



面向5G边缘计算的Kubernetes资源调度策略

孔德瑾, 姚晓玲

(太原理工大学 财经学院, 太原 030024)

摘要: 容器云是5G边缘计算的重要支撑技术,5G的大带宽、低时延和大连接三大特性给边缘计算带来较大的资源压力,容器云编排器Kubernetes仅采集Node剩余CPU和内存两大资源指标,并运用统一的权重值计算Node优先级作为调度依据,该机制无法适应边缘计算场景下精细化的资源调度需求。面向5G边缘计算的资源调度场景,通过扩展Kubernetes资源调度评价指标,并增加带宽、磁盘两种评价指标进行节点的过滤和选择,提出一种基于资源利用率进行指标权重自学习的调度机制WSLB。根据运行过程中的资源利用率动态计算该应用的资源权重集合,使其能够随着应用流量的大小进行自适应动态调整,利用动态学习得到的资源权重集合来计算候选Node的优先级,并选择优先级最高的Node进行部署。实验结果表明,与Kubernetes原生调度策略相比,WSLB考虑了边缘应用的带宽、磁盘需求,避免了将应用部署到带宽、磁盘资源已饱和的Node,在大负荷与异构请求场景下可使集群资源的均衡度提升10%,资源综合利用率提升2%。

关键词: 5G网络;边缘计算;资源调度;权重自学习;Kubernetes调度策略

开放科学(资源服务)标志码(OSID):



中文引用格式:孔德瑾,姚晓玲.面向5G边缘计算的Kubernetes资源调度策略[J].计算机工程,2021,47(2):32-38.

英文引用格式:KONG Dejin, YAO Xiaoling. Kubernetes resource scheduling strategy for 5G edge computing [J]. Computer Engineering, 2021, 47(2): 32-38.

Kubernetes Resource Scheduling Strategy for 5G Edge Computing

KONG Dejin, YAO Xiaoling

(School of Finance and Economics, Taiyuan University of Technology, Taiyuan 030024, China)

[Abstract] Container cloud is a key supporting technology for 5G edge computing, but edge computing faces great resource pressure imposed by the large bandwidth, low latency, and massive connections of 5G. The scheduler of container cloud, Kubernetes, only collects remaining CPU and memory of nodes, and uses a fixed weight to calculate the priority of nodes as the basis of scheduling. The mechanism cannot meet the demand for refined resource scheduling in edge computing scenarios. To address the resource scheduling needs of 5G edge computing, this paper expands the resource scheduling evaluation indicators of Kubernetes, adds bandwidth and disk evaluation indicators to filter and select nodes, and on this basis proposes a scheduling mechanism named WSLB, which realizes weight self-learning based on resource occupancy. WSLB dynamically calculates the resource weight set of the application according to its resource utilization during the running process to enable the weight set to dynamically and adaptively adjust itself based on the size of application traffic. The resource weight set obtained from dynamic learning is used to calculate the priority of candidate nodes, and the node with the highest priority is selected for deployment. Experimental results show that compared with the native scheduling strategy of Kubernetes, WSLB fully considers the bandwidth and disk requirements of edge applications, and avoids deploying applications to nodes where resources are all occupied. In the heavy load and heterogeneous request scenario, the balance of cluster resources under the WSLB mechanism is increased by 10%, the comprehensive utilization rate of resources increased by 2%.

[Key words] 5G network; edge computing; resource scheduling; weight self-learning; Kubernetes scheduling strategy

DOI: 10.19678/j.issn.1000-3428.0058047

0 概述

边缘计算^[1-2]作为云计算的演进,将应用程序托

管方式从集中式数据中心下沉到网络边缘^[3],在靠近用户的网络边缘提供存储与计算能力,从而降低

基金项目:国家自然科学基金(11771321)。

作者简介:孔德瑾(1965—),男,副教授,主研方向为容器云、边缘计算;姚晓玲,副教授。

收稿日期:2020-04-13 修回日期:2020-05-18 E-mail: kongdejin7773@126.com

用户访问延时,是5G网络实现低延迟和提升带宽速率的关键技术之一。容器云^[4-5]在传统云环境中得到广泛应用,是边缘计算的重要支撑技术^[6],然而边缘计算具有与传统云计算环境不同的特征,在业务场景方面,AR/VR、4K/8K视频等应用部署于边缘会给边缘云带来更大的流量压力,此外随着物联网的兴起,边缘计算Node也面临着海量物联网终端超大连接数的压力,在流量特征方面,移动用户特有的移动性,可能会导致边缘计算流量随着时间、地点而变化,进而触发边缘云内容器的需求变化,在网络环境方面,受限于部署环境,网络边缘存储、计算、带宽等资源非常稀缺。因此,在5G边缘计算场景下,容器云的资源调度机制将面临更大挑战,若将应用部署在不合适的Node上,会增大扩缩容频次,影响应用的性能并造成资源的浪费。

