系统可用性分析

Kubernetes节点通常运行许多OS系统守护进程，除了像kubelet，运行时等之类的kubernetes守护进程和用户pod。 Kubernetes假定节点中所有可用的计算资源（称为Capacity）可用于用户pod。 实际上，系统守护进程使用非常少量的资源，其可用性对系统的稳定性至关重要。 为了解决这个问题，本提案介绍了可分配的概念，它标识可用于用户pod的计算资源的数量。 具体来说，kubelet将提供几个旋钮来为OS系统守护程序和kubernete守护程序预留资源。

通过明确保留计算资源，其目的是避免过度使用节点，并且不会有系统守护进程与用户pod竞争。 系统守护程序和用户pod可用的资源将根据用户指定的预留进行限制。

如果可分配可用，调度程序将使用而不是容量，从而不会超过节点。

设计

定义

节点容量 - 已经作为NodeStatus.Capacity提供，这是从节点实例读取的总容量，并假定为常量。

     系统保留（提议） - 为未由Kubernetes管理的流程预留的计算资源。 目前，这涵盖了/系统原始容器中集中的所有进程。

     Kubelet可分配 - 计算可用于调度的资源（包括计划和非计划资源）。 这个价值是这个建议的重点。 请参阅下面的更多细节。

     Kube-Reserved（提出） - 为诸如docker守护进程，kubelet，kube代理等的Kubernetes组件预留的计算资源。

API的变化

Allocatable

在NodeStatus中添加Allocatable

type NodeStatus struct {

...

// Allocatable represents schedulable resources of a node.

Allocatable ResourceList `json:"allocatable,omitempty"`

...

}

分配将由Kubelet计算并报告给API服务器。 它被定义为：

[Allocatable] = [Node Capacity] - [Kube-Reserved] - [System-Reserved] - [Hard-Eviction-Threshold]

调度pod时，调度程序将使用Allocatable代替Capacity，Kubelet将在执行准入检查时使用它。

Notice:系统层面的波动不再Kubernetes的控制,通过提供监控的metrics API来Reporting Kernel usage.

#### Kube-Reserved

KubeReserved是指定为kubernetes组件保留的资源的参数（4）。 它在启动时作为命令行标志提供给Kubelet，因此在正常的Kubelet操作期间不能更改（这可能会在将来更改）。

该标志将被指定为序列化资源列表，资源由API ResourceName定义，并以resource.Quantity格式指定，值为

--kube-reserved=cpu=500m,memory=5Mi

最初我们只会支持CPU和内存，但最终将支持更多的资源，如本地存储和io比例权重，以提高节点的可靠性。

#### System-Reserved

在初始实现中，SystemReserved在功能上等同于KubeReserved，但具有不同的语义含义。 虽然KubeReserved指定了用于kubernetes组件的资源，SystemReserved指定为非kubernetes组件留出的资源（目前报告为非系统节点上/ system raw容器中的所有进程集中在一起）。

28/5000

Kubelet驱逐阈值

为了提高节点的可靠性，每当节点运行内存不足或本地存储时，kubelet都会逐渐消灭pod。一起，驱逐和节点分配有助于提高节点的稳定性。

从v1.5起，驱逐是基于相对于容量的总体节点使用。 Kubelet根据QoS和用户配置的逐出阈值驱逐pod。本文档中有更多的邮资

从v1.6开始，如果使用cgroups，所有pod上的所有pod都默认实现Allocatable，则pod不能超过Allocatable。内存和CPU限制是使用cgroup来实现的，但是并不存在强制执行存储限制的简单方法。使用Linux配额实现存储限制是不可能的，因为它不是分级的。一旦存储作为Allocatable的资源支持，Kubelet除了能力之外还要执行基于可分配的驱逐。

请注意，驱逐限制仅在pod上实施，系统守护程序可以自由使用任何数量的资源，除非执行其保留。

下面是一个例子来说明节点可分配的内存：

节点容量为32Gi，kube保留为2Gi，系统保留为1Gi，逐出硬设置为<100Mi

对于该节点，有效节点分配只有28.9Gi;即如果kube和系统组件占用了所有的预留，则可用于pod的内存只有28.9Gi，一旦pod的总体使用量超过该阈值，kubelet就会驱逐pod。

如果我们通过顶级cgroup强制执行节点分配（28.9Gi），则pod不能超过28.9Gi，在这种情况下，除非内存内存消耗高于100Mi，否则将不执行驱逐。

为了支持驱逐，并避免memcg OOM杀死pod，我们将把pod的顶级cgroup限制设置为Node Allocatable + Eviction Hard Thresholds。

