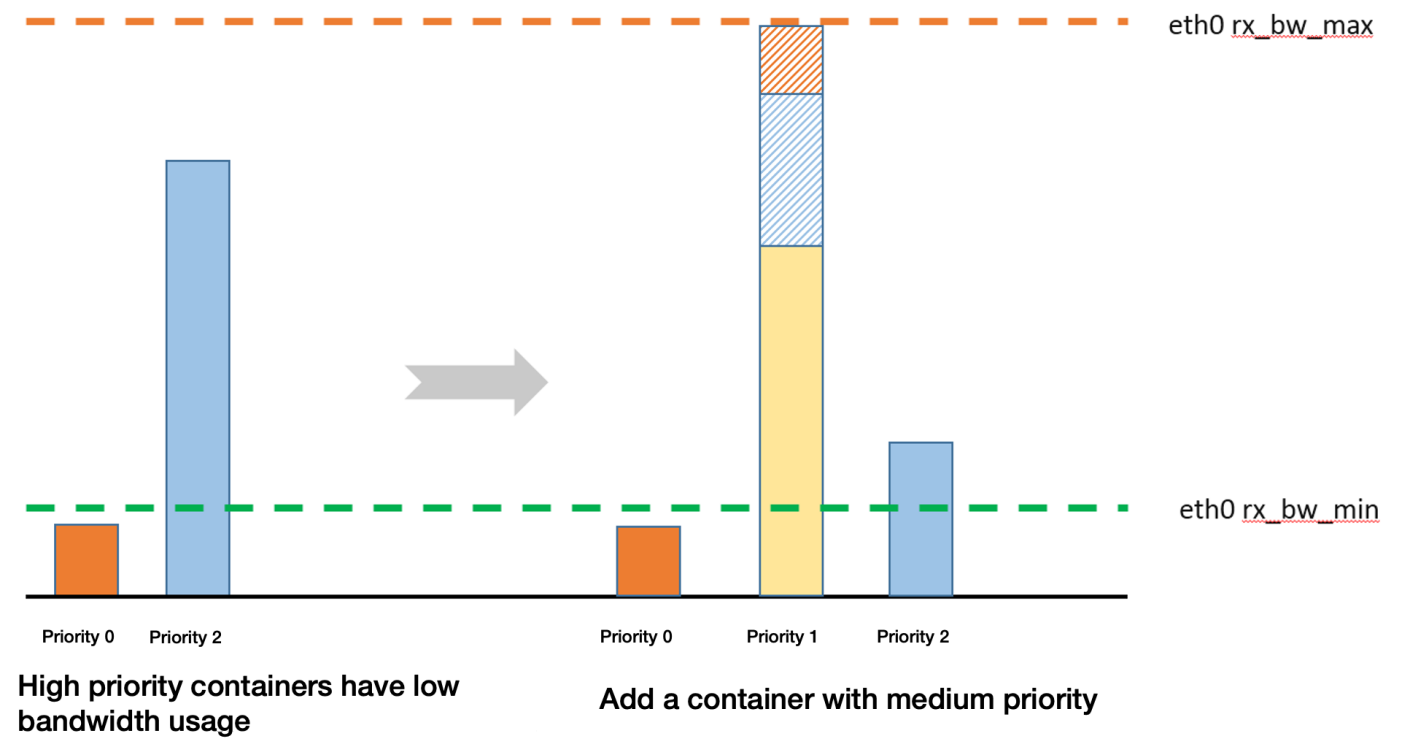
Scenario 1

- When containers with higher priorities have lower bandwidth usage, an idle bandwidth will be allocated to the containers with low priorities.
- When containers with higher priorities have higher bandwidth usage exceeding rx_bw_max, only the minimum guaranteed bandwidth will be allocated to the containers with low priorities.