Kubernetes(K8s)是应用最为普遍的容器云编排管理系统^[7],广泛应用于边缘计算场景^[8-9],其资源调度策略首先根据用户声明的最小资源需求过滤掉不符合要求的Node,再将Node的剩余CPU利用率、剩余内存利用率作为评价指标,利用加权求和方式对候选Node评分,选择评分最高的Node进行部署。这种调度策略有以下弊端:没有考虑应用对于带宽、磁盘等资源的需求,无法对边缘计算场景下带宽、磁盘资源倾向型应用进行合理的调度;对于需求各异的应用,Kubernetes采用固定的求和权重对Node评分,不能满足应用的个性化资源需求。上述不足导致Kubernetes原生资源调度机制难以适用于5G时代的边缘计算场景。

本文针对Kubernetes原生资源调度策略的不足,提出一种基于权重自学习的Kubernetes容器调度机制WSLB。增加带宽、磁盘两类指标,在Node过滤阶段和优先级计算阶段,将4类剩余资源均作为评价指标,改进了K8s原生调度策略无法满足带宽型、磁盘型Pod应用调度需求的不足。通过部署于Node的监控代理,采集Pod动态资源利用率,根据Pod运行过程中的资源占用率计算并调整Pod的资源权重集合,资源权重集合衡量该Pod的资源倾向,当创建Pod时,WSLB根据Pod的最小资源需求过滤不符合要求的Node,利用资源权重集合对候选Node的各类剩余资源利用率加权求和,得到候选Node的优先级,选择优先级最高的Node进行部署。WSLB将某项资源需求大的应用调度到该资源剩余较多的Node,避免Node由于某个资源被过度分配从而导致整个Node无法再分配新容器的问题。

1 Kubernetes调度策略

在Kubernetes中,Node是执行任务的载体,以物理服务器或虚拟机的形式存在。Pod是一个应用实例,为Kubernetes调度的最小单位,用户在创建Pod时,Kubernetes采用的调度策略如下^[10-11]:

1)第一阶段进行Node过滤,过滤掉不满足Pod最小资源需求的Node,支持的过滤策略包括端口冲突、CPU、内存容量检测和服务占用等。

2)第二阶段称为Node优先级计算,Kubernetes将剩余的Node根据CPU和内存空闲资源率进行综合评分,按照评分的大小排序选出最合适的Node。最后将Pod绑定到选出的目标Node,调度器完成容器应用的部署^[12]。目前支持的优先级评分函数主要有以下3种:

(1)LeastRequestedPriority函数:由Node空闲资源与Node资源总量的比值来决定Node评分,空闲资源越多,评分越高,CPU和内存具有相同权重值,该函数是应用最多的评分函数。

(2)BalancedResourceAllocation函数:CPU和内存使用率越接近的Node评分越高,该函数用于简单地调节在部署Pod应用后各Node的CPU和内存资源利用率的均衡性。

(3)SelectorSpreadPriority函数:对于属于同一个Service、Replication Controller的Pod副本,尽量调度到不同的Node上。

此外,调度函数还包括NodePreferAvoidPodsPriority、NodeAffinityPriority、TaintTolerationPriority等一系列特殊的评分函数。

图1为Kubernetes默认资源调度策略示例,令资源集合 $R=\{\text{CPU, 内存, 带宽, 磁盘}\}$,待部署Pod的资源最小需求量 r 表示部署该Pod所需的上述4类资源的最小值, $r=\{4.0, 0.5, 0.1, 0.01\}$,Node的资源总量指该Node的CPU、内存、带宽、磁盘的总量,Node1~Node4的资源总量均为 $\{16, 32, 1, 1\}$,每个Node剩余资源如表1所示。

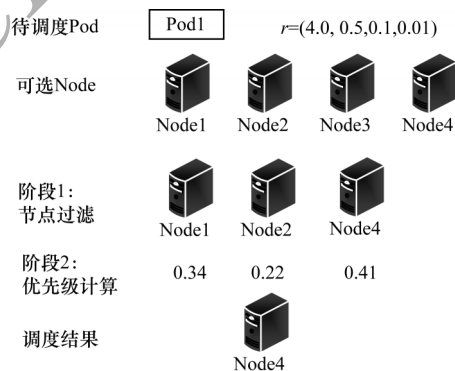


图1 资源调度策略示例

Fig.1 Example of resource scheduling strategy

表1 可选Node的剩余资源

Table 1 Remaining resources of optional Node

Node	CPU(core)	内存/GB	带宽/(Gb·s ⁻¹)	磁盘/GB
Node1	10.0	2.0	0.5	0.3
Node2	5.0	4.0	0.9	0.2
Node3	2.0	25.0	0.7	0.6
Node4	5.0	16.0	0.1	0.2

