|  |  |
| --- | --- |
| 交底书名称 | 一种基于CPU使用率的容器驱逐策略 |
| 技术联系人姓名 | 马殿军 |
| 技术联系人电话 | 13269325833 |
| 技术联系人Email | madianjun@jd.com |

（技术联系人信息用于与外部代理沟通，发明人信息在ERP专利申请系统中填写）

注意事项：

1、代理人并不是技术专家，交底书要使代理人能看懂，尤其是完整技术方案，一定要写得全面、清楚。

2、在后续与专利代理人进行沟通时，对于代理人的疑问应认真讲解，要求补充的材料应及时补充（禁止通过私人邮箱与代理人沟通）。

3、常用检索网站：www.soopat.com（SOOPAT），http://so.baiten.cn/（佰腾），patents.google.com（谷歌专利）。

# 1. 现有技术

/\* 应记载某个应用场景或者解决某个技术问题当前所采用的技术，可以概述该技术，也可以仅给出参考文献的链接或相关专利号。

Kubernetes是一种容器集群编排、调度和管理的分布式系统，它将容器调度并运行在集群的多个节点上。当节点资源不足时，kubernetes会按照优先级驱逐节点上的容器，直到节点资源恢复正常。

# 2. 现有技术的缺点

/\* 需要指出现有技术存在的缺点，本发明也不能克服的缺点无需提供。

/\* 应根据现有技术的实现过程，有针对性地说明缺点产生的原因。

当节点资源不足时，kubernetes会驱逐节点上的容器。现有技术检查资源不足的条件是针对内存和磁盘的使用率，而没有考虑到CPU使用率。如果某些容器的CPU使用率过高，会导致当前节点的CPU资源紧张，从而影响该节点上的其他容器。

# 3. 本发明技术方案

3.1 本发明所要解决的技术问题（即发明目的）

/\* 描述本发明所要解决的技术问题，与“2.现有技术的缺点”部分指出的缺点相对应。

本发明主要解决的是根据节点和容器的CPU使用率状况，对CPU使用率过高的容器进行驱逐，以保障节点上其他容器能够正常运行。

3.2 本发明的完整技术方案的详细阐述

/\* 这是本文档最重要的部分，需要详细完整的阐述，不能光有原理，也不能仅有功能性介绍或操作说明。

/\* 在描述具体的技术方案时，必须结合附图（方法型专利按照数据流向或实现步骤抽象框图，装置型专利按照组成部件抽象框图）进行说明，每个附图都应当有对应的文字描述。如果本发明方案包含多个主题，方法与装置等，则需要分别进行描述。

1. 架构

图1中的descheduler组件用于执行驱逐逻辑。descheduler定期从apiserver上获取所有pod的信息，并过滤出已经被调度到node上但没有正常运行的pod，并将其驱逐，进而触发scheduler的对新pod的重新调度，新产生的pod将以一定概率被调度其他node上，因此可从一定概率上实现使pod不在同一个节点上反复重启。架构图如下：

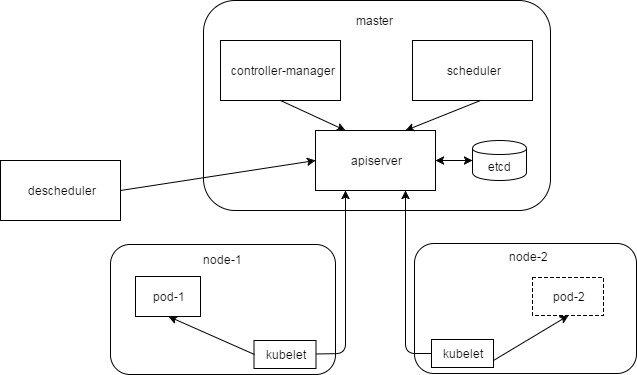


图1 descheduler与kubernetes交互架构图

详细流程描述如下：

1. pod-1在node-1上启动，但并没有成功运行，kubelet将其状态上报至apiserver。
2. descheduler定期从apiserver上获取pod列表，并检查pod状态。当发现pod-1处于“已经被调度但未成功运行”的状态时，向apiserver发出evict驱逐请求。
3. node-1上的kubelet接收到evict请求，将pod-1从node-1上删除，并将状态同步到apiserver。
4. controller-manager监听apiserver时，发现pod-1被删除，就创建一个pod-1的替代副本，称为pod-2，并将pod-2同步到apiserver。
5. scheduler监听apiserver时，发现pod-2虽然被创建了，但是没有被调度，则执行调度算法选择一个合适的node（该node可能是node-1或node-2）与pod-2绑定。
6. 如果pod-2与node-2绑定，则node-2上的kubelet从apiserver上监听到此状态变更，就在node-2上创建pod-2。