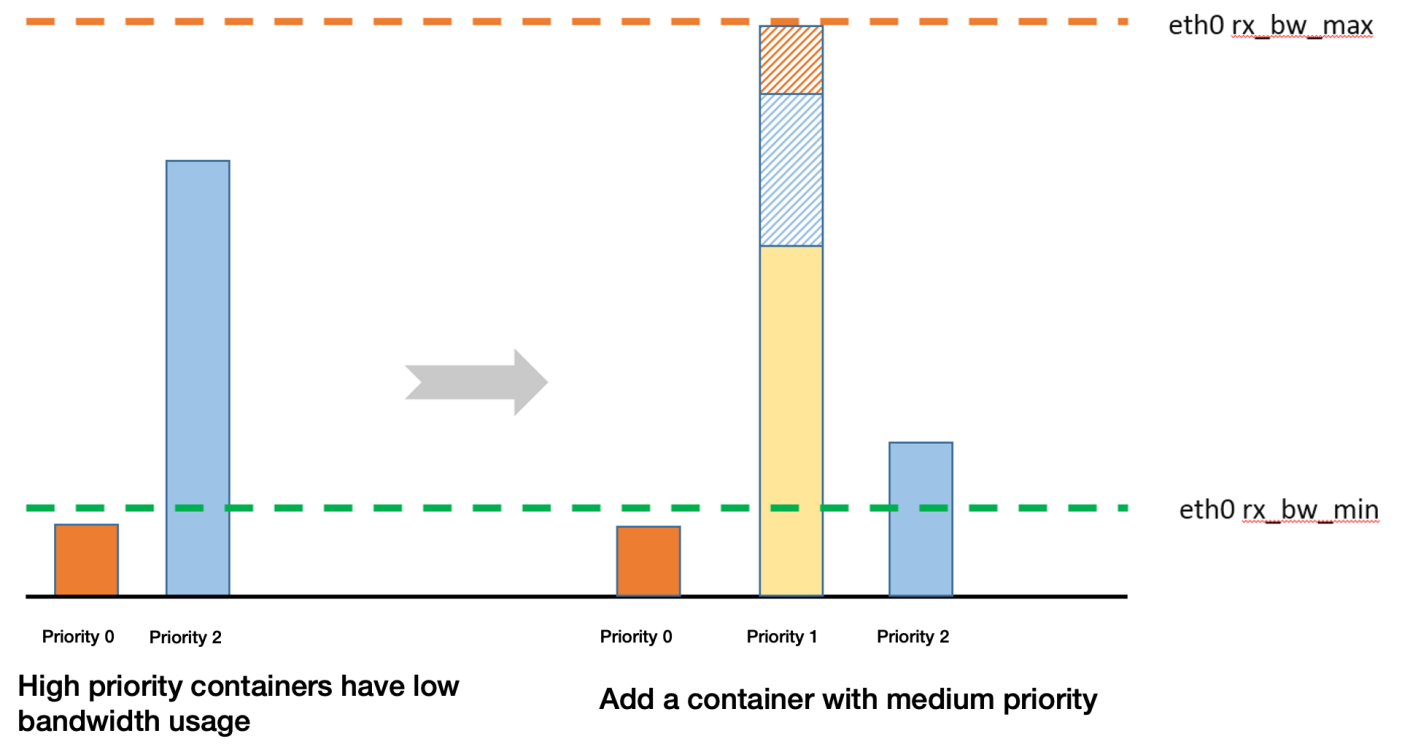
Scenario 2

- When containers with higher priorities have lower bandwidth usage, an idle bandwidth will be allocated to the containers with low priorities.
- When a container with medium priority is added, the bandwidth of the containers with relatively low priorities will be preempted.



Scenario 3

- When containers with higher priorities have lower bandwidth usage, an idle bandwidth will be allocated to the containers with low priorities.
- When a container with medium priority is added, the bandwidth of the containers with relatively low priorities will be preempted. The maximum bandwidth which can be preempted shall subtract the bandwidth used by the containers with higher priorities and the minimum guaranteed bandwidth of the containers with lower priorities.



Operation Steps

1. Deploy QoS Agent.
2. Within the **Add-on Management** page in the cluster, locate the successfully deployed QoS Agent, and click **Update Configuration** on the right.
3. On the add-on configuration page for modifying QoS Agent, tick the **Network QoS Enhancement**.
4. Click **Complete**.
5. Deploy the business A.
6. Deploy the PodQOS object associated with the business, and select the business to apply, as shown below:

```
apiVersion: ensurance.crane.io/v1alpha1
kind: PodQOS
metadata:
  name: a
spec:
  labelSelector:
    matchLabels:
      k8s-app: a # select the Label of the business
```

```
  resourceQOS:
    netIOQOS:
      netIOPriority: 6 # select the network priority of 0-7, a total
of 8 levels
```

7. Deploy the business B.
8. Deploy the PodQOS object associated with the business, and select the business to apply, as shown below:

```
apiVersion: ensurance.crane.io/v1alpha1
kind: PodQOS
metadata:
  name: b
spec:
  labelSelector:
    matchLabels:
      k8s-app: b # select the Label of the business
  resourceQOS:
    netIOQOS:
      netIOPriority: 7 # select the network priority of 0-7, a total
of 8 levels
```

This feature needs to be used jointly with the node bandwidth limitation in NodeQOS, specifying rxBpsMin, rxBpsMax, txBpsMin and txBpsMax with NodeQOS.

