基于拥塞博弈的微服务运行时资源管理方法

罗睿辞[1],叶 蔚[2],刘学洋[2],孙基男[2],张世琨[2]
(1.北京大学信息科学技术学院,北京 100871;2.北京大学软件工程国家工程研究中心,北京 100871)

摘 要: 随着云计算技术的不断发展,微服务体系结构逐渐成为一种广泛应用的软件设计风格.在基于微服务的应用系统中,微服务数量众多、相互依赖关系复杂、持续在线演化等特征使得微服务运行时资源的有效管理面临新的挑战.本文充分考虑微服务之间的关系特征,提出了一种基于拥塞博弈理论的运行时资源管理方法.首先,对微服务之间的复杂依赖关系进行建模,给出了带权有向无环图描述的微服务调用关系模型;然后,基于微服务关系调用模型对各个微服务的请求到达频率进行计算,并用排队论中的M/G/1队列刻画微服务处理请求的过程,进而设计了一种以服务等级协议(Service Level Agreement)满足程度为衡量标准的服务收益函数;最后利用拥塞博弈模型刻画对计算资源的竞争关系,给出了求解博弈的纳什均衡状态的多项式算法.实验表明,该方法在计算资源有限的场景下可以有效地提高微服务应用的整体性能.

关键词: 微服务体系结构;资源管理;博弈论
中图分类号: TP311 **文献标识码:** A **文章编号:** 0372-2112 (2019)07-1497-09
电子学报URL: http://www.ejournal.org.cn **DOI:** 10.3969/j.issn.0372-2112.2019.07.013

A Runtime Resource Management Approach of Microservices Based on Congestion Game

LUO Rui-ci[1], YE Wei[2], LIU Xue-yang[2], SUN Ji-nan[2], ZHANG Shi-kun[2]
(1. *School of Electronics Engineering and Computer Science, Peking University, Beijing* 100871, *China*;
2. *National Engineering Research Center for Software Engineering, Peking University, Beijing* 100871, *China*)

Abstract: With the continuous development of cloud computing technology, the microservice architecture has gradually become a widely used software design style. In microservice-based applications, different microservices collaborate with one another via interface calls, but they may also compete for limited resources during online evolvement. This poses new challenges for allocating resources efficiently during runtime. To tackle the problem, we propose a novel approach based on congestion game. Firstly, we use a weighted directed acyclic graph to model the inter-relationship of the microservices that compose an application. Then we use M/G/1 queue in queue theory to describe the arrival process of access requests, and combine it with the above graph to calculate the arrival rate of access requests to each microservice, which in turn is used to estimate response time in a newly-designed microservice revenue function. Finally, we define resources competing problem as a congestion game where each microservice is a player aiming to maximize its revenue, and propose an algorithm to find Nash equilibrium in polynomial time. Experiment results show that our approach can effectively improve the overall performance of the system with limited resources.

Key words: microservice architecture; resource management; game theory

1 引言

云计算技术的蓬勃发展不仅促进了计算机软硬件及体系结构的发展,也引发了软件开发和使用方式上的变革.IT资源服务化的思想日益普及,呈现出一切皆服务(X as a Service, XaaS)的趋势,以IaaS、PaaS和SaaS为代表的服务模型已经得到了广泛的应用和实践.在开放、动态、复杂的云计算环境中,软件系统需要持续在线演化[1]以快速响应用户需求,导致软件复杂性日趋增加.考察软件工程中建模和控制软件复杂性的原则、

收稿日期:2018-05-11;修回日期:2018-11-28;责任编辑:李勇锋

方法和技术,其基本思想可以归纳为分治(Separation of Concerns)[2]. 微服务体系结构(Microservice Architecture)正是为了解决云环境下软件复杂性不断增加而被提出的一种基于"分解"的分治方法[3],该方法提倡将应用分解成一系列小服务,每个服务专注于单一业务功能,运行于独立、隔离的环境中,进而使得服务之间边界清晰,并可以采用轻量级通信机制(如 HTTP/REST)进行集成,形成一种高内聚、低耦合的体系结构.

在基于微服务的大规模应用系统中,微服务数量众多、持续在线演化、微服务之间的运行时依赖关系复杂等特征使得微服务运行时的资源管理面临新的挑战. 一方面,微服务之间需要通过协作(通常是相互调用)来完成应用的特定业务功能;另一方面,基于容器的运行时环境特点使得微服务之间存在对计算资源的竞争关系. 考虑微服务之间的协作与竞争关系,本文引入博弈论,将运行时资源管理问题建模为拥塞博弈模型,通过求解博弈的纳什均衡状态进行资源的优化调度.