但是，调度程序不会使用超过28.9Gi，因此节点分配节点状态将为28.9Gi。

如果kube和系统组件没有占用所有的预留，那么在上面的例子中，荚将面临memcg OOM，在kubelet驱逐出现之前，可以从节点可分配cgroup中杀死。为了在这种情况下更好地执行QoS，Kubelet将应用硬驱逐阈值在节点可分配cgroup上，如果节点可分配是强制的。用户pod的结果行为将相同。通过上面的例子，每当pod消耗超过28.9Gi的时候，Kubelet将会消除pod，这将是29Gi的<100Mi，这将是Node Allocatable cgroup的内存限制。

一般准则

系统守护进程将被视为与Guaranteed pod。系统守护进程可能会在其有限的cgroup中爆发，并且此行为需要作为kubernetes部署的一部分进行管理。例如，Kubelet可以拥有自己的cgroup并与Container Runtime共享KubeReserved资源。但是，如果KubeReserved被强制执行，Kubelet不能突发并使用所有可用的Node资源。

建议用户在执行SystemReserved预留时要特别小心，因为它可能导致关键服务被CPU饿死或OOM在节点上死亡。该建议只有在用户对其节点进行了详尽的剖析来得出精确的估计值时才强制执行SystemReserved。

开始仅在pod上执行分配。一旦有足够的监视和警报来跟踪kube守护进程，则试图根据启发式来实施KubeReserved。更多在第二阶段。

随着越来越多的功能被添加，kube系统守护进程的资源需求将随着时间的推移而增长。随着时间的推移，该项目将试图降低利用率，但这不是目前的优先事项。所以期望随着时间的推移，分配能力下降。

sysd-logind将ssh会话放在/user.slice下。其使用情况将不会在节点中予以考虑。配置SystemReserved时，请考虑/user.slice的资源预留。理想情况下，/user.slice应驻留在SystemReserved顶级cgroup下。

推荐的Cgroups设置

以下是Kubernetes节点推荐的cgroup配置。 所有OS系统守护进程都将被置于一个顶级SystemReserved cgroup下。 Kubelet和Container Runtime将被放置在KubeReserved cgroup下。 推荐在KubeReserved下放置Container Runtime的原因如下：

     Kubernetes节点上的容器运行时间预计不会在Kubelet之外使用。

     资源消耗与节点上运行的pod数量有关。

请注意，下面的层次结构建议为kubelet和运行时间单独跟踪其使用情况。

|  |
| --- |
| / (Cgroup Root)  .  +..systemreserved or system.slice (Specified via `--system-reserved-cgroup`; `SystemReserved` enforced here \*optionally\* by kubelet)  . . .tasks(sshd,udev,etc)  .  .  +..podruntime or podruntime.slice (Specified via `--kube-reserved-cgroup`; `KubeReserved` enforced here \*optionally\* by kubelet)  . .  . +..kubelet  . . .tasks(kubelet)  . .  . +..runtime  . .tasks(docker-engine, containerd)  .  .  +..kubepods or kubepods.slice (Node Allocatable enforced here by Kubelet)  . .  . +..PodGuaranteed  . . .  . . +..Container1  . . . .tasks(container processes)  . . .  . . +..PodOverhead  . . . .tasks(per-pod processes)  . . ...  . .  . +..Burstable  . . .  . . +..PodBurstable  . . . .  . . . +..Container1  . . . . .tasks(container processes)  . . . +..Container2  . . . . .tasks(container processes)  . . . .  . . . ...  . . .  . . ...  . .  . .  . +..Besteffort  . . .  . . +..PodBesteffort  . . . .  . . . +..Container1  . . . . .tasks(container processes)  . . . +..Container2  . . . . .tasks(container processes)  . . . .  . . . ...  . . .  . . ... |

系统保留和kube保留的cgroups预计将由用户创建。 如果Kubelet为自身和docker守护进程创建cgroup，它将自动创建kubereserved cgroups。

kubepods cgroups将由kubelet自动创建，如果它还没有。 kubepods cgroup的创建与QoS Cgroup的支持有关，它由--cgroups-per-qos标志控制。 如果cgroup驱动程序设置为systemd，那么Kubelet将通过systemd创建一个kubepods.slice。 默认情况下，Kubelet将直接通过cgroupfs将mkdir / kubepods cgroup。

容器化kubelet

如果使用容器运行时管理Kubelet，请运行时为kubereserved下的kubelet创建cgroup。

度量

Kubelet识别它是自己的cgroup，并通过Summary metrics API（/ stats / summary）公开其使用指标。通过docker运行时，kubelet还会识别docker运行时的cgroup，并通过Summary metrics API公开其指标。 为了提供一个节点的完整概述，Kubelet将公开来自执行SystemReserved，KubeReserved和Allocatable的cgroup的指标。

