边缘计算环境下基于动态反馈的 Kubernetes 调度算法

林 博* 张惠民
LIN Bo ZHANG Hui-min

摘 要 针对资源受限的边缘计算环境下的容器应用的调度问题，提出了一种基于动态反馈的 Kubernetes 调度算法，该调度算法结合了边缘计算的资源环境，加入了网络带宽、磁盘容量和 I/O 速率作为调度侧率输入参数，并结合动态反馈的机制提升调度算法对节点硬件资源的感知能力，以提高边缘计算环境下的应用调度效率和硬件资源的利用效率。通过构建集群验证该调度算法相较与 Kubernetes 默认算法具有明显提升。

关键词 Kubernetes；容器技术；边缘计算；动态反馈

doi：10.3969/j.issn.1672-9528.2019.10.028

0 引言

边缘计算是目前物联网应用中的核心技术[1]，边缘计算通过边缘设备及边缘服务器的服务能力在边缘侧完成数据的实时处理；同时，边缘服务器通过容器技术的虚拟化环境[2]，为不同用户和边缘设备提供计算服务卸载能力，借助容器技术无需考虑程序的环境配置问题和资源隔离的安全问题的特点，增加了物联网应用的灵活性。

然而，对于边缘计算环境而言，其硬件资源（CPU、内存、磁盘、网络等）通常处于受限状态，如何在边缘计算环境下高效的管理和调度边缘计算环境下的容器应用是目前边缘计算应用服务面临的一大问题[3]。目前，主流云服务厂商对于容器应用的管理和调度通常采用 Kubernetes 容器集群管理工具。但是 Kubernetes 容器集群管理工具的设计目标是用于管理云计算环境下的大规模容器应用，因此在调度算法上使用了简单稳定的静态调度算法[4]，但是该算法对于边缘计算的资源受限环境而言，存在较大的局限性。本文针对这一问题提出了一种基于动态反馈的 Kubernetes 调度算法，并且加入了网络、磁盘容量和 I/O 速率作为调度算法的输入参数，通过实验分析验证，本算法有效的改善了在边缘计算环境下的调度效率。

1 相关研究

Kubernetes 是 Google 根据其内部集群管理系统 Borg 思想所设计的开源容器集群管理工具[5]，一种用于在服务器或集群上管理和维护容器化应用程序的系统级工具。通过 Kubernetes 用户可以定义容器化应用的运行方式，应用服务内部的交互方式，以及和外部服务或外部用户间的交互途径，并提供应用服务的水平扩容和垂直扩容，应用服务的灰度升级等实用的服务运维功能[6]。

Kubernetes Scheduler 调度器是 kubernetes 资源调度的核心组件，其职责是将 API Server 或 Controller Manager 新建的待调度 Pod 根据指定的调度算法与集群中某个合适的工作节点进行绑定，并将 Pod 和 Node 的绑定信息写入 etcd 中[7]。而后工作节点 Node 通过守护进程 Kubelet 监听到 Kubernetes 调度器发出的 Pod 和 Node 绑定信息，并从 etcd 中获取 Pod 配置文件，最后根据配置文件完成容器应用的启动。Kubernetes 调度器在调度中主要完成以下两部分工作：第一，收集当前 Kubernetes 集群中所有 Node 节点的资源负载情况并根据目标参数分析节点信息，第二，根据分析的节点信息和调度算法分发新建的 Pod 到 Kubernetes 集群中的目标节点。

Kubernetes 调度器中的默认调度方法分为两个阶段，Predicates 阶段和 Priority 阶段。Predicates 阶段的工作是查询集群中所有节点，根据 Predicates 的算法选择适用的节点完成初筛。Priority 阶段的工作是根据 Priority 中的算法给 Predicates 阶段初筛的节点进行评分，挑选出得分最高的节点作为调度的目标节点。Predicate 筛选算法包含有五种策略，分别是：PodFitsPorts、PodFitsResources、NoDiskConflict、PodSelectorMatches、PodFitsHost。

Predicate 阶段要求满足条件的节点必须通过所有筛选

* 陆军装甲兵学院信息通信系 北京 100072

策略。下面分别介绍五种策略的筛选内容：

1）PodFitsPorts：筛选 Pod 所需端口是否被占用，如果被占用则过滤该节点。

2）PodFitsResources：筛选 Pod 所需资源是否满足，如果不满足则过滤该节点。

3）NoDiskConflict：筛选节点上的磁盘卷标与 Pod 中的磁盘卷标是否存在冲突，如果存在冲突则过滤该节点。

4）PodSelectorMatches：如果 Pod 指定了 NodeSelector 属性，则筛选出与 Pod NodeSelector 属性匹配的节点。

5）PodFitsHost：如果 Pod 指定了 HostName，则筛选出与 HostName 匹配的节点。

在 Priority 阶段的算法有：Least RequestedPriority、Balanced ResourcesAllocation。下面简单说明两种算法的主要内容：

