·热点与综述· 文章编号:1000-3428(2021)09-0044-07 文献标志码:A 中图分类号:TP391

面向微服务架构的云系统负载均衡机制

施凌鹏[1],朱 征[2],周俊松[3],李 鑫[3],李 静[3]
(1.国网上海市电力公司 信息通信公司,上海 200072;2.国网上海市电力公司 电力科学研究院,上海 200437;
3.南京航空航天大学 计算机科学与技术学院,南京 210023)

摘 要: 针对微服务架构的请求响应延迟上升问题,提出一种微服务链感知的请求负载均衡算法。从负载均衡器入手,将微服务链上的平均请求延迟和主机负载情况作为衡量指标,形式化微服务环境和请求延迟,在此基础上研究微服务链调用中存在的共享微服务竞争问题。模拟实验结果表明,与RR算法相比,该算法在复杂的微服务链环境下能够有效降低请求延迟,且在实例分布不均匀的环境中保持较好的负载性能,均衡不同主机之间的负载。在更接近真实应用环境的高频请求测试中,算法能有效降低系统的综合响应时间。

关键词: 微服务架构;云计算;负载均衡;微服务链;容器化平台

开放科学(资源服务)标志码(OSID):

中文引用格式: 施凌鹏,朱征,周俊松,等.面向微服务架构的云系统负载均衡机制[J].计算机工程,2021,47(9):44-50,58.

英文引用格式: SHI L P,ZHU Z,ZHOU J S,et al.Load balancing mechanism for microservice architecture in cloud-based systems[J].Computer Engineering,2021,47(9):44-50,58.

Load Balancing Mechanism for Microservice Architecture in Cloud-based Systems

SHI Lingpeng[1],ZHU Zheng[2],ZHOU Junsong[3],LI Xin[3],LI Jing[3]
(1.Information and Communication Company,State Grid Shanghai Electric Power Company,Shanghai 200072,China;
2.Electric Power Research Institute,State Grid Shanghai Electric Power Company,Shanghai 200437,China;
3.College of Computer Science and Technology,Nanjing University of Aeronautics and Astronautics,Nanjing 210023,China)

【Abstract】 In order to solve the rising delay in responding to requests in Microservices Architecture(MSA),a load balancing algorithm for requests is proposed based on microservice chain awareness.To formalize the request delay and microservice environment,the algorithm takes the average request delay and the host loads as the evaluation indicators.On this basis,the competition for shared microservices in microservice chain calling is studied.The simulation results show that compared with the RR algorithm,the proposed algorithm can significantly reduce the delay in requests in complex microservice chain environment.In addition,it keeps excellent loading performance in an environment with unevenly distributed instances,and balances the loads of different hosts.In the more realistic tests of high-frequency requests,the proposed algorithm can significantly reduce the overall response time of the system.

【Key words】 Microservices Architecure(MSA);cloud computing;load balancing;microservice chain;container platform

DOI: 10.19678/j.issn.1000-3428.0058747

0 概述

单体架构应用在面临当前互联网环境的快速迭代和用户需求的迅猛变化情形时,开发成本与运维成本大幅上升,因此,越来越多的应用采用微服务的架构。微服务架构(Microservices Architecure,MSA)是面向服务架构(Service-Oriented Architecure,SOA)的一种变体[1],是将一个大型的复杂软件应用拆分为多个松耦合的微服务,集群调度和扩容的粒度降低到微服务实例级别,能够降低开发、部署和扩缩容

基金项目: 国家电网有限公司科技项目"云环境下的容灾备份恢复与业务连续性管理关键技术研究与示范应用"(SGSHXT00JFJS1900093)。
作者简介: 施凌鹏(1974—),男,高级工程师、硕士,主研方向为信息通信技术;朱 征,工程师;周俊松,硕士研究生;李 鑫、李 静,副教授。
收稿日期: 2020-06-24 **修回日期:** 2020-08-26 **E-mail:** lics@nuaa.edu.cn

的成本[2]。

在降低服务粒度的同时，服务治理将面临更大的挑战。请求通常需要通过若干个与之相关的微服务的处理才能够将正确预期的结果返回给用户。而在复杂微服务框架下如何治理请求以及优化请求的处理时延是本文研究的目标。

本文通过负载均衡与加入微服务链感知的方法降低微服务架构中服务请求的时延，并提出基于微服务链感知的负载均衡算法。阐述微服务架构部署场景，绘制云数据中心的架构拓扑图，并给出微服务实例、微服务链和主机描述及用户请求延迟的衡量标准，以减少微服务链在数据中心网络中的通信成本。最后基于形式化的模型，根据Python编写模拟实验，并设置多组接近于真实环境的不同实验。

1　相关工作

微服务概念是由LEWIS等于2014年定义的，它是一种架构风格[3]和SOA的具体实现[4]，主要思想是将传统单体应用根据业务逻辑和其功能拆分成一系列可以被独立设计、开发、部署、运维的软件服务单元，并且在遵守服务边界的前提下，各个微服务能够彼此相互配合与协作来实现整个系统的价值[5]。