实现阶段

阶段1 - 引入可分配给系统而不执行

在这个阶段，Kubelet将支持通过kubelet标志指定KubeReserved和SystemReserved资源预留。 这些标志的默认值为“”，表示零cpu或内存预留。 Kubelet将计算分配并更新Node.Status以包含它。 如果可用，调度程序将使用可分配而不是容量。

阶段2 - 在Pods上执行分配

状态:目标1.6

在这个阶段，Kubelet将自动创建一个顶级cgroup，以便在所有用户端口上实施节点分配。 此cgroup的创建由--cgroups-per-qos标志控制。

Kubelet将支持指定KubeReserved和SystemReserved的顶级cgroup，并支持可选地在这些顶级cgroup上放置资源限制。

用户应根据其部署要求指定KubeReserved和SystemReserved。

Kubelet和运行时的资源需求通常与节点上运行的pod数量成比例。 一旦用户识别出每个节点的最大pod密度，他们将能够使用该性能仪表板来计算KubeReserved。 这个博客文章解释了如何解释仪表板。 请注意，此仪表板仅提供停泊运营商的使用指标。

将在此阶段引入支持基于分配的驱逐。

本阶段推出的新旗帜如下：

New flags introduced in this phase are as follows:

--enforce-node-allocatable=[pods][,][kube-reserved][,][system-reserved]

* 1. This flag will default to pods in v1.6.
  2. This flag will be a no-op unless --kube-reserved and/or --system-reserved has been specified.
  3. If --cgroups-per-qos=false, then this flag has to be set to "". Otherwise its an error and kubelet will fail.
  4. It is recommended to drain and restart nodes prior to upgrading to v1.6. This is necessary for --cgroups-per-qos feature anyways which is expected to be turned on by default in v1.6.
  5. Users intending to turn off this feature can set this flag to "".
  6. Specifying kube-reserved value in this flag is invalid if --kube-reserved-cgroup flag is not specified.
  7. Specifying system-reserved value in this flag is invalid if --system-reserved-cgroup flag is not specified.
  8. By including kube-reserved or system-reserved in this flag's value, and by specifying the following two flags, Kubelet will attempt to enforce the reservations specified via --kube-reserved & system-reserved respectively.

--kube-reserved-cgroup=<absolute path to a cgroup>

* 1. This flag helps kubelet identify the control group managing all kube components like Kubelet & container runtime that fall under the KubeReserved reservation.
  2. Example: /kube.slice. Note that absolute paths are required and systemd naming scheme isn't supported.

--system-reserved-cgroup=<absolute path to a cgroup>

* 1. This flag helps kubelet identify the control group managing all OS specific system daemons that fall under the SystemReserved reservation.
  2. Example: /system.slice. Note that absolute paths are required and systemd naming scheme isn't supported.

--experimental-node-allocatable-ignore-eviction-threshold

* 1. This flag is provided as an opt-out option to avoid including Hard eviction thresholds in Node Allocatable which can impact existing clusters.
  2. The default value is false.

### Phase 3 - Metrics & support for Storage

Status: Targeted for v1.7

In this phase, Kubelet will expose usage metrics for KubeReserved, SystemReserved and Allocatable top level cgroups via Summary metrics API. Storage will also be introduced as a reservable resource in this phase.

Kubelet的驱逐策略

本文档介绍了当计算资源太低时，kubelet如何排除pod的规范。

目标

当可用的计算资源较低时，节点需要一种保持稳定性的机制。

这在处理不可压缩的计算资源（如内存或磁盘）时尤其重要。如果任一资源耗尽，节点将变得不稳定。

通过使系统OOM杀手级别看到更高的OOM得分调整相对于他们的请求消耗了最大内存量的容器的分数，kubelet有一些支持来影响系统行为响应系统OOM。系统OOM事件非常计算密集，并可能停止节点，直到OOM杀死进程完成。此外，由于由于OOM而被杀死的容器重新启动或新的pod被安排到节点上，所以系统容易返回到不稳定状态。

相反，我们更喜欢一个系统，其中kubelet可以主动监视和防止计算资源的完全饥饿，并且在可能发生的情况下，主动地失败一个或多个pod，因此工作负载可以当/其后备控制器创建一个新的pod时，被移动并安排在其他地方。