对于待创建的Pod1,第1阶段过滤掉剩余资源不满足需求的Node3,第2阶段采用LeastRequestedPriority评分函数计算各Node的优先级。 LO_{cpu} 、 LO_{memory} 分别表示Node剩余CPU、内存的比例,两者权重均为0.5。优先级 f 的计算公式为:

$$f = 0.5 \times LO_{cpu} + 0.5 \times LO_{memory}$$

其中, f 越高表示该Node的资源配置越适合部署该Pod,经过计算,最终选择Node4进行部署。

Node4并非最佳选择,主要体现在以下两点:

1)Kubernetes的调度算法未考虑带宽和磁盘资源,Node4部署该Pod后,剩余带宽资源为0,当该Node上任意Pod遇到突发流量时,不仅该Pod的最小带宽需求无法得到满足,所有Pod的性能都将受到影响。在网络边缘,并非只有CPU、内存是稀缺资源,带宽、磁盘同样非常稀缺,如果调度策略仅考虑CPU、内存,将无法满足视频等大带宽应用及数据库等磁盘需求较大应用的调度需求,影响此类应用的性能。

2)即使仅考虑CPU、内存,Kubernetes的默认调度算法仍有弊端:部署该Pod后,Node4将剩余1个CPU和15.5 GB内存,资源使用得过度倾斜,导致Node剩余资源不均衡,使该Node无法再创建更多的容器,从而造成过剩资源的浪费。LeastRequestedPriority函数对于任何应用,CPU、内存权重都相同,完全没有考虑应用对于某种资源的需求倾向。该Pod是一个CPU密集型应用,内存需求较小,调度策略应该将其部署在剩余CPU较为丰富而内存余量较少的Node,这样主要有以下优势:(1)为应用留有较多的冗余资源,Kubernetes根据用户声明的最小资源过滤Node,实际占有资源可能超过最小资源;(2)更有利于Node剩余资源的均衡,均衡的资源分布有助于实现更高的资源利用率。

2 基于权重自学习的调度策略

对于容器资源调度的研究主要有两个方向:

1)增加评价指标,从而适应物联网、智慧城市等场景下的调度需求,如文献[13-14]提出一种面向智慧城市容器云的网络性能感知调度系统,增加了RTT评价指标,在Node过滤阶段排除RTT不满足需求的Node,文献[15]在Mesos调度策略中增加了容器个数。考虑到评价指标应有效代表资源的使用情况,且指标的获取不应给系统带来过多开销,本文选择CPU、内存、带宽、磁盘使用情况作为评价指标。

2)优化Node优先级评分函数中的权重计算方法,如文献[15]采用层次分析法^[16-17]计算权重集合。文献[18]采用模糊层次分析法自动建模求解容器应用多维资源权重参数。利用层次分析法计算权重要求用户掌握应用各类资源的重要程度,但是,很多用

户对于自己的应用需要占用多少资源,每种资源的重要程度并不清楚。基于以上考虑,本文将Pod运行过程中资源使用自动生成的权重,用于Node优先级的计算。

在Pod运行过程中,监测模块将周期性地采样该Pod 4类资源的占用率,求得监控窗口内每类资源占用率平均值,进而计算全局资源占用率,根据全局资源占用率计算每类资源的权重,称为Pod专用权重。使用同样镜像的所有Pod各类资源权重平均值作为该镜像的专用权重。

2.1 资源利用率采集方案

本文采集3类信息:1)Node资源总量,对所有Node的各类资源总量求和得到全局资源总量,进而计算出Node的各类资源在全局资源中所占份额;2)Node当前资源利用率,进而求得剩余资源比例;3)Pod资源在当前Node资源中所占份额。

采集方案如图2所示,在每个Node中部署监控代理Proxy,用于采集上述3类信息。Pod资源占用率数据库用于存储每个Pod监控窗口内的CPU、内存、带宽、磁盘占用率。Node全局资源份额数据库用于存储每个Node的资源总量和监控窗口内的资源利用率。采集控制器用于接收资源调度模块的监控指令,对于第1)类、第2)类信息,指令中包括Node标识,对于第3)类信息,指令中包括被监控的Pod标识,采集控制器通过Pod标识获取部署该Pod的Node标识,通过Node标识,采集控制器可查询得到Proxy的IP、端口,进而向Proxy发送监控指令。Proxy采集到所需要的监控信息后存于对应的数据库。

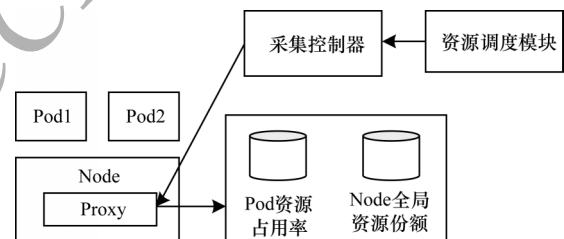


图2 资源利用率采集方案

Fig.2 Resource utilization rate collection scheme

2.2 资源调度流程

调度模块维护3类权重集合:1)Pod专用权重,用于Pod扩容时选择最优Node,在测量窗口,统计Pod及其副本占用的各类资源占整个集群资源的比例,即全局占用率,根据各类资源的全局占用率计算Pod专用权重;2)镜像专用权重,是使用该镜像的所有Pod各资源权重的平均值,如果待创建Pod的镜像有使用记录,则使用镜像专用权重选择最优Node;3)Kubernetes默认权重,即所有资源权重相等。