MSA是指根据整个应用系统的业务需求，通过预先划定服务边界，定义微服务进行服务组合而形成的企业级分布式应用体系架构[6]。文献[7-8]研究传统云计算架构中的任务调度与负载均衡问题，而针对微服务的治理未进一步扩展。文献[9]研究基于服务网格的微服务治理方案，但没有涉及对服务链治理问题。

文献[10-11]对容器运行时的系统进行了基于事件的性能分析，提出在以容器为基础的微服务系统中，负载均衡对微服务具有显著影响。文献[12]通过沙盒化微服务构建性能模型，合理地为不同服务配置容量，从另一种角度对负载问题提出了可行的解决方案。文献[13]提出基于消息队列的面向链的负载均衡算法，将HTTP与消息队列结合使用，但在微服务系统中会导致额外的操作复杂度，增加开销。

本文在上述研究的基础上，针对微服务链调用中存在共享微服务竞争的问题，从负载均衡的角度入手分析云化微服务架构下的特点，提出微服务链感知的请求负载均衡算法，并通过模拟实验从微服务链上的平均请求延迟和主机请求负载均衡的角度对算法进行有效评估。

2　微服务链感知的请求负载均衡算法

本节从微服务链式调用的场景出发，通过预先对请求对应的微服务链进行建模分析，获取最优的服务执行路径，实现对负载配置和请求延迟的综合优化。

2.1　应用场景描述

2.1.1　数据中心网络

本文从实际场景出发，对云数据中心的微服务通信组合问题展开研究。数据中心网络拓扑结构如图1所示。

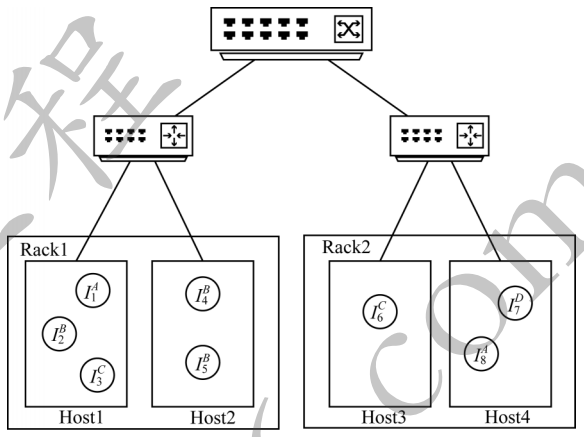

图1　数据中心网络拓扑结构

Fig.1　Structure of data center network topology

微服务以容器的方式进行部署并运行在主机上，同一主机可同时部署多个微服务实例，因此将主机H_i描述为一个微服务实例的子集，即$H_i \subseteq I$。微服务实例与主机之间的关系如式(1)所示：

$$\mathrm{InHost}(H_i, I_j^k)=\begin{cases}0, \text{微服务器} I_j^k \text{不位于主机} H_i \\ 1, \text{微服务器} I_j^k \text{位于主机} H_i\end{cases} \tag{1}$$

主机内部的圆形(形式为I_j^k)代表在该主机上部署的微服务实例，i是该微服务实例的唯一标识，j代表该微服务实例隶属于哪一种微服务。

2.1.2　微服务链

微服务链是微服务的有序集合，为本文微服务通信组合的主要研究对象，以符号C_i表示，i为该微服务链的唯一标识。

用户请求R_i由特定微服务链提供响应，i是用户请求的唯一标识。用户请求将唯一对应一个微服务链，式(2)描述了该映射关系：

$$\mathrm{RToChain}(R_i)=G_j \tag{2}$$

图2所示为用户请求由一个基本的微服务链对应的过程。其中，箭头代表数据传输的方向，箭头上的符号D_{A}代表微服务，M_{A}代表传送给微服务，M_{B}代表请求的数据大小，D_{B}、D_{C}、D_{D}以此类推，在服务链式调用的场景下，服务间的传输时延同样会对服务性能产生较大影响，因此通过D_{A}计算传输时延，并以此作为链路权重。

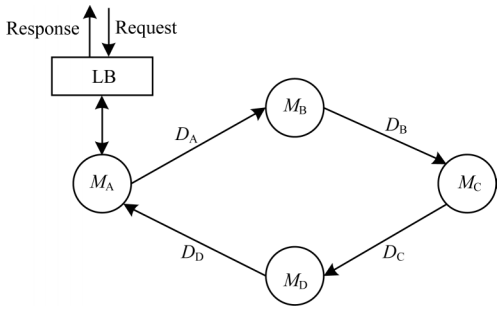

图2 微服务链简单拓扑

Fig.2 Simplified topology of microservice chain

微服务链可以表示如下：

$$C_1=\{M_A,M_B,M_C,M_D\} \tag{3}$$

其中：M_i表示微服务实例对象，i为唯一性标识。

2.1.3 微服务与微服务实例

微服务链与微服务实例之间存在多对多的关系，通过式(4)描述微服务与微服务链联系：

$$\mathrm{NInChain}(C_i,M_j)=\begin{cases}0,\text{微服务器}M_j\text{不属于服务链}C_i\\1,\text{微服务器}M_j\text{属于服务链}C_i\end{cases} \tag{4}$$