提案范围

此提案定义了用于回收计算资源的吊舱撤销策略。

截至目前，支持内存和磁盘驱逐。 该提案侧重于简单的默认驱逐策略，旨在涵盖最广泛的用户工作负载类别。

触发信号

Kubelet将会支持根据下面的信号来决定驱逐决定

| **Eviction Signal** | **Description** |
| --- | --- |
| memory.available | memory.available := node.status.capacity[memory] - node.stats.memory.workingSet |
| nodefs.available | nodefs.available := node.stats.fs.available |
| nodefs.inodesFree | nodefs.inodesFree := node.stats.fs.inodesFree |
| imagefs.available | imagefs.available := node.stats.runtime.imagefs.available |
| imagefs.inodesFree | imagefs.inodesFree := node.stats.runtime.imagefs.inodesFree |

以上每个信号支持基于字面或百分比的值。 相对于与每个信号相关联的总容量计算基于百分比的值。

kubelet只支持两个文件系统分区。

     kubelet用于卷，守护程序日志等的nodefs文件系统

     容器运行时用于存储图像和容器可写层的imagefs文件系统。

imagefs是可选的。 kubelet使用cAdvisor自动发现这些文件系统。 kubelet不关心任何其他文件系统。 kubelet当前不支持任何其他类型的配置。 例如，将卷和日志存储在专用的imagefs中是不行的。

驱逐门槛

Kubelet支持指定驱逐的门槛

一个驱逐的门槛是下面的形式的

|  |
| --- |
| <eviction-signal><operator><quantity | int%>   * valid eviction-signal tokens as defined above. * valid operator tokens are < * valid quantity tokens must match the quantity representation used by Kubernetes * an eviction threshold can be expressed as a percentage if ends with % token. |

如果满足阈值条件，kubelet将采取主动行动来尝试回收与驱逐信号相关联的饥饿计算资源。

kubelet将支持软硬驱逐门槛。

例如，如果节点具有10Gi的存储器，并且如果可用存储器低于1Gi，则希望诱发驱逐，则可以将驱逐信号指定为以下任一（但不是两者）

* memory.available<10%
* memory.available<1Gi

软驱逐门槛

软起动阈值将驱逐阈值与所需的管理员指定的宽限期配对。 kubelet不采取任何措施来回收与驱逐信号相关的资源，直到超过宽限期。 如果没有提供宽限期，启动时kubelet会出错。

另外，如果已经满足软驱逐阈值，则操作者可以指定在从节点驱逐pod时使用的最大允许的pod终止宽限期。 如果指定，则kubelet将使用pod.Spec.TerminationGracePeriodSeconds中的较小值和允许的最大宽限期。 如果没有指定，kubelet会立即杀死荚，没有优雅的终止。

要配置软驱逐阈值，将支持以下标志：

--eviction-soft="": A set of eviction thresholds (e.g. memory.available<1.5Gi) that if met over a corresponding grace period would trigger a pod eviction.

--eviction-soft-grace-period="": A set of eviction grace periods (e.g. memory.available=1m30s) that correspond to how long a soft eviction threshold must hold before triggering a pod eviction.

--eviction-max-pod-grace-period="0": Maximum allowed grace period (in seconds) to use when terminating pods in response to a soft eviction threshold being met.

硬驱逐门槛

硬驱逐阈值没有宽限期，如果观察到，kubelet将立即采取行动来收回相关的饥饿资源。 如果满足硬驱逐阈值，kubelet会立即杀死荚，而不会优雅终止。

通过以下的命令来支持

--eviction-hard="": A set of eviction thresholds (e.g. memory.available<1Gi) that if met would trigger a pod eviction.

驱赶监控的间隔

kubelet将首先评估与cAdvisor家务管理相同的内部管理间隔的驱逐阈值。

在Kubernetes 1.2中，这是默认的10秒。

将监视间隔缩短到更短的窗口是一个目标。 这可能需要更改cAdvisor，以便为选定的数据指定备用内务时间间隔（https://github.com/google/cadvisor/issues/1247）

为了本建议书的目的，我们预计监控时间间隔不超过10秒才能知道阈值何时触发，但我们将努力减少延迟时间。

节点条件

kubelet将支持对应于每个逐出信号的节点条件。

如果已经满足硬驱逐阈值，或者独立于其相关宽限期，则满足软驱逐阈值，则kubelet将报告反映节点处于压力状态。

定义了与指定的逐出信号相对应的以下节点条件。

| **Node Condition** | **Eviction Signal** | **Description** |
| --- | --- | --- |
| MemoryPressure | memory.available | Available memory on the node has satisfied an eviction threshold |
| DiskPressure | nodefs.available, nodefs.inodesFree, imagefs.available, or imagefs.inodesFree | Available disk space and inodes on either the node's root filesystem or image filesystem has satisfied an eviction threshold |