1）LeastRequestedPriority：选择资源最小消耗节点算法分别计算各节点已运行的 Pod 和待调度的 Pod 所需的 CPU 和内存消耗量，计算调度至该节点后的 CPU 和内存空闲率，并扩展 10 倍作为 CPU 和内存的最终得分，最后取两者得分的算术平均值，计算公式如下：

$$CpuScore = \frac{NodeCputotal - PodCputotal}{NodeCputotal} \times 10$$

$$MemoryScore = \frac{NodeMemorytotal - PodMemorytotal}{NodeMemorytotal} \times 10$$

$$LeastScore = \frac{CpuScore + MemoryScore}{2}$$

其中$NodeCputotal$和$NodeMemorytotal$分别表示该候选节点的 CPU 和内存资源总量，$PodCputotal$和$PodMemorytotal$表示已运行的 Pod 和待调度的 Pod 所需 CPU 和内存资源消耗量，若$MemoryScore$和$CpuScore$小于 0，则直接返回 0。

2）BalancedResourcesAllocation：选择资源使用最均衡节点算法分别计算各节点已运行的 Pod 和待调度的 Pod 所需的 CPU 和内存使用率，取内存和 CPU 使用率之间的差值再乘以 10，最后由 10 来减去前一步的计算结果，获得节点 CPU 和内存间的均衡率，计算公式如下：

$$CpuUse = \frac{PodCputotal}{NodeCputotal}$$

$$MemoryUse = \frac{PodMemorytotal}{NodeMemorytotal}$$

$$BalanceScore = 10 - \text{abs}(CpuUse - MemoryUse) \times 10$$

最终在获取了两种算法对候选节点的评分后，Kubernetes 调度器取两种算法的加权平均值作为各个节点的最终评分：

$$FinalScore = LeastScore \times 1 + BalanceScore \times 1$$

默认情况下两种算法的加权因子均为 1，取评分最高的节点作为调度该 Pod 的目标节点。

通过分析 Kubernetes 调度器的默认算法可以看出，其算法对于边缘计算节点而言存在较大的局限性，算法虽然考虑到了节点的 CPU 和内存使用率和空闲率，但是缺乏对网络带宽和磁盘 I/O 性能的考虑，从而直接影响 Kubernetes 调度器调度应用的运行结果不能达到预计结果，且网络带宽和磁盘性能可能会间接影响应用部署的速度降低 Kubernetes 调度器的调度性能。

其次，Kubernetes 调度器的默认算法对于资源的占用情况分析过于粗糙，在 LeastRequestedPriority 和 BalancedResourcesAllocation 算法中分别计算了节点的 CPU 和内存的资源空闲率和占用率，但是仅仅通过取了 CPU 和内存的算术平均值，作为该算法的最终评价得分，这导致了对于 CPU 和内存的实际运行状态监测模糊，同样的最终评价得分可能产生的原因有多种情况，比如：内存占用高、CPU 占用低，CPU 占用高、内存占用低两种不同的节点运行状态最终会产生同样的评分，这就导致可能调度高 CPU 使用率和低内存使用率的 Pod 应用至内存占用高、CPU 占用低的节点，从而严重影响 Pod 应用的运行状态。

2 基于动态反馈的 Kubernetes 资源调度算法

2.1 算法改进思路

针对目前 Kubernetes 调度算法在边缘计算环境下存在的问题，本文提出了一种改进型的资源调度方法：首先，根据调度的 Pod 应用状态划分调度应用的所需资源占用的权重；第二，仍然采用 Kubernetes 调度器中 Predicates 阶段，筛选出可用的候选节点，并根据权重归一化设置 Priority 阶段节点资源的计算时的加权平均值权重因子；最后，根据设置的权重和得分计算出目标节点的具体评价总分，并以此作为调度器的调度依据。

2.2 Pod 应用部署算法

在 Pod 应用的调度过程中，不同服务类型的 Pod 应用的对于资源的需求各不相同，而不同类型的资源需求对于节点的性能影响也不相同。因此，根据 Pod 应用的资源需求，改进算法将分别计算不同 Pod 应用的资源占用比例，从而更加高效准确的完成 Pod 应用的调度任务。

在 Pod 应用的配置文件中会准确的列出该应用所需的资源使用情况，改进算法通过对 Pod 配置文件的预处理，读取出 Pod 应用的所需资源，与参考的标准应用的使用资源间进行对比，分别计算出该 Pod 的资源使用权重，作为 Pod 应用部署时的调度依据。例如，对于一个 Pod 应用其资源需求文件如图 1 所示：

```
resources:
  limits:
    cpu: 1
    memory: 2048Mi
  requests:
    cpu: 0.5
    memory: 1024Mi
```