同时可将微服务实例定义为I_i^j，其中，i是该微服务实例的唯一标识，j说明该微服务实例提供的是微服务M_j的服务。

2.2 系统模型

复杂网络中频繁多次的请求是造成服务时延增加的主要原因[11]，本文将负载均衡问题拓展为微服务环境下服务通信组合调用问题，以微服务链为研究对象，寻求服务链式组合情况下的性能优化策略。

本节将影响微服务环境中负载配置和请求响应效率的关键因素予以形式化处理(见表1)，并基于此构建面向微服务链的请求响应模型。

表1 参数的形式化描述

Table 1 Formalized description of the parameters

符号	定义
R	请求集合
R_i^j	请求ID为i，微服务链j的请求
R^j	微服务链j的请求总数
C_i	微服务链i
M_i	微服务i
I_i^j	实例ID为i，微服务j的微服务实例
H_i	主机ID为i的主机
P_i^k	请求ID为i，微服务k的等待与执行时间
E^k	微服务k的执行时间
W_i^x	请求i在微服务实例x上的等待时间
T_i^{xy}	请求i在微服务实例x到微服务实例y的传输时间

为简化抽象系统，本文重点研究微服务环境下的服务请求问题[14]，设定微服务实例所获资源相同，且主机环境同构，并将主机任务(即微服务实例运行的任务)串行化。

所有发向负载均衡器的请求由一个特定队列接收，并对队列中的请求分别构建微服务通信组合模型，通过代价函数遍历组合实例得到代价最小的实例组合[15]，该组合在一段时间内具有有效性。

2.3 衡量指标

衡量指标是结果分析中的重要因素，针对云系统下的微服务架构环境，本文选取微服务链上的平均请求延迟和主机负载作为衡量指标[16]。

微服务链上的平均请求延迟计算方法如式(5)所示：

$$L_c=\frac{\sum_i l_i}{R^c},\left\{i\,\middle|\,\mathrm{RToChain}(R_i)=R_c\right\} \tag{5}$$

其中：l_i是单个请求的延迟时间[17-18]，如式(6)所示，主要由等待执行时间P_i^k和请求数据传输时间T_i^{xy}组成：

$$l_i=\alpha\sum P_i^k+\beta\sum T_i^{xy} \tag{6}$$

其中：等待执行时间P_i^k由等待时间W_i^x和执行时间$E^{m(k)}$组成($m(k)$可获取微服务实例k的微服务类型)，如式(7)所示：

$$P_i^k=W_i^k+E^{m(k)} \tag{7}$$

等待时间W_i^k的计算如式(8)所示，其代表当前实例处理完成所有等待在队列中的请求的时间之和。

$$W_i^k=\sum E^{m(i(r))}\ \ \mathrm{s.t.}\left\{r\,\middle|\,r\in Q(h(k))\right\} \tag{8}$$

其中：$h(k)$为k实例所在的主机；$Q(x)$为所有在主机x队列中的请求；$i(r)$为获取请求r所在的实例。

除等待时间外，数据传输时间对请求响应的性能同样也有显著影响，因此为传输链路设置权重如式(9)所示：

$$T_i^{xy}=\frac{D_{m(x)}}{\mathrm{speed}(x,y)} \tag{9}$$

传输速度使用2个微服务实例之间的距离进行衡量，如式(10)所示：

$$\mathrm{speed}(x,y)=\begin{cases}\mathrm{sameHost},\text{实例}x\text{和}y\text{在同一主机}\\\mathrm{sameRack},\text{实例}x\text{和}y\text{在同一机架}\\\mathrm{diffRack},\text{实例}x\text{和}y\text{在不同机架}\end{cases} \tag{10}$$

综上，可计算出每个微服务链上的平均请求延迟。平均请求延迟代表整个微服务架构中服务质量方面的性能，若微服务链上的平均请求延迟越小，则证明该架构的服务质量越高。

2.4 算法设计

微服务链感知的请求负载均衡算法如算法1所示。

算法1 微服务链感知的请求负载均衡算法

输入 用户请求集合R

输出 请求微服务实例的顺序

1.for each $R_i\in R$ do

2.chain ← RToChain(R_i)；
3.MSset ← getMSfromChain(chain)；
4.DSset ← getDSfromMSset(MSset)；
5.insOrder ← setEmptyInsfromMSset(MSset)；
6.while DSSet is not empty do
7.data，priorityMS ← findMaxDataSize(DSSet)；
8.nextMS ← next(MSset，priorityMS)；
9.pMSIns ← getInsfromMS(priorityMS)；
10.nMSIns ← getInsfromMS(nextMS)；
11.pIns，nIns ← findMinCost(pMSIns，nMSIns)；
12. if insOrder. find(priorityMS) is true and insOrder. get(priorityMS) is null then
13.insOrder.set(priorityMS，pIns)；
14.end if
15. if insOrder. find(nextMS) is true and insOrder. get(nextMS) is null then
16.insOrder.set(nextMS，nIns)；
17.end if
18.DSset.pop(priorityMS，data)；
19.end while
20.send(R_i，insOrder)；
21.end for