- rxBpsMin: The minimum guaranteed bandwidth in the inbound direction at the per-priority level. It is shared by the containers at this level. Currently, the minimum guaranteed bandwidth in the inbound direction is the same at different priority levels, except for priority 0.
- txBpsMin: The minimum guaranteed bandwidth in the outbound direction at the per-priority level. It is shared by the containers at this level. Currently, the minimum guaranteed bandwidth in the outbound direction is the same at different priority levels, except for priority 0.
- rxBpsMax: Maximum inbound bandwidth of the network interface.
- txBpsMax: Maximum outbound bandwidth of the network interface.

> **Note:**
> Unit: Mbps.

```
apiVersion: ensurance.crane.io/v1alpha1
kind: NodeQOS
```

```
metadata:
  name: total
spec:
  netLimits:
    rxBpsMin: 10
    rxBpsMax: 10240
    txBpsMin: 10
    txBpsMax: 10240
```

Feature 3: Port Allowlist

Overview

To prevent the containers with low priorities from starving, there are two mechanisms:

1. Minimum guaranteed bandwidth of per-priority: Shared by the containers with this priority.
2. Port allowlist mechanism: Users can add a local port or peer port from the container to the allowlist. The traffic of this port will not be throttled. It is generally used to protect control messages of specific protocols.

Operation Steps

1. Deploy QoS Agent.
2. Within the **Add-on Management** page in the cluster, locate the successfully deployed QoS Agent, and click **Update Configuration** on the right.
3. On the add-on configuration page for modifying QoS Agent, tick the **Network QoS Enhancement**.
4. Click **Complete**.
5. Deploy a business.
6. Deploy the PodQOS object associated with the business, and select the business to apply, as shown below:

```
apiVersion: ensurance.crane.io/v1alpha1
kind: PodQOS
metadata:
  name: a
spec:
  labelSelector:
    matchLabels:
      k8s-app: a # select the Label of the business
  resourceQOS:
    netIOQOS:
```

```
      whitelistPorts: # The local port 5201 and the remote port 5205
representing the business a are set as allowlist ports and are not
throttled. The default values for lports and rports are 0, with the
value range of 0- to 13-digit integers.
        lports: 5201
        rports: 5205
```

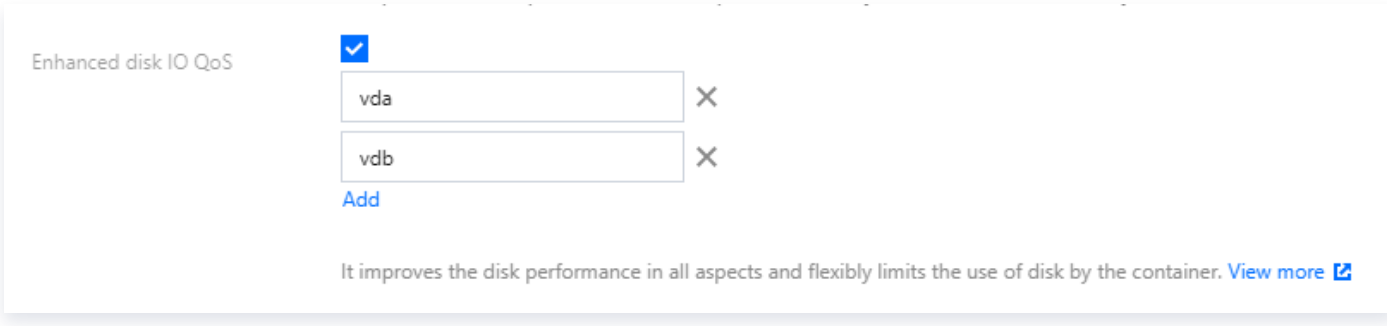
Fine-Grained Disk IO Scheduling

Last updated：2024-12-24 15:47:53

The fine-grained scheduling capability of disk IO provides a series of features to ensure the service quality for business disks and flexibly limit the usage of disk transfers by a container.

Use Limits

1. Deploy QoS Agent.
2. Within the **Add-on management** page in the cluster, locate the successfully deployed QoS Agent, and click **Update Configuration** on the right.
3. On the add-on configuration page for modifying QoS Agent, select **Disk IO QoS Enhancement**.
4. Enter the name of disk with limits specified by parameters. As shown in the following figure, you can enter multiple names:



> **Note:**
> 1. The disk names shall match the actual device names. For information on how to obtain the disk names of a node, please refer to How to Determine the Disk Name of a Node.
> 2. When entering in the console, please click **Add** to add a new disk name once the name of a disk device is entered each time. Do not enter multiple disk names at a time.
> 3. If the corresponding disk name is missing on the node, fine-grained scheduling for disk IO will not take effect.
> 4. This feature is not supported for non-native nodes. Please ensure that there are native nodes in the cluster, and that Pods configured with Disk IO limits are scheduled on the native nodes.