图 1　Pod 资源需求图

其所需的 CPU 核数为 C，内存为 M，参考标准应用资源需求为 CPU 核数为 Cs，内存为 Ms，得出其权重为$\frac{C}{Cs}:\frac{M}{Ms}$，归一化后分别为$\frac{C/Cs}{C/Cs+M/Ms}:\frac{M/Ms}{C/Cs+M/Ms}$。根据归一化的比值，计算候选节点得分情况：

$$InitNodeScore = Cn_1 \times \frac{C/Cs}{C/Cs+M/Ms} + Mn_1 \times \frac{C/Cs}{C/Cs+M/Ms}$$

其中，节点 CPU 核数为Cn_1，内存为Mn_1，节点得分为$InitNodeScore$，取节点得分最高的节点作为第一次部署节点。

2.3　Pod 应用调度算法

监测第一次部署的 Pod 应用运行状态，通过 Kubernetes 内置接口获取 Pod 应用运行中的 CPU 占用情况、内存消耗量、磁盘消耗量、网络带宽、I/O 速率信息，分别计算权重后，进行归一化处理，（计算方法如 2.3 小节中 Pod 应用部署所示），作为动态反馈调度算法评价节点状态的参考值。

读取候选节点的 CPU 占用情况、内存消耗量、磁盘消耗量、网络带宽、I/O 速率信息，分别计算上述 5 个参数的权重，根据 Pod 应用运行时的归一化参数，求取节点的最终评分，取最高点作为节点调度的目标。

3　实验分析

3.1　Kubernetes 集群评价方法

为了对比基于动态反馈的改进算法与 Kubernetes 调度器默认算法的区别，本文引入了一种集群负载状态评价方法，该方法首先计算各个节点的负载状态评分，再计算所有节点评分的平均值。其中，单个节点负载状态评分的计算方法如下：

读取各个节点的资源空闲状态，CPU 空闲率 Rc、内存空闲率 Rm、磁盘空闲率 Rd、网络带宽空闲率 Rn、磁盘 I/O 空闲率 Ri，分别使用节点的空闲率与标准的节点空闲状态对比，得出个节点的空闲状态，取空闲状态标准状态比值的标准差作为节点负载的最终评分。

$$S_{node} = \sigma(\frac{Rc}{Rcs} + \frac{Rm}{Rms} + \frac{Rd}{Rds} + \frac{Rn}{Rns} + \frac{Ri}{Ris})$$

其中，S_{node}表示该节点的最终得分，Rcs、Rms、Rds、Rns、Ris 分别表示参考 CPU、内存、磁盘、网络带框和磁盘 I/O 空闲率，其中参考空闲率可根据节点配置规格调整。

3.2　实验设计

为降低集群性能对实验结果准确性的影响，本文实验环境采用同一个集群环境测试，Kubernetes 调度工具均采用 1.15.2 版本，并基于该版本加入本文提出的基于动态反馈的 Kubernetes 调度算法。集群采用一个 Master 节点，四个 Node 节点，各节点均采用相同的表所示配置规格。

表 1　节点配置表

资源类型	资源规格
CPU	Intel Core i7 1 核
内存	2GB
磁盘容量	40GB
网络带宽	10Mbps
操作系统	Centos 7
Kubernetes 版本	V1.15.2

测试方法：

1）使用为改进的 Kubernetes 搭建默认集群环境；

2）构建测试所用的 20 个 Pod 应用镜像，作为测试用例；

3）开始部署 20 个 Pod 应用，待调度完成后，分别计算各节点的评分；

4）删除默认 Kubernetes 搭建的集群环境，并使用改进调度算法的 Kubernetes 搭建心的集群环境；

5）重复第 3 步骤内容，测试完成后删除集群。

3.3　实验结果分析

分别测试 Kubernetes 默认调度算法和基于动态反馈的 Kubernetes 调度算法后，得到各节点的负载评分如表 2 和图 2 所示下：

表 2　实验结果

调度算法	Node1	Node2	Node3	Node3
默认算法	0.53	0.34	0.42	0.25
改进算法	0.21	0.35	0.15	0.21

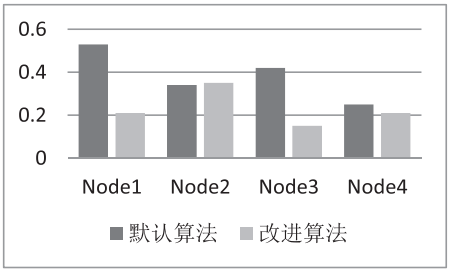

图 2　实验结果柱状图

分析图 2 的实验结果可以看出，对于 Node1、Node3 和 Node4，改进算法均提升了节点的负载评分，Node2 虽然仍低于默认算法的负载评分，但是从总体上来看改进算法提升了集群的整体负载均衡度。