算法1首先接收到用户的请求，通过用户请求内容分析出功能所属的微服务链，根据微服务链确定该链上的微服务类型以及它们的依赖关系；再借助微服务链中的数据传输信息，分析出首要解决的微服务通信组合，并使用代价函数对其组合中的实例进行遍历计算，寻找出所得代价最小的实例组合，使用该实例组合对该请求的微服务实例顺序进行更新；然后继续去寻找需要优化的微服务通信组合。重复上述过程，直到所有微服务都选中了实例，最终将接受请求并将微服务实例请求顺序与请求本身一并转发，完成对该请求的负载均衡。

为简化分析算法的时间复杂度，假定请求对应的微服务链中微服务集与数据集数目均为n，数据集DSSet预先排序降低之后的时间开销，算法的时间复杂度主要由排序算法和while循环的时间开销共同决定，因此总的时间复杂度为$O(n\text{lb}\ n)$。

3　仿真实验结果与性能分析

模拟实验根据Python编写，通过面向对象的编程范式来对上文的问题模型进行抽象和编写。当前有很多的云模拟器如CloudSim[19]，但大部分云模拟器都是面向虚拟机模式的，较少有支持容器这种新型的进程隔离“虚拟化”方式，也因为问题模型较为独特，所以参考文献[20]采用自定义调度器的方案。

3.1　参数设置

以上文设计的模型为基础，模拟实验中的网络架构与数据中心网络架构一致，即1个路由器、2个交换机、3台主机。在该模拟实验中，模拟的时间片的单位为ms，数据量大小为KB，对于某个微服务而言，其返回数据量范围为1~100 KB，其实例处理请求的时间范围为1~10 ms。通过在程序初始化中描述不同的微服务来确定其返回的数据量大小与单一请求处理时长。对于数据中心网络之间的传输链路，本文粗略地将其认为是一个定值，也可以理解为是其链路的权重。对于同主机的2个实例，它们之间通信的速率主要受到磁盘吞吐量的限制；对于同机架不同主机的2个实例，它们之间的通信速率主要受到网卡吞吐量的限制；最后对于不同机架实例之间的通信，速率应该是三者最低的，而且考虑到数据中心骨干网络的压力。具体实验参数如表2所示。

表2　实验参数设置

Table 2　Parameters setting of experiment

参数	参数值	备注
α	1	等待处理时间的权值
β	1	请求传输时间的权值
sameHost	512	同一主机微服务示例之间的传输速度
sameRack	102	不同主机同一机架中微服务实例之间的传输速度
diffRack	30	不同机架中微服务实例之间的传输速度