kubelet将继续以--node-status-update-frequency指定的频率报告节点状态更新，默认为10秒

节点条件的振荡

如果一个节点在软驱逐阈值以上或以下振荡，但不超过其相关的宽限期，则会导致相应的节点条件在真假之间持续振荡，从而导致调度决策不佳。

为了防止这种振荡，定义了以下标志来控制在过渡到压力条件之前kubelet必须等待多长时间。

- 压力 - 压力 - 过渡期= 5m0s：kubelet必须等待的持续时间

在逐出驱逐压力条件之前。

kubelet将确保在将条件切换回为假之前指定的期间内没有观察到针对指定压力条件的逐出阈值。

驱逐场景

Memory

让我们假设Kubelet启动的时候带如下参数

--eviction-hard="memory.available<100Mi"

--eviction-soft="memory.available<300Mi"

--eviction-soft-grace-period="memory.available=30s"

kubelet将运行同步循环，通过计算（capacity-workingSet）从cAdvisor报告，查看节点上的可用内存。 如果观察到可用内存降至100Mi以下，那么kubelet将立即启动驱逐。 如果观察到可用内存低于300Mi，则会在高速缓存中内部观察到该信号时记录。 如果在下一次同步时，该条件不再满足，则该信号将清除缓存。 如果该信号被视为满足长于指定时间段，则kubelet将启动驱逐以尝试回收已满足其逐出阈值的资源。

Disk

让我们假设Kubelet启动的时候带如下参数

--eviction-hard="nodefs.available<1Gi,nodefs.inodesFree<1,imagefs.available<10Gi,imagefs.inodesFree<10"

--eviction-soft="nodefs.available<1.5Gi,nodefs.inodesFree<10,imagefs.available<20Gi,imagefs.inodesFree<100"

--eviction-soft-grace-period="nodefs.available=1m,imagefs.available=2m"

kubelet将运行一个同步循环，查看从cAdvisor报告的节点支持的分区上的可用磁盘。如果节点的主文件系统上的可用磁盘空间被观察到低于1Gi或节点的主文件系统上的可用的inode小于1，那么kubelet将立即启动驱逐。如果观察到节点的图像文件系统上的可用磁盘空间降低到10Gi以下，或者节点的主映像文件系统上的可用的inode小于10，则kubelet将立即启动。

如果节点的主文件系统上的可用磁盘空间被观察到低于1.5Gi，或者节点的主文件系统上的可用的inode小于10，或者节点的映像文件系统上的可用磁盘空间被观察到低于20Gi，或者如果节点的图像文件系统上的自由inode小于100，则会在高速缓存中内部观察到该信号时记录。如果在下一次同步时，该标准不再满足，则该信号将清除缓存。如果该信号被视为满足长于指定时间段，则kubelet将启动驱逐以尝试回收已满足其逐出阈值的资源。

驱逐Pod

如果已经满足了逐出阈值，则kubelet将启动逐出荚果的过程，直到观察到信号已经低于其定义的阈值。

驱逐顺序如下：

     对于每个监测间隔，如果已经达到驱逐阈值

     找候选荚

     失败的pod

     块直到pod在节点上终止

如果一个容器没有被终止，因为一个容器没有发生死亡（例如，进程卡在磁盘IO中），那么kubelet可以选择一个附加的pod来进行故障。 kubelet将调用在运行时界面上公开的KillPod操作。 如果返回错误，kubelet将选择一个后续的pod。

驱逐策略

kubelet将实施围绕pod质量服务类定义的默认驱逐策略。

它将针对饥饿的计算资源相对于其调度请求的最大消费者的pod来定位。它按照以下顺序对服务质量等级进行排序。

   BestEffort  消费最多的饥饿资源的pod首先失败。

    Burstable 消耗最大数量的饥饿资源相对于他们对该资源的请求的爆发荚首先被杀死。如果没有pod超出其要求，该策略将针对饥饿资源的最大消费者。

    Guaranteed相对于他们的请求消耗最大量的饥饿资源的保护荚首先被杀死。如果没有pod超出其要求，该策略将针对饥饿资源的最大消费者。

保证的pod保证永远不会被驱逐，因为另一个pod的资源消耗。也就是说，担保仅仅是建立在基础上的基础。如果一个系统守护进程（即kubelet，docker，journald等）消耗的资源比通过系统保留或kube保留分配保留的资源多，而节点只有保留的pod保留，则节点必须选择驱逐保证的pod，以保持节点的稳定性，并将意外消耗的影响限制到其他保证的pod。

基于Disk的驱赶