博弈论已经成功应用于很多计算机资源相关的优化问题,如互联网定价、流量拥塞控制、网络路由优化等[4]. 纳什均衡(Nash Equilibrium)则是博弈论中被应用最广泛的一种问题求解方法:对于博弈的任意参与者,在其他参与者不改变策略的情况下,都无法通过改变自身的策略获取更大的收益,此时所有参与者的策略组合构成纳什均衡状态. 本文将应用的各个微服务作为博弈的参与者,考虑微服务之间的交互关系进而建立博弈的收益函数,给出了一种有效的资源管理方法,其中:(1)建立微服务应用模型,基于运行时微服务的交互信息量化微服务之间的调用关系;(2)将服务等级协议(Service Level Agreement,SLA)的满足程度作为微服务的收益评价标准,基于排队理论和微服务应用模型,建立微服务的收益函数;(3)利用拥塞博弈模型来刻画微服务之间对计算资源的竞争关系,结合服务收益评价标准对资源管理问题建模,给出资源管理优化问题的多项式求解算法.

2 相关研究

云计算环境中的资源规划、调度与管理方法主要分为三类:最优化方法,自适应方法和博弈论方法.

最优化理论是解决云环境下计算资源调度分配的重要方法. 文献[5]采用贪心算法实现服务动态加入和离开情况下最优化部署问题. 文献[6]从计算资源成本的角度入手,以最小化所有背包的成本之和为目标研究动态背包问题,实现服务部署成本最优化. 文献[7]考虑多个 IaaS 的场景,以整体消耗最小为目标,采用随机整数规划方法对虚拟机的部署优化问题进行求解. 最优化理论建模和求解的一般性问题都是 NP-hard 问题,无法在多项式时间内精确求出问题的解,只能在特定领域对问题进行简化或仅仅考虑近似最优解. 人工智能中的启发式方法,如遗传算法[8]、蚁群算法[9]、粒子群算法[10]等在这类最优化问题求解中有着广泛的应用.

自适应方法通过对应用自身运行时数据的监控(Monitor)和分析(Analyze),规划(Plan)应用所需的资源并执行(Execute),结合一个共享的知识库(Knowledge Base)形成经典的 MAPE-K 反馈回路[11]. 自适应方法通常包含两个部分:被管理元素是指自适应软件的应用逻辑,这部分可以在运行过程中被动态的调整;管理元素是指自适应软件的自适应逻辑,这部分通常通过反馈回路对应用逻辑进行调控. 文献[12]考虑云环境下的应用通常会部署于 IaaS 和 PaaS 相结合的混合运行环境下,因此实现了两种虚拟化资源(IaaS 层的虚拟机实例和 PaaS 层的容器)的调度方法,最后通过控制论的反馈回路机制将它们整合在一起. 文献[13]考虑系统调整的延迟性,提出了一种基于控制理论前馈和反馈的虚拟资源动态分配方法. 自适应资源管理方法主要关注对基础设施层面运行时信息的收集和分析,进而对应用的资源进行有效地管理.

由于云环境下应用程序之间的资源竞争行为与经济学中的自由竞争市场类似,基于博弈论的方法能够更加深入地刻画资源管理中的竞争关系. 现有研究通常将和资源相关的角色(IaaS,SaaS 等)建模为博弈中的参与者,每一个参与者通过决策获取相应的资源使得自身收益/成本最优化. 博弈论中最重要并且被应用最广泛的一个概念是纳什均衡(Nash Equilibrium):任意博弈参与者都不可能在其他参与者不改变策略的情况下通过改变自身策略来提升自己的收益. 文献[14]考虑多个 SaaS 提供商将应用运行于同一个 IaaS 中的场景,建立了数学模型来衡量 SaaS 的收益和成本,并给出 IaaS 和 SaaS 在博弈中的决策空间,最后求解纳什均衡来计算 IaaS/SaaS 资源定价/获取的最优方案. 文献[15]使用博弈论来刻画业务层面并行计算任务对资源的竞争特点,考虑优化和公平两个因素,提出了一种基于完成时间和成本双重约束的调度算法,将相互关联的服务在多个不可分割的资源上进行调度,从而形成合理的部署方案. 现有的博弈论方法,大多数都重点关注基础设施(虚拟机)资源的调度,没有在应用层面考虑更细粒度的调度单元以及它们之间的协作关系.