本文主机同构的串行化假设主机性能相同，不需要使用加权轮询(RR)。因此，轮询的负载均衡算法成为本文的对比算法。

3.2　微服务链上的平均请求延迟评估

本文通过设置不同的微服务链、不同的实例放置方法、不同的请求发送机制和不同的负载均衡算法进行多次实验。

3.2.1　3条微服务链

本文微服务实例放置方式如图3所示，设置的3条微服务链如图4所示。

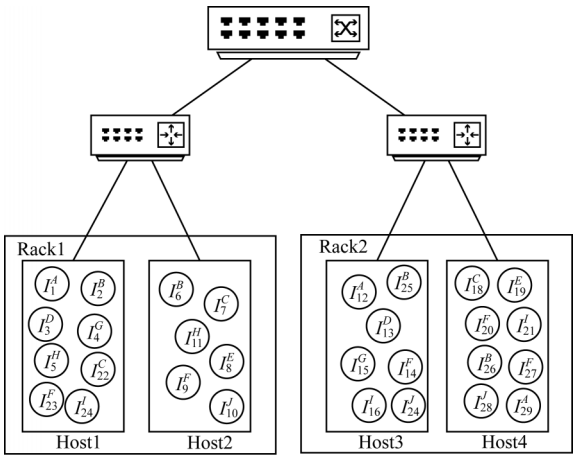

图3　3条微服务链实例放置方式

Fig.3　Instances placement way of three microservice chains

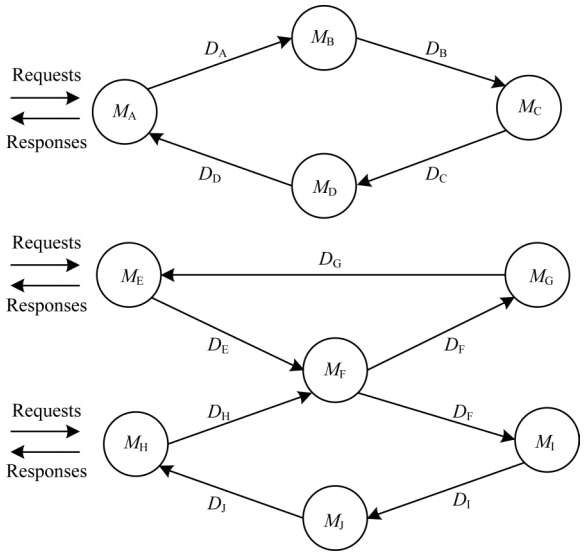

图4 3条微服务链简化拓扑

Fig.4 Simplified topology of three microservice chains

3条微服务链分别为：$C_1=\{M_A,M_B,M_C,M_D\}$，$C_2=\{M_E,M_F,M_G\}$，$C_3=\{M_H,M_F,M_I,M_J\}$。

相关的微服务详细信息如表3所示。

表3 3条微服务列表

Table 3 List of three microservices

微服务类型(ID)	微服务任务执行时间/ms	微服务返回数据量
A(1)	1	100
B(2)	5	200
C(3)	2	400
D(4)	1	300
E(5)	1	100
F(6)	3	200
G(7)	1	50
H(8)	2	100
I(9)	4	300
J(10)	3	100

本文使用对3条微服务链轮流请求的请求模式，每2个时间片(2 ms)向系统中发送1个服务请求，并通过式(5)计算每个微服务链的平均请求延迟，得到如图5所示的结果。

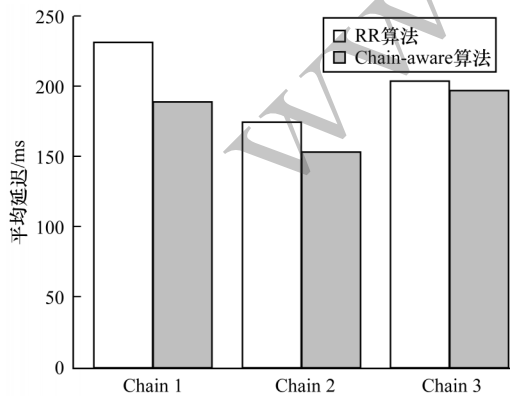

图5 3条微服务链的平均请求延迟

Fig.5 Average request latency of three microservice chains

本文提出的负载均衡方法的微服务平均延迟低于轮转调度的方法(−18.47%，−12.30%，−2.62%)，取得了较好的结果。

3.2.2 1条微服务链

除了3条微服务链的情况外，本文还设置了仅有1条微服务链的实验($C_1=\{M_A,M_B,M_C,M_D\}$)，以验证本文算法在不同频次请求下的性能，微服务详细信息如表4所示。

表4 1条微服务列表

Table 4 List of one microservice

微服务类型(ID)	微服务任务执行时间/ms	微服务返回数据量
A(1)	1	100
B(2)	3	200
C(3)	5	400
D(4)	2	300

对于1条微服务链的情况，本文设置2种请求模式来衡量性能，部署模式如图6所示。首先向系统发送中等频次的轮流请求，即每3个时间片(3 ms)发送1个请求，并通过式(5)计算微服务链的平均请求延迟，得到如图7所示的结果。