5. Click **Complete**.

Feature 1: Disk IOPS Limits (direct IO + buffer IO)

1. Deploy the components according to the above usage limits, enable the relevant switches, and enter the

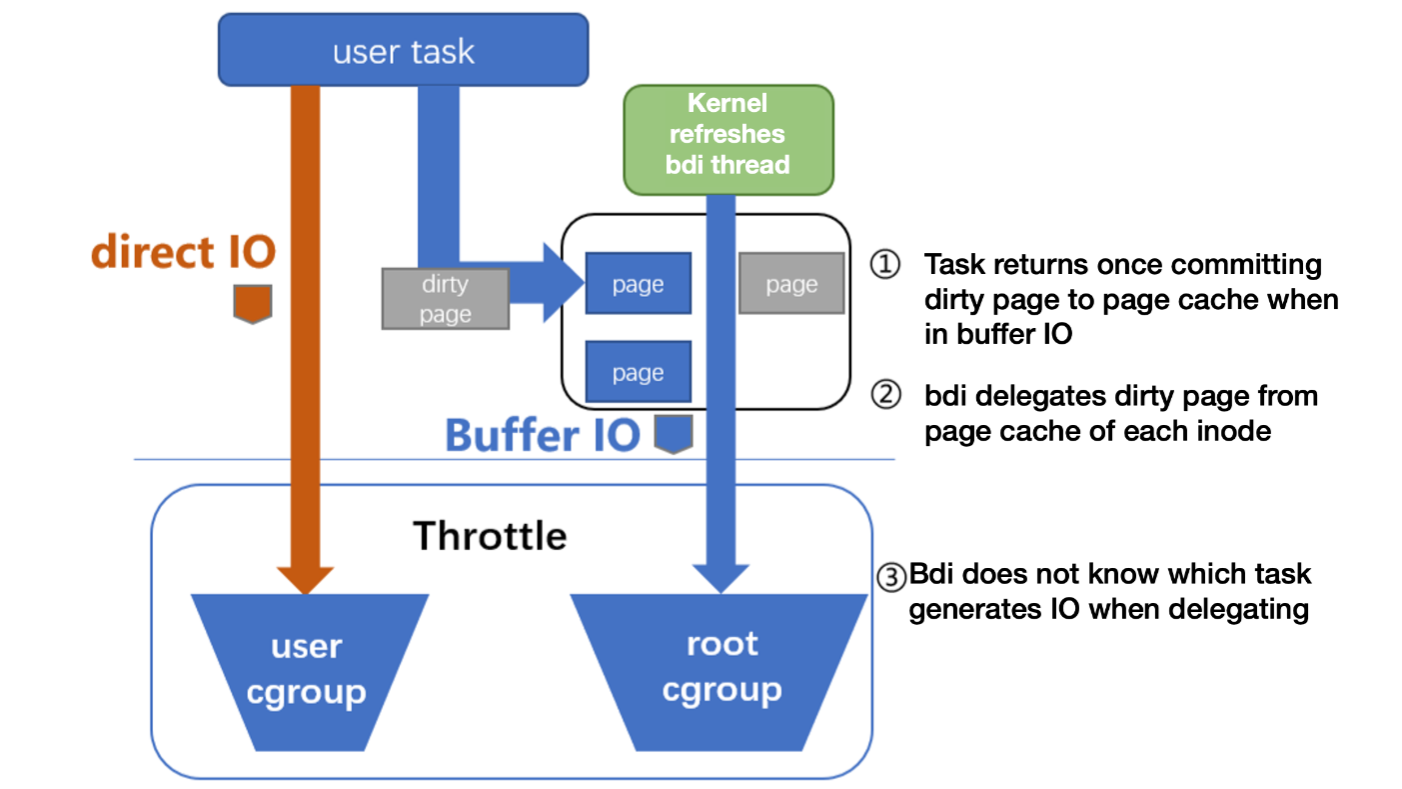
relevant disk names.