3 场景分析

以基于微服务体系结构风格的社交应用为例,如图 1 所示,每一个方块代表一个微服务,可由独立的团队开发和维护,通过与其他微服务进行交互来实现特定的业务功能.

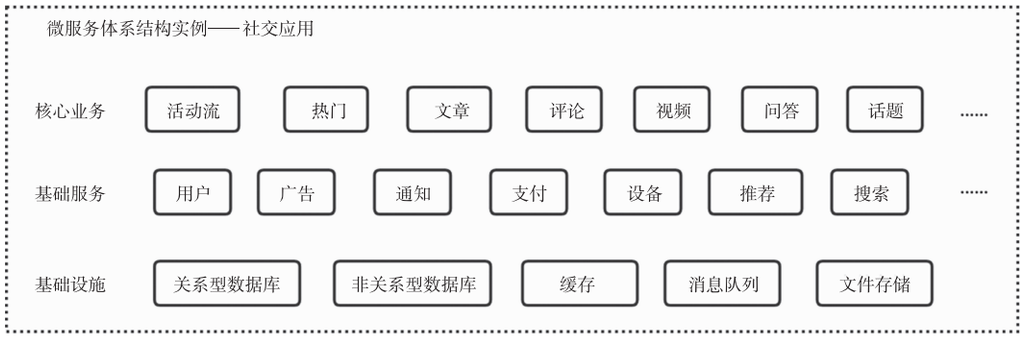


图1　基于微服务体系结构风格的社交应用实例

3.1　微服务部署结构

微服务体系结构实现通常采用基于容器的轻量级虚拟化技术,其中每一个微服务会被打包为可执行的容器单元,运行于一个或者多个 IaaS 虚拟机. 一台虚拟机上运行的服务实例越多,每一个服务实例在单位时间内能够分享到的计算资源就越越少. 微服务之间具有复杂的运行时依赖关系,通过接口调用相互协作完成特定的业务功能.

主流云服务提供商的弹性应用解决方案中,容器化的微服务实现通常通过 API 访问端点(Endpoint)集合向外提供业务能力,当访问量发生变化时,真正实现业务功能的后端计算实例会进行水平伸缩,从而实现运行时资源的动态管理. 一个微服务的内部部署结构如图 2 所示.

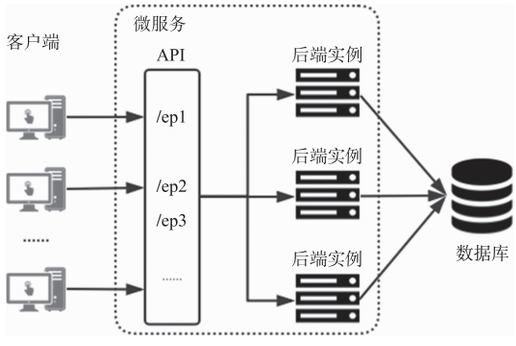


图2　一个微服务的部署结构

图 2 中/ep1,/ep2,/ep3 表示微服务提供的端点(HTTP API 或 RPC 接口),外部对端点的每一次访问请求会被路由至服务的某一个后端实例进行处理.

3.2　微服务收益函数

各个微服务对计算资源的需求、负载以及服务质量的要求不尽相同. 例如:活动流服务为用户展示感兴趣的话题、文章和视频等内容,同时用于投放广告,是对用户体验与应用收入影响最大的服务;用户服务存储着海量的社交关系,对数据分析和推荐系统具有重要意义,但从应用收入的角度来说重要性要低于活动流. 可见,应用的运行时资源管理首先需要从业务场景出发考虑各个服务的重要程度. 此外,服务等级协议的满足程度会影响服务的收益. 其中用户的请求响应时间是衡量满足程度最重要的因素——响应时间越短、SLA 满足程度越高,产生收益(SLA 中约定的收入、用户体验带来的收入等)会越高;响应时间越长、SLA 满足程度越低,产生收益(违约罚款,用户流失等情况)会越低.

因此,针对服务重要性和服务响应时间两个特征,本文对于每一个微服务的一次请求给出如下收益函数:

$$\vartheta_k = \nu_k + m_k \gamma_k \tag{1}$$

其中 γ_k 表示访问请求的响应时间,一次函数的斜率 m_k <0,也就是说当响应时间越短,SLA 的满足程度越高,收益越大;而响应时间越长,SLA 的满足程度越低,收益越小(可能为负值,收益变成惩罚). 针对不同的微服务,可以设置不同的系数以刻画其重要程度. 例如,社交应用中的活动流微服务,对应于较小的一次函数斜率,表示当响应时间越小,其获得的收益会大于其他服务所带来的收益.