本文采用两级调度机制,如图3所示。

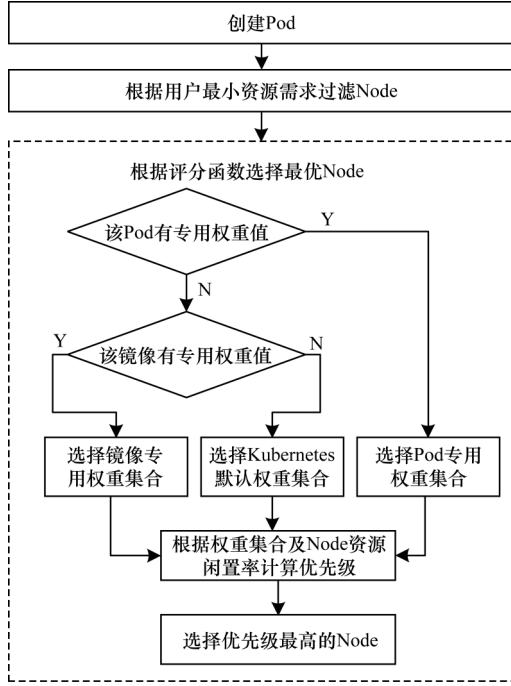


图3 改进的资源调度流程

Fig.3 Improved resource scheduling procedure

调度过程分两个阶段,第一阶段根据用户最小资源需求过滤掉不满足需求的Node,第二阶段选择权重集合对候选Node的各类资源的闲置率加权求和,得到候选Node的优先级。权重集合选择策略如下:

- 1) 如果待调度的Pod有专用权重值,则根据Pod专用权重值、资源剩余情况选择最优Pod。
- 2) 如果待调度Pod没有专用权重值,但镜像有专用权重值,则采用镜像专用权重值。
- 3) 其他情况采用Kubernetes的默认权重值,即所有资源权重值相等。

2.3 Node优先级计算方法

符号定义及Node优先级计算方法如下:

1) 符号定义

假设Kubernetes集群中有 k 台不同规格的Node, $N=\{1, 2, \dots, k\}$; 每个Node上有 m 种资源, $R=\{1, 2, \dots, m\}$, 某Pod由 j 个容器构成, $P=\{1, 2, \dots, j\}$, 使用镜像 i 的所有Pod集合为 $I=\{1, 2, \dots, h\}$ 。

2) Pod p 各类资源全局占用率计算方法

本文根据Pod p 的CPU、内存、带宽、磁盘4类资源全局占用率计算Pod p 的专用权重,全局占用率是指Pod p 某项资源的占用额度占集群中该资源总额的比重,衡量了Pod p 对该资源的需求。通过监控代理可以采集到Node i 各类资源总量和Node i 上容器的本地资源占用率。为计算Pod p 的资源 r 全局占用率,需要首先计算Node i 的资源 r 全局份额 $\text{nodeGlobalShare}(i, r)$ 。将 $\text{nodeGlobalShare}(i, r)$ 与容器 d 的资源 r 本地占用率相乘,即可得到容器 d 的资源 r 全局占用率 $\text{containerLocalRU}(i, d, r)$ 。Pod p 由多个容器组成,将Pod p 中的所有容器资源 r 的全

局占用率求和,即可得到Pod p 资源 r 的全局占用率。

(1) 计算Node i 的各类资源全局份额

设Node i 上资源 r 的总量为 $\text{nodeTotal}(i, r)$, 则集群资源 r 总量 $\text{clusterTotal}(r) = \sum_{i \in N} \text{nodeTotal}(i, r)$ 。

Node i 上资源 r 的全局份额为:

$$\text{nodeGlobalShare}(i, r) = \text{nodeTotal}(i, r) / \text{clusterTotal}(r) \quad (1)$$

(2) 计算容器 d 的各类资源全局占用率

$\text{containerLocalRU}(i, d, r)$ 表示部署在Node i 的容器 d 及资源 r 的占用率,其全局资源占用率为:

$$\text{containerGlobalRU}(i, d, r) = \text{containerLocalRU}(i, d, r) \times \text{nodeGlobalShare}(i, r) \quad (2)$$

(3) 计算Pod p 的各类资源全局占用率

Pod p 由 j 个容器组成, $\text{podGlobalRU}(p, r)$ 表示Pod p 资源 r 的全局占用率。

$$\text{podGlobalRU}(p, r) = \sum_{d \in P} \text{containerGlobalRU}(i, d, r) \quad (3)$$