以上过程即实现了将pod-1从node-1驱逐并重新调度的过程。需要注意的是：

如果第(5)步中scheduler选择了将node-1与pod-2绑定，则pod-2会在node-1上创建，那么pod-2仍然无法正常运行。假设scheduler执行重新调度时，每个节点被选择的概率相同，那么重调度策略在一定概率上保证pod-2被调度到其他节点;集群中节点数量越多，pod-2被调度到其他节点的概率越大。

1. 基于CPU使用率的驱逐策略

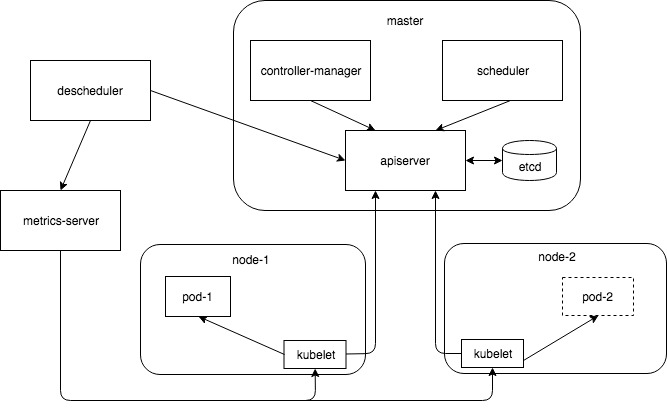


图2 descheduler通过metrics-server采集节点和容器的CPU使用率

如图2所示，在原有架构基础上加入metrics-server组件（该组件为开源项目，不是本发明提出的）。metrics-server与kubelet通信，采集节点以及节点上容器的CPU使用率；descheduler通过查询metrics-server获得CPU使用率，然后执行以下计算流程，得出应该驱逐哪些CPU使用率过高的容器：

1. 遍历所有节点，筛选出CPU可用率（CPUFree）低于预定阈值（CPUThreshold）的节点，这些节点即为异常节点。计算CPU可用率的方法：CPUFree=CPUAllocatable-CPUUsage，其中CPUAllocatable是节点的配置参数，CPUUsage来自于metrics-server,判断条件为CPUFree< CPUThreshold。
2. 针对每个异常节点，将节点上的所有容器按优先级排序，得出一个容器的驱逐顺序列表。排序算法如下：
3. 计算每个容器的CPUUtilization: CPUUtilization=CPUUsage/CPUReqeust(CPUUsage来自于metrics-server，CPURequest是容器的配置参数)
4. 获取每个容器的QoS，QoS值分为Guaranteed,Burstable,BestEffort三种（QoS的概念由Kubernetes定义，不是本发明提出的）
5. 结合CPUUtilization和QoS共同判定容器的驱逐顺序：先比较QoS，如果QoS相同，则比较CPUUtilization。

QoS优先级由高到低为：Guaranteed，Burstable，BestEffort;QoS优先级越高，则容器越不应该被驱逐，所以驱逐顺序为BestEffort，Burstable，Guaranteed。

按CPUUtilization比较，则是CPUUtilization值越大的容器越应该被驱逐。

以上排序算法得出的容器列表，就是按照容器“被驱逐的紧要程度”作的排序：越是排在前面的容器，越是异常容器，则越应该被驱逐。

1. 遍历容器的驱逐顺序列表，将容器逐个驱逐；每驱逐一个容器时，重新计算节点的CPU可用率,即：新的节点CPUFree=旧的节点CPUFree+被驱逐容器的CPUUsage，如果新的节点CPU可用率超过预定阈值，则说明节点的CPU资源已经充足，那么停止驱逐。

3.3 本发明希望保护的技术创新点

/\* 指出技术方案中希望保护的技术关键点，并概括说明该关键点的技术原理。

本发明的技术创新点是根据节点和容器的CPU使用率状况，设计出一个新的驱逐策略，以避免节点CPU资源不足时影响正常容器的运行。

3.4 针对3.3中的技术方案，是否还有别的替代方案同样能完成发明目的？

/\* 替代方案可以是完整技术方案的替代，也可以是部分结构或者步骤的替代。

3.5交底书中技术术语的名词解释

/\* 记载交底书中出现的专业技术术语、缩写、外文的解释。

node：集群中的节点，对应一台物理机。

kubernetes：由google开源的容器集群编排和管理系统。

master：kubernetes的控制节点，包含apiserver, controller-manager, scheduler三个组件。

etcd：一种分布kv数据库，为kubernetes提供数据持久化服务。

apiserver：为kubernetes其他组件提供api访问，它直接读写etcd。

controller-manager：维护pod副本数量，使之与用户期望的数量一致。

scheduler：调度pod，使pod与合适的node绑定。

kubelet：管理和监控本节点上的pod。

pod：是容器的一种抽象表示，它可以包含一个或多个容器实例。Kubernetes以pod作为最小的管理和调度单位。

evict：驱逐，即将pod从node上删除。

descheduler：实现驱逐策略的组件。

metrics-server：采集节点及节点上容器的资源使用率，提供给其他组件使用。