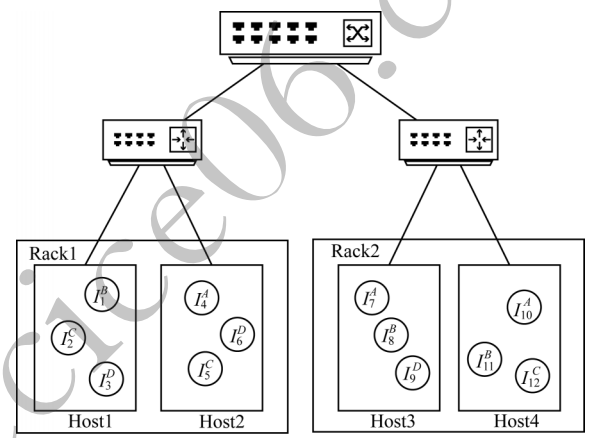

图6 1条微服务实例的放置方式

Fig.6 Instances placement way of one microservice chain

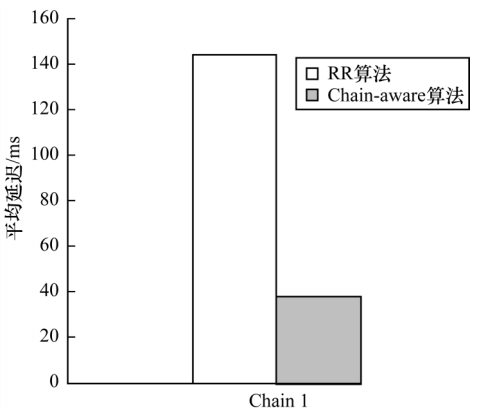

图7 中频次请求的平均延迟

Fig.7 Average latency of medium frequency requests

从图7可以看出，本文的方法能够极大地降低请求延迟，降低幅度达到73.6%。

为测试极端情况的性能（如高并发情景），向系统发送高等频次的轮流请求，即每2个时间片（2 ms）发送一个请求，得到如图8所示的实验结果。

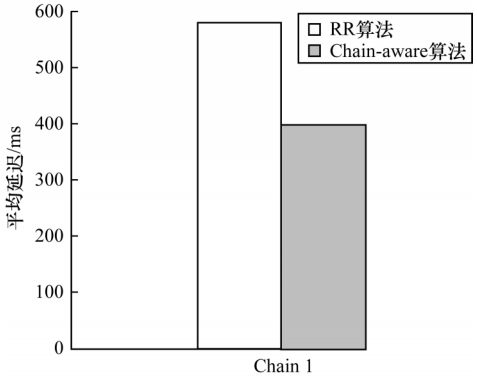

图8　高频次请求的平均延迟

Fig.8　Average latency of high frequency requests

从图8可以看出，本文方法即使在高负载情况下也有更好的性能。综合上述实验结果，在复杂多条微服务链的环境中，也就是更接近于真实微服务架构的环境中，该算法能够有效降低请求延迟。

3.3　主机负载情况评估

由于容器进程隔离的特性，容器能够与宿主机共享内核和计算资源。基于2.2节所提的串行化假设，当请求任务在执行时，则能够利用当前主机所有可用的计算资源。因此，本文只要通过每个主机的请求等待序列，并根据微服务划分的执行速度权重，就能够很好地衡量出主机实际的负载情况。

3.3.1　3条微服务链请求模式

对于3种微服务链轮流请求的请求模式，每2个时间片（2 ms）向系统中发送一个服务请求，并在每个时间片中记录每个主机的负载值，得到的结果如图9所示（彩图效果见《计算机工程》官网HTML版，下同）。

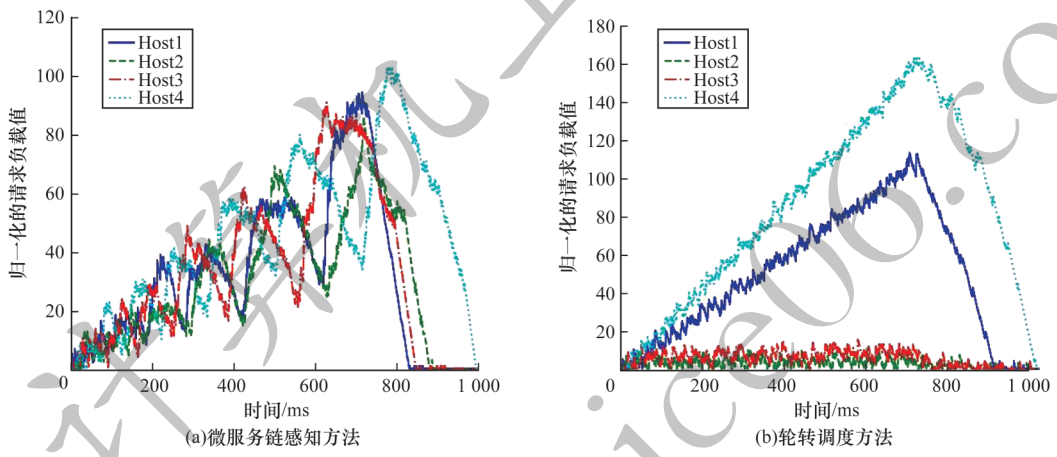

图9　3条微服务链分别请求下的主机负载情况

Fig.9　Host load conditions under separate requests of the three microservice chains

从图9可以看出，本文方法能够使主机之间的负载更加均衡，而轮转调度方法会造成严重的负载倾斜。

3.3.2　1条微服务链部署模式

对于1条微服务链的部署模式，首先需要向系统发送中等频次的请求，即每3个时间片（3 ms）发送一个请求，然后通过记录每个时间片上主机的负载值，得到如图10所示的1条微服务链中频次请求下主机负载情况。