3) Pod p 专用权重计算方法

$\text{weightPod}(p) = \{w(p, 1), w(p, 2), \dots, w(p, m)\}$ 表示Pod p 专用权重集合。 weightPod 满足 $\sum_{r \in R} w(p, r) = 1$,

其中 $w(p, r)$ 是在计算候选Node优先级时资源 r 的权重,如式(4)所示,通过计算Pod p 的资源 r 的全局占用率与该Pod中所有资源全局占用率之和的比值得到 $w(p, r)$ 。例如Pod p 的专用权重集合为 $\{1\%, 2\%, 3\%, 0.01\%\}$,表示该Pod每消耗集群1%的CPU,将伴随消耗2%的内存、3%的带宽和0.01%的磁盘,那么在对候选节点计算优先级评分时,内存的优先级应是CPU的2倍,通过式(4)即可计算得到归一化权重:

$$w(p, r) = \text{podGlobalRU}(p, r) / \sum_{r' \in R} \text{podGlobalRU}(p, r') \quad (4)$$

4) 镜像 i 专用权重计算方法

$\text{weightImage}(i) = \{w(i, 1), w(i, 2), \dots, w(i, m)\}$ 表示镜像 i 的专用权重集合,其中 $w(i, r)$ 表示镜像 i 资源 r 的专用权重。当需要创建新的Pod时,数据库中尚未创建该Pod的专用权重,此时,参考该Pod使用的镜像 i 的专用权重。 $\text{weightImage}(i)$ 为所有使用该镜像的Pod权重平均值, h 为使用镜像 i 的Pod总数。

$$w(i, r) = \sum_{p \in I} w(p, r) / h \quad (5)$$

5) 优先级评分函数

当创建Pod时,通过优先级评分函数选择剩余资源分布最优的Node进行部署,优先级越高,表示该候选节点资源分布越适合部署该Pod。权重集合的选择顺序如图3所示,优先级评分函数为候选Node闲置率与权重集合的加权求和。Pod p 为待部署的Pod, $\text{priority}(p, i)$ 表示Node i 的优先级评分, $\text{nodeFree}(i, r)$ 表示Node i 资源 r 的闲置率,是Node i 剩余的資源 r 与Node i 资源 r 总量的比值。以Pod p

有专用权重为例, $\text{priority}(p, i)$ 的计算方法为:

$$\text{priority}(p, i) = \sum_{r \in R} \text{nodeFree}(i, r) \times \text{weightPod}(p, r) \quad (6)$$

6) 实例

集群内有2个Node,资源总量情况如表2所示,根据式(1)计算得到4种资源的全局份额。

表2 集群资源情况

Table 2 Cluster resource situation

资源名称	资源总量		资源名称	全局份额/%	
	Node1	Node2		Node1	Node2
CPU/(core)	8	8	CPU	50	50
内存/GB	16	32	内存	33	67
带宽/(Gb·s ⁻¹)	10	10	带宽	50	50
磁盘/GB	10	1	磁盘	91	9

Pod p 有1个副本, Pod p 部署在Node1上,即容器1, Pod p 的副本部署在Node2上,即容器2。通过测量得到资源利用率 containerLocalRU , 如表3所示, 根据式(2)可以得到2个容器4种资源的全局资源占用率。

表3 容器资源占用率

Table 3 Containers resource occupation rate %

资源名称	容器本地资源占用率		容器全局资源占用率		Pod全局资源占用率
	容器1	容器2	容器1	容器2	
CPU	30.00	40.00	15	20	35
内存	3.00	4.00	1	3	4
带宽	80.00	70.00	40	35	75
磁盘	0.01	0.15	1	1	2

根据式(4), Pod p 的CPU权重值 $w(p, \text{cpu}) = \frac{\text{podGlobalRU}(p, \text{cpu})}{\sum_{r \in R} \text{podGlobalRU}(p, r)} = 0.30$, 同理可以计算得到内存、带宽、磁盘的权重值, $\text{weightPod}(p) = \{0.30, 0.03, 0.65, 0.02\}$ 。

经测量Node1、Node2各类资源闲置率如表4所示。此时Pod p 需要再增加一个副本, 根据式(6)可以得到Node1和Node2的优先级评分分别为0.40、0.49。根据优先级评分, Pod p 的副本应部署在Node2上, 很明显Node2是更优选择, 由表4可知, Pod p 是带宽型应用, 大约需要占用3.5 Gb/s带宽, 如果部署到Node1, 带宽资源仅剩余0.5 Gb/s, 难以应对突发流量。

表4 Node资源闲置率

Table 4 Node resources idle rate %

资源名称	Node1资源闲置率	Node2资源闲置率
CPU	40	30
内存	20	22
带宽	40	60
磁盘	60	20

2.4 评价指标

本文评价指标主要有以下3种:

1) 集群资源失衡度

对于Node i , 各类资源利用率的标准差可以反映该Node资源均衡状况。假设集群中有 k 台不同规格的Node, 每个Node上有 m 种资源。 $U(i, r)$ 表示Node i 资源 r 的利用率, Node i 资源利用率标准差表示为:

$$\text{STD}(i) = \sqrt{\sum_{r \in R} (U(i, r) - U_{\text{AVR}}(i))^2}$$

$$U_{\text{AVR}} = \sum_{r \in R} U(i, r) / m$$

定义集群资源失衡度 $\text{STD}_{\text{AVE}} = \sum_{i \in N} \text{STD}(i) / K$, STD_{AVE}

值越小, 代表集群中各类资源利用越均衡, 资源倾斜的概率越小, 从而使得集群能够部署更多的容器。

2) 资源综合利用率

Node i 的资源综合利用率为该Node各类资源利用率的均值, 定义集群资源综合利用率为所有Node资源综合利用率的平均值, 衡量集群的资源是否得到充分利用。

3) 调度合理率

如果将Pod p 调度到Node i 后, Node i 的各类资源未触及资源上限, 那么此次调度结果是合理的。调度合理率是指对于请求集合 R 调度合理的比例。

3 实验

3.1 实验环境

为验证本文提出的容器调度算法, 基于墨尔本大学开源的云计算仿真框架 ContainerCloudSim^[19-20] 进行仿真, 利用该软件模拟一个包含30个Node的Kubernetes边缘云, 每个Node的资源信息如表5所示。

表5 Node资源信息

Table 5 Node resource information

CPU(core)	内存/MB	带宽/(Gb·s ⁻¹)	磁盘/GB
18	16 384	10	1 000

本文基于边缘应用资源需求多样性、差异化的特点, 按照CPU倾向型、内存倾向型、存储倾向型、带宽倾向型以及标准无倾向型5类应用资源请求构建了Pod资源需求, 表6为部分Pod资源需求, 表7展示了WSLB对于Pod自动学习到的权重。

表6 Pod资源需求

Table 6 Pod resource requirement

序号	CPU(core)	内存/MB	带宽/(Gb·s ⁻¹)	磁盘/GB
1	0.4	300	0.10	2.400
2	0.1	100	0.01	2.000
3	0.5	102	0.60	0.345
4	0.8	204	0.50	0.300
5	0.2	512	0.05	0.100
⋮	⋮	⋮	⋮	⋮

表7 基于自学习机制得到的权重

Table 7 Weights based on self-learning mechanism

序号	CPU	内存	磁盘	带宽
1	0.638 90	0.143 75	0.138 00	0.079 35
2	0.712 12	0.133 52	0.085 45	0.689 00
3	0.658 86	0.174 41	0.097 67	0.069 06
4	0.654 22	0.167 27	0.107 05	0.071 46
5	0.651 32	0.162 83	0.112 90	0.072 95
⋮	⋮	⋮	⋮	⋮

3.2 实验结果与分析

在上述 Kubernetes 边缘云集群中,分别采用 K8s 原生调度算法与本文提出的 WSLB 自学习调度算法,从集群资源均衡度、综合资源利用率和调度合理率3个角度来对比两种算法。

1) 集群资源失衡度

集群资源失衡度变化曲线如图4所示,当 Pod 数量小于 6 000 时,集群负荷较低,WSLB 机制与 K8s 原生调度机制的资源失衡度相差并不大,甚至有可能在某些特定的 Pod 需求下,略低于 K8s 原生调度机制,但当 Pod 数量大于 6 000 时,随着容器资源请求数量继续上升,WSLB 自学习得到的权重开始产生作用,WSLB 的资源失衡度明显优于 K8s 原生调度算法。由于 WSLB 考虑了 4 种资源的权重情况,降低了集群中 Node 出现某一资源用尽而其余资源大量剩余的可能性,从而使得集群的资源使用失衡度整体下降约 10%,尤其是当 Pod 数量大于 7 000,集群资源饱和后,WSLB 失衡度平均下降 26.2%,体现了 WSLB 在集群资源饱和情况下可以有效调节集群资源平衡度。

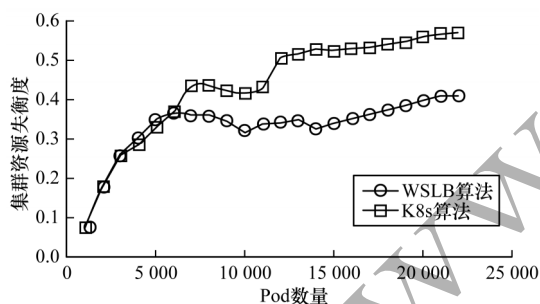


图4 集群资源失衡度变化曲线

Fig.4 Cluster resource lopsidedness change curve

2) 资源综合利用率

资源综合利用率变化曲线如图5所示。图5中截取了 Pod 数量大于 6 000 的部分,当 Pod 数量小于 6 000 时,集群负荷较小,Pod 调度请求均可满足,两种算法资源综合利用率没有差异。当 Pod 数量大于 6 000 时,集群负荷较大,WSLB 的资源综合利用率总体优于 K8s 原生调度算法,相比 K8s 平均提升 1.6%。主要原因如下:随着 Pod 数量的增加,K8s 算法失衡度上升,集群中部分 Node 出现资源倾斜,导