（下转 108 页）

（5）垃圾桶语音播报

能在垃圾桶满桶时，智能播报当前桶内垃圾物的状态，当垃圾桶满时，迅速联系附近工作人员，清理垃圾。

（6）手机 APP 控制

通过蓝牙无线通信，实现手机 APP 与垃圾桶内控制器之间的交互，实现手机对垃圾桶的控制，以及垃圾桶内垃圾物的盛放情况。

5 结束语

本文采用的基于图像识别、树莓派和 Arduino 开发板的智能垃圾桶模拟方案，树莓派作为上位机系统，主要负责将收集回来的垃圾物照片进行分析处理，判定出垃圾物的回收种类。并将种类信息通过串口发送给 Arduino 开发板，进而控制舵机工作，实现分类效果。并结合其他传感器，实现一些附加功能。通过对智能垃圾桶的研究和模拟智能垃圾桶的分类功能的实现，可以初步构建智能垃圾桶的模型与理论基础，为智能垃圾桶技术提供更多的可能。

参考文献

[1] 廉小亲 , 成开元 , 安飒 , 吴叶兰 , 关文洋 . 基于深度学习和迁移学习的水果图像分类 [J]. 测控技术 ,2019,38(06):15-18.

[2] 王玥 , 刘苇 , 崔昊 . 基于 STC89C52 单片机的智能分类垃圾桶的设计 [J]. 科技经济导刊 ,2019,27(16):66.

[3] 赵永永 . 基于 PID 智能控制算法的智能车车速控制研究 [J]. 软件 ,2019,40(01):195-198.

[4] 安飒 , 廉小亲 , 成开元 , 王俐伟 , 李麟杰 . 基于 OpenMV 的无人驾驶智能小车模拟系统 [J]. 信息技术与信息化 ,2019(06):16-20.

[5] 张美平 , 吴德平 , 王灿杰 , 谢玮铭 . 基于树莓派的智能家居设计与实现 [J]. 计算机系统应用 ,2019(08):109-114.

[6] 徐昆 , 朱国华 , 刘文凤 , 范超 .YOLO 算法在目标姿态检测中的应用 [J/OL]. 电子技术与软件工程 ,2019(16):181-182[2019-08-27].

【作者简介】

王科举（1995-），男，河北邯郸人，硕士，研究方向：物联网技术与应用；

廉小亲（1967-），通讯作者，女，河南省阳人，博士，教授，研究方向：计算机测控技术；

安飒（1994-），女，辽宁锦州人，硕士，研究方向：物联网技术与应用；

陈彦铭（1997-），男，甘肃金昌人，硕士，研究方向：物联网技术与应用；

龚永罡（1973-），男，河南洛阳人，副教授，研究方向：计算机应用。

（收稿日期：2019-09-05）

（上接 103 页）

4 结语

本文针对 Kubernetes 在资源受限的边缘计算环境下调度算法存在的不足展开研究。根据边缘计算 CPU、内存、网络、磁盘受限的条件改善了 Kubernetes 调度器的默认算法，使其能更加精确地检测节点的资源使用状况，提升 Kubernetes 在边缘计算环境的调度效率。最后，通过构建轻量级集群进行试验验证，改进后的算法均衡了各节点间的资源负载。

参考文献

[1] 董思岐 , 吴嘉慧 , 李海龙 , 等 . 面向优先级业务的移动边缘计算资源分配策略 [J]. 计算机工程 .2019,1-7.

[2] 张楠 . 移动云环境下资源协同管理与分配机制研究 [D]. 北京科技大学 ,2018.

[3] 杨世欣 . 面向边缘计算的服务资源分配问题研究 [J]. 微型电脑应用 ,2019,35(8):54-56.

[4] 唐瑞. 基于 Kubernetes 的容器云平台资源调度策略研究 [D]. 成都 : 电子科技大学 ,2017.

[5] 杜军 . 基于 Kubernetes 的云端资源调度器的改进 [D]. 浙江大学 ,2016.

[6] 彭丽苹 . 基于 Docker 的云资源弹性调度策略 [J]. 计算机应用 , 2018,38(2):557-562.

[7] XinLi, Zhuzong Qian, Sanglu Lu, Jie Wu. Energy efficientvirtual machine placement algorithm with balanced andimproved resource utilization in a data center[J]. Mathematicaland Computer Modeling, 2013.

【作者简介】

林博（1996-），男，四川资中人，硕士研究生，主要研究方向：边缘计算；

张惠民（1973-），男，福建上杭人，教授，博士，主要研究方向：云计算、大数据、深度学习、物联网。

（收稿日期：2019-09-06）