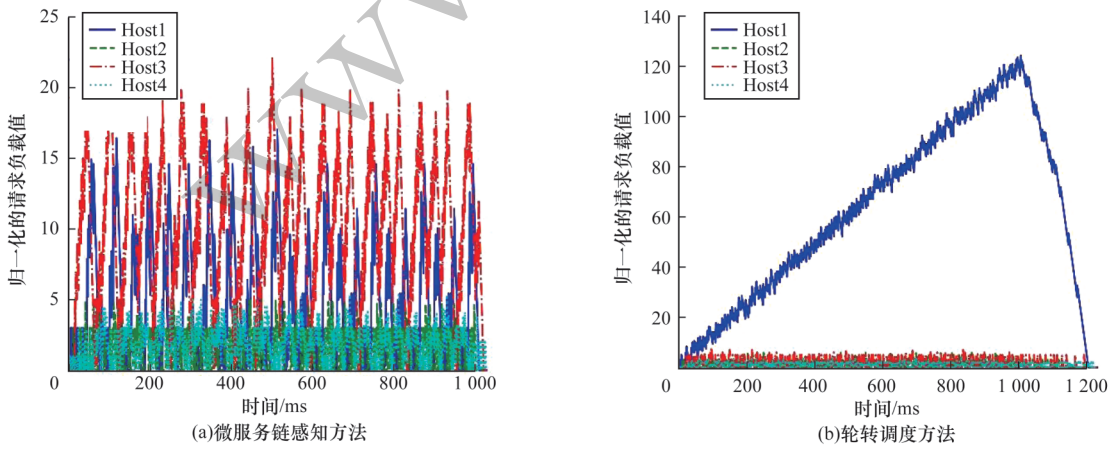

图10　1条微服务链中频次请求下的主机负载情况

Fig.10　Host load conditions under separate requests of one microservice chain

图10(a)为使用本文算法的主机负载情况，图10(b)为使用轮转算法的主机负载情况，可以看到在2组实验中本文方法主机之间的负载会更均衡，而轮转调度方法还是会存在严重的负载倾斜。

通过向系统发送高频次的请求，即每2个时间片(2 ms)发送1个请求，得到如图11所示的主机负载情况。从图11可以看出，在2组实验中，本文方法主机之间的负载会更均衡，而轮转调度方法还是会存在严重的负载倾斜。

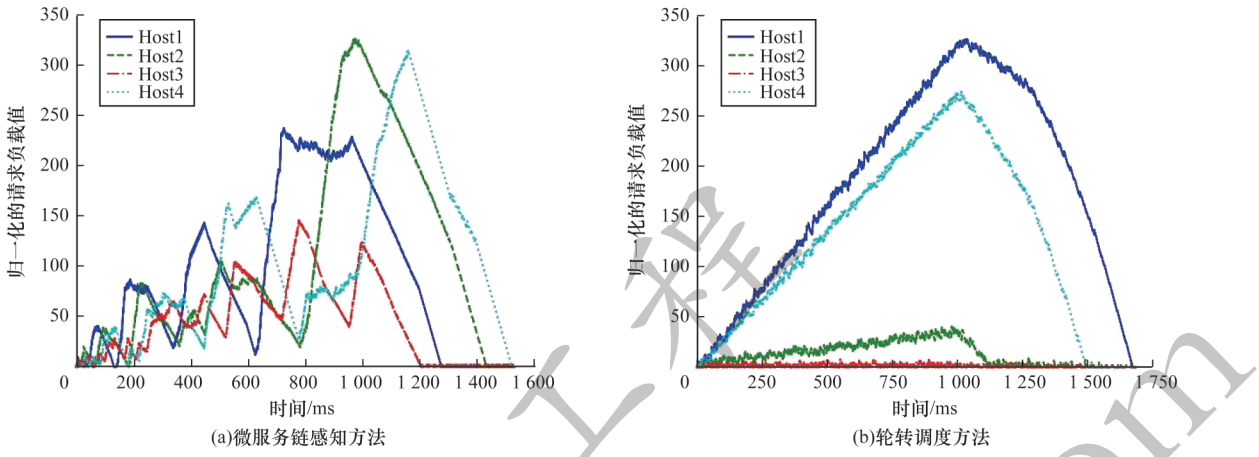

图11 1条微服务链高频次请求下的主机负载情况

Fig.11 Host load conditions under single high-frequency request of one microservice chain

3.4 实验结果

本文通过在4个3条微服务链和4个1条微服务链上不同的实验，根据不同的实例设置方式和请求模式，得到了较为接近真实环境的实验结果。

在3条微服务链的环境中，使用本文负载均衡策略的实验在微服务链的平均请求延迟上要比对比策略更低，主机的负载情况会比对比算法更加均衡。同时，本文算法在微服务链之间相互有交叉(即相关)的情况下表现更加优异，能够在极大地降低微服务链请求平均延迟的同时，保证主机负载的均衡。

在1条微服务链的环境中，当微服务实例较少且摆放方案不均匀时，本文微服务链感知的负载均衡策略能够有效地降低微服务链的请求时延，并且在高并发场景下仍有较好的表现，能够保证在接近于真实微服务架构环境中的复杂微服务链下，本文的负载均衡策略可有效降低请求延迟。