致部分 Pod 调度请求无法得到满足;K8s 原生算法没有考虑带宽和存储资源,致使部分 Node 带宽、存储资源饱和,这部分 Node 无法充分满足应用的带宽、存储需求。

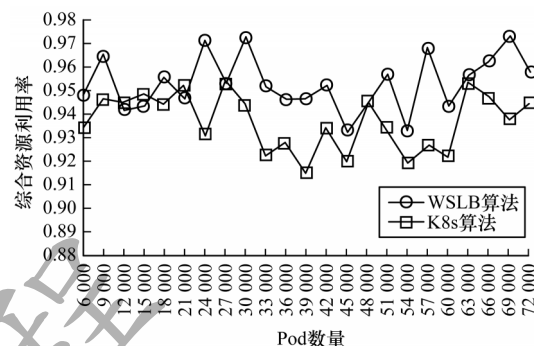


图5 集群资源综合利用率

Fig.5 Cluster resource comprehensive utilization rate

3) 调度合理率

如图6所示,WSLB 的调度合理率始终为 1,当 Pod 数量大于 1 500 时,WSLB 调度合理率较 K8s 平均提升 9.3%。原因在于 WSLB 充分考虑了 Pod 的带宽、磁盘需求,避免了 Node 带宽、磁盘资源饱和情况的出现,当 Pod 数量大于 3 000 时,K8s 原生调度策略开始出现调度不合理的情况,这是因为 K8s 原生算法不考虑 Pod 的带宽、磁盘需求,将部分 Pod 调度到带宽或磁盘资源饱和的 Node,该 Node 上所有 Pod 的性能将受到影响。

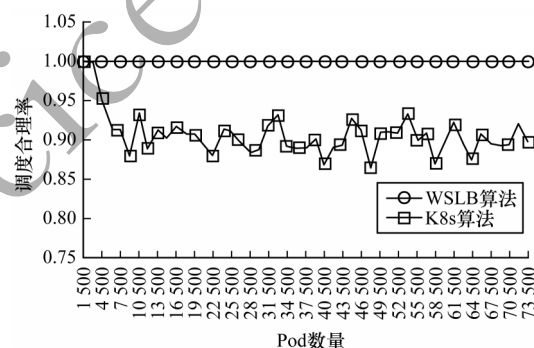


图6 调度合理率

Fig.6 Scheduling reasonable rate

图7所示为当 Pod 数量为 7 500 时,K8s 原生调度算法和 WSLB 算法的 Node 带宽利用率,K8s 原生调度算法由于在调度时没有考虑带宽的使用情况,使得各 Node 的带宽利用率分布极为不均衡,标准差达到了 0.496 8,且有 5 个 Node 的带宽利用率已远超 1,这表明这些 Node 的带宽资源已经饱和,饱和率达到 16%。在 5G MEC 的业务场景之下,意味着这些 Node 网络已严重拥堵,5G MEC 带来的低延时优势已经不再具备。而在 WSLB 调度机制下,大部分 Node 带宽利用率较为均匀地分布于 0.4~0.8 之间,标准差为 0.137 4,相比于 Kubernetes 原生调度算法,下降了 72%,保障了整个集群的带宽使用率。

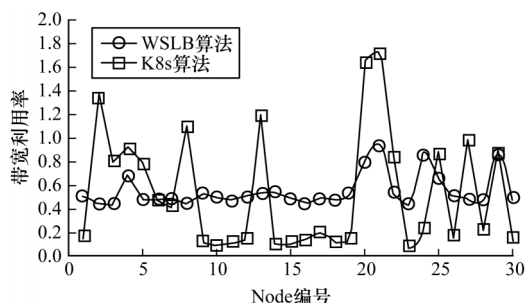


图7 集群带宽利用率

Fig.7 Cluster bandwidth utilization rate

4 结束语

容器云是5G边缘计算的重要支撑技术,网络边缘的资源较为稀缺,而边缘应用与传统应用的差异性给容器云的调度机制带来了新的需求。Kubernetes仅考虑CPU、内存,且对需求各异的应用采用相同权重的资源调度策略,无法适用于边缘云的调度需求。针对Kubernetes资源调度策略的不足,本文增加了带宽、磁盘利用率两项指标,以满足边缘计算场景下带宽、磁盘需求较大应用调度需求,同时提出一种根据Pod运行过程中资源占用情况自动计算Pod专用权重、镜像专用权重的方法,对不同资源倾向的应用采用不同的权重,以满足大规模异构化的容器资源申请需求。在ContainerCloudSim仿真平台上的实验结果表明,与Kubernetes原生调度策略相比,基于权重自学习的调度策略具有更高的资源利用率。将Pod调度到合适的Node上只是资源调度的开始,后续研究将考虑在Node上分配各Pod之间的资源,从而实现公平、高效的资源调度,进一步提升系统的资源利用率。