2. Deploy a business.
3. Deploy the PodQOS object associated with the business, and select the business to apply, as shown below:

```
apiVersion: ensurance.crane.io/v1alpha1
kind: PodQOS
metadata:
  name: a
spec:
  labelSelector:
    matchLabels:
      k8s-app: a # select the Label of the business
  resourceQOS:
    diskIOQOS:
      diskIOLimit:
        readIOps: 1024 # readIOps represents the limit on the amount
of read IO for the Pod, measured in IOPS/s
        writeIOps: 1024 # writeIOps represents the limit on the
amount of write IO for the Pod, measured in IOPS/s
```

Throttling Mechanism for Buffer IO

Due to a low-version kernel or cgroup v1 not being able to effectively throttle the buffer IO, it may cause the Buffer IO to interfere with the normal business IO (the direct IO is usually used in a database scenario). TKE provides the Buffer IO throttling feature based on cgroup v1. The Buffer IO throttling based on cgroup v2 is consistent with the upstream kernel.
The main reason of cgroup v1 not being able to support throttling is that, during the asynchronous dirty page flushing, the kernel cannot determine which blkio cgroup this IO should be submitted to. For example:



To solve this accounting problem, TKE binds the cgroup of the page cache with the corresponding blkio cgroup. Then the kernel thread can determine the target blkio cgroup when asynchronously flushing.

Feature 2: Disk BPS Limits (direct IO + buffer IO)

1. Deploy the components according to the above usage limits, enable the relevant switches, and enter the relevant disk names.
2. Deploy a business.
3. Deploy the PodQOS object associated with the business, and select the business to apply, as shown below:

```
apiVersion: ensurance.crane.io/v1alpha1
kind: PodQOS
metadata:
  name: a
spec:
  labelSelector:
    matchLabels:
      k8s-app: a
  resourceQOS:
    diskIOQOS:
      diskIOLimit:
```

```
        readBps: 1048576 # readBPS limits the read bandwidth of the
Pod, measured in mbps
        writeBps: 1048576 # writeBPS limits the write bandwidth of
the Pod, measured in mbps
```

FAQs

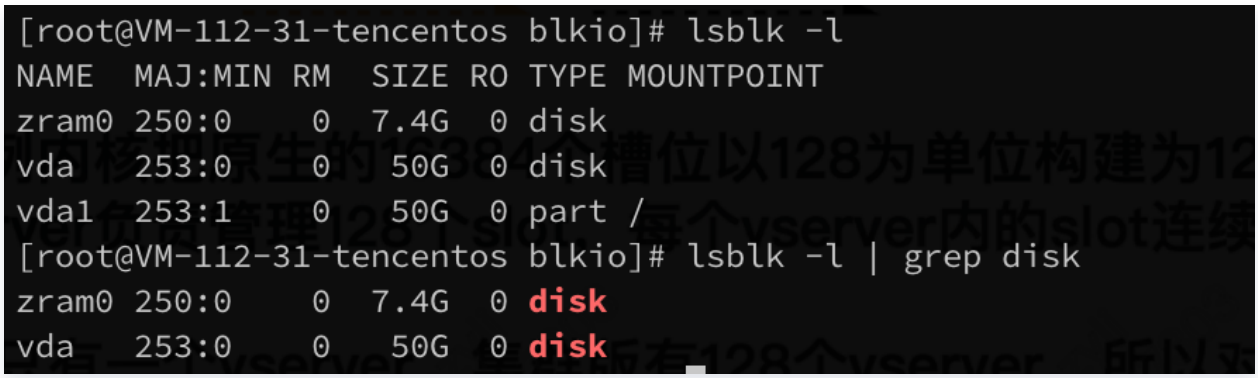
How to Determine the Disk Name of a Node?

1. Log in to the native nodes where the disk IO needs to be enabled. For operation details, please refer to Enabling SSH Key Login for a Native Node.
2. Execute the following command to obtain all disk devices:

```
lsblk -l | grep disk
```

The execution result is shown below:

> **Note:**
> 1. In the image below, zram0 is a module used for virtual memory compression, not a disk name.
> 2. In the image below, vda1 is a part partition block, also not a disk name.
> 3. The example below only shows one disk with the disk name of: vda.



Community BFQ Issues May Cause Native Node Kernel Panics

Suggestion: Upgrade the version of QoS Agent to 1.1.2 or above, avoiding kernel panics on native nodes due to a bug in the Linux community kernel BFQ module. The relevant issues are as follows:

- kernel/git/stable/linux.git – Linux kernel stable tree
- Re: [PATCH 0/5 v2] bfq: Avoid use-after-free when moving processes between cgroups – Jan Kara

Query Solution: You can verify the scheduling policy by executing the command of `cat /sys/block/[disk_name]/queue/scheduler` on the node. If the output contains the field of

`[bfq]`, it means that the BFQ is enabled and changes shall be made.

Modification Solution: Change the scheduling policy to mq-deadline. For example, if the disk name is vda, you can modify it as follows:

```
echo mq-deadline > /sys/block/vda/queue/scheduler
```