在主机负载方面，本文算法能够取得较为均衡的结果，而轮转调度方式会经常出现严重的负载倾斜。

4 结束语

本文从微服务架构的服务调用与负载均衡出发，结合传统云计算架构中的负载均衡和任务调度算法，根据微服务赋予云环境下的感知能力，提出微服务链感知的请求负载均衡算法。通过降低微服务架构中请求延迟时间平衡主机之间的负载，解决服务链调用中存在共享微服务竞争的问题。实验结果表明，该算法能够在复杂微服务请求调用链下降低请求平均延迟，有效均衡主机的工作负载。随着容器技术和微服务架构的不断发展，软件行业将对微服务治理提出更高的优化目标，由于微服务基于容器运行时的特点，下一步将微服务部署与优化相结合，提出更适用于特定场景的部署方案。

参考文献

[1] THONES J. Microservices[J]. IEEE Software, 2015, 32(1): 116-126.
[2] HASSELBRING W, STEINACKER G. Microservice architectures for scalability, agility and reliability in e-commerce [C]//Proceedings of IEEE International Conference on Software Architecture. Washington D. C., USA: IEEE Press, 2017: 243-246.
[3] FOWLER M, LEWIS J. Microservices[EB/OL]. (2014-03-25). [2020-05-20]. http://martinfowler. com /arti-cles/microservices. html.
[4] ZIMMERMANN O. Microservices tenets[J]. Computer Science-Research and Development, 2017, 32 (3/4): 301-310.
[5] 毕小红，刘渊，陈飞. 微服务应用平台的网络性能研究与优化[J]. 计算机工程，2018，44(5)：53-59.
BI X H, LIU Y, CHEN F. Research and optimization of network performance for micro-service application platform[J]. Computer Engineering, 2018, 44(5): 53-59. (in Chinese)
[6] NWEMAN S. Building microservices[M]. [S. 1.]: O'Reilly Media, Inc., 2015.
[7] JIANG S H, KIM T Y. The study of genetic algorithm-based task scheduling for cloud computing[J]. International Journal of Control and Automation, 2012, 5(4): 157-162.
[8] FERRIS J M. Load balancing in cloud-based networks[S]. USA Patent: 8, 849, 971, 2014-11-30.
[9] 郑俊褒，沈林强. 基于服务网格的微服务架构服务治理研究[J]. 计算机系统应用，2019，28(2)：55-61.
ZHENG J B, SHEN L Q. Research on service governance of microservice architecture based on service mesh[J] Computer Systems and Applications, 2019, 28(2): 55-61. (in Chinese)

(下转第58页)

（上接第50页）

[10] BRONDOLIN R, FERRONI M, SANTAMBROGIO M. Performance-aware load shedding for monitoring events in container based environments[C]//Proceedings of ACM SIGBED'19. New York, USA: ACM Press, 2019: 27-32.

[11] LLOYD W. Serverless computing: an investigation of factors influencing microservice performance [C]// Proceedings of IEEE International Conference on Cloud Engineering. Washington D. C., USA: IEEE Press, 2018: 159-169.

[12] JINDAL A, PODOLSKIY V, GERNDT M. Performance modeling for cloud microservice applications [C]// Proceedings of ACM/SPEC International Conference on Performance Engineering. New York, USA: ACM Press, 2019: 25-32.

[13] NIU Y P, LIU F M, LI Z P. Load balancing across microservices [C]//Proceedings of IEEE International Conference on Computer Communications. Washington D. C., USA: IEEE Press, 2018: 198-206.

[14] FAZIO M. Open issues in scheduling microservices in the cloud[J]. IEEE Cloud Computing, 2016, 3(5): 81-88.

[15] MA W J, LIU Y, TANG X G. A dynamic programming approach for optimal signal priority control upon multiple high-frequency bus requests [J]. Journal of Intelligent Transportation Systems, 2013, 17(4): 282-293.

[16] NIKOLAOS A, HJALMTYSSON G. System, method and apparatus for network service load and reliability management[S]. USA Patent: 6, 760, 775, 2004-06-06.

[17] BIRRELL A D, NELSON B J. Implementing remote procedure calls [J]. ACM Transactions on Computer Systems, 1984, 2(1): 39-59.

[18] DUSIA A, YANG Y, TAUFER M. Network quality of service in docker containers [C]//Proceedings of IEEE International Conference on Cluster Computing. Washington D. C., USA: IEEE Press, 2015: 527-528.

[19] RAJKUMAR B, RAJIV R, CALHEIROS R N. Modeling and simulation of scalable Cloud computing environments and the CloudSim toolkit: challenges and opportunities[C]// Proceedings of International Conference on High Performance Computing & Simulation. Leipzig, Germany: [s. n.], 2009: 1-11.

[20] VICTOR M. Adaptive application scheduling under interference in kubernetes[C]//Proceedings of the 9th IEEE/ACM International Conference on Utility and Cloud Computing. Washington D. C., USA: IEEE Press, 2016: 426-427.

编辑　索书志