参考文献

- [1] SHI Weisong, ZHANG Xingzhou, WANG Yifan, et al. Edge computing: state-of-the-art and future directions[J]. Journal of Computer Research and Development, 2019, 56(1): 73-93. (in Chinese)
施巍松, 张星洲, 王一帆, 等. 边缘计算: 现状与展望[J]. 计算机研究与发展, 2019, 56(1): 73-93.
- [2] SHI Weisong, SUN Hui, CAO Jie, et al. Edge computing—an emerging computing model for the Internet of everything era[J]. Journal of Computer Research and Development, 2017, 54(5): 907-924. (in Chinese)
施巍松, 孙辉, 曹杰, 等. 边缘计算: 万物互联时代新型计算模型[J]. 计算机研究与发展, 2017, 54(5): 907-924.
- [3] YIN Dongming. MEC constructing the edge cloud for 5G network architecture[J]. Telecommunications Network Technology, 2016(11): 43-46. (in Chinese)
尹东明. MEC构建面向5G网络构架的边缘云[J]. 电信网技术, 2016(11): 43-46.
- [4] BURNS B, GRANT B, OPPENHEIMER D, et al. Borg, omega, and Kubernetes[J]. Communications of the ACM, 2016, 59(5): 50-57.
- [5] LI Shancang, XU Lida, ZHAO Shanshan. 5G Internet of things: a survey[J]. Journal of Industrial Information Integration, 2018(10): 1-9.
- [6] CHA S J. Multi-access edge computing in 5G network slicing: opportunities and challenges[C]//Proceedings of the 10th International Conference on Information and Communication Technology Convergence. Jeju Island, Korea: Institute of Electrical and Electronics Engineers Inc., 2019: 1-10.
- [7] PAHL C. Containerization and the PaaS cloud[J]. IEEE Cloud Computing, 2015, 2(3): 24-31.
- [8] MAKRIS N. On minimizing service access latency: employing MEC on the front haul of heterogeneous 5G architectures[C]//Proceedings of IEEE International Symposium on Local and Metropolitan Area Networks. Paris, France: [s. n.], 2019: 324-336.
- [9] MORABITO R. Virtualization on Internet of things edge devices with container technologies: a performance evaluation[J]. IEEE Access, 2017, 5: 2169-3536.
- [10] WANG Wei, LI Baochun, LIANG Ben. Dominant resource fairness in cloud computing systems with heterogeneous servers[C]//Proceedings of IEEE INFOCOM'13. Turin, Italy: Institute of Electrical and Electronics Engineers, 2013: 587-598.
- [11] CHUN B. Kubernetes enhancement for 5G NFV infrastructure[C]//Proceedings of 2019 International Conference on Information and Communication Technology Convergence. Jeju Island, Korea: [s. n.], 2019: 325-339.
- [12] MEDEL V, TOLOSANA-CALASANZ R, BANARES J Á, et al. Characterising resource management performance in Kubernetes[J]. Computers Electrical Engineering, 2018, 68: 286-297.
- [13] SANTOS J, WAUTERS T, VOLCKAERT B, et al. Resource provisioning in fog computing: from theory to practice[J]. Sensors, 2019, 19(10): 2238-2262.
- [14] SANTOS J. Towards network-aware resource provisioning in Kubernetes for fog computing applications[C]//Proceedings of IEEE Conference on Network Softwarization. Paris, France: IEEE Press, 2019: 457-468.
- [15] CUI Guangzhang, ZHU Zhixiang. Improved container cloud resource scheduling policy[J]. Computer and Digital Engineering, 2017, 45(10): 1931-1936. (in Chinese)
崔广章, 朱志祥. 容器云资源调度策略的改进[J]. 计算机与数字工程, 2017, 45(10): 1931-1936.
- [16] SAATY T L. Decision making with the analytic hierarchy process[J]. International Journal of Services Sciences, 2008, 1(1): 83-98.
- [17] VAIDYA O S, KUMAR S. Analytic hierarchy process: an overview of applications[J]. European Journal of Operational Research, 2006, 169(1): 1-29.
- [18] SAATY T L, DECISION H T M A. The analytic hierarchy process[J]. European Journal of Operational Research, 1990, 48(2): 9-26.
- [19] PIRAGHAJ S F, DASTJERTI A V, CALHEIROS R N, et al. ContainerCloudSim: an environment for modeling and simulation of containers in cloud data centers[J]. Software: Practice and Experience, 2017, 47(4): 505-521.
- [20] CALHEIROS R N, RANJAN R, BELOGLAZOV A, et al. CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms[J]. Software, 2011, 41(1): 23-50.