3.3　博弈论的引入

微服务为了获取尽可能多的获取收益,需要对有限的计算资源进行竞争,同时又需要与其它微服务以接口调用的方式进行协作. 博弈论中的拥塞模型[16]可用于刻画博弈参与者对资源的竞争关系. 本文将各个微服务作为博弈的参与者,结合微服务自身业务特征和微服务之间的交互关系对参与者的收益进行量化,进而对计算资源的竞争关系用拥塞博弈进行建模,通过求解博弈的纳什均衡状态进行资源的优化调度. 因此,本文将建立以下三个模型:(1)基于运行时微服务的交互信息建立微服务应用模型;(2)基于本节定义的收益函数设计微服务收益模型;(3)给出面向运行时资源管理的微服务博弈模型. 第四节中将详细介绍以上

三个模型.

4 微服务资源管理问题建模

4.1 微服务应用模型

根据上述分析,本文给出基于微服务的应用定义.

应用是一个三元组(S,E,T),其中:

S 是组成应用的微服务集合,对于任意微服务 $s \in S$,存在一个访问端点集合 $E_s = \{e_1, e_2, \cdots, e_n\}$;

E 是所有微服务的访问端点集合,$E = \cup_{s \in S} E_s$;

T 表示端点之间的调用交互关系,$\forall t \in T, t = (e_o, e_q, p), e_o, e_q \in E, p \in \mathbb{R}^+$.

本质上,上述定义可以看成一个有端点集合和调用请求集合构成的有向无环图(DAG),对于图中的每一条边,它关联两个端点表达了调用关系,同时用一个正实数来表达:一次对源端点的访问会级联产生目标端点的访问次数期望:$p < 1$表示只有部分请求会触发该级联子请求(例如程序中出现分支判断的情况),$p \geq 1$ 则表示每一次请求平均情况下多于 1 次子请求会被触发(程序中的循环、分支等情况).

图 3 以一个简单的例子描述了基于微服务的应用结构.

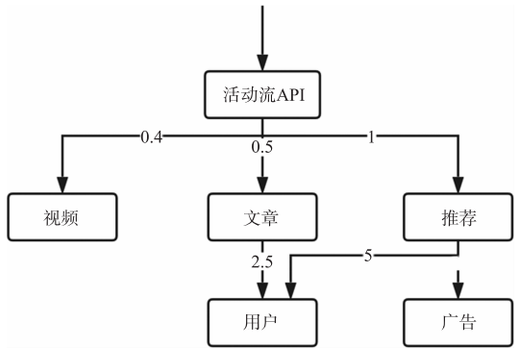


图3 基于微服务的应用结构

表 1 分布式追踪系统中的日志实例

服务名	编号	开始时间	耗时(ms)	方法名	节点 ip	状态
Nginx	0	2018 -05 -28 17:09:12	14	GET	192. 168. 100. 1	OK
API	1	2018 -05 -28 17:09:12	23	GET	192. 168. 100. 15	OK
设备服务	2	2018 -05 -28 17:09:12	1. 08	GetDevice	192. 168. 100. 13	OK
用户服务	3	2018 -05 -28 17:09:12	0. 78	IsBan	192. 168. 100. 12	OK
关系服务	4	2018 -05 -28 17:09:12	0. 348	GetFollower	192. 168. 100. 13	OK
数据打包	5	2018 -05 -28 17:09:12	0. 52	PackUserList	192. 168. 100. 5	OK

图 3 中,一次活动流服务的访问请求,会触发 1 次推荐服务访问,同时还伴随着平均 0.4 次的对视频服务的访问,这是因为并非每次请求都要访问视频服务,例如:只有已经登录的用户才需要访问,尚未登录的用户无需访问,根据数据统计结果,两者的比例是 2:5,所以这条边的权重设置为 0.4. 所以,用户服务的平均访问请求数可以计算为:$0.5 \times 2.5 + 1 \times 5 = 6.25$.

为了获得组成应用的各个微服务的调用关系图,本文基于 Spring Cloud Sleuth[17] 实现了一种微服务体系结构下的基于日志的分布式追踪系统,该系统可以收集应用运行时的服务调用记录.

通过在服务的调用日志中植入用于唯一标识该次请求的属性. 通过收集系统将日志统一收集,可以方便地追踪微服务体系结构下的调用关系,表 1 中展示了典型的调用日志实例.

表 1 的调用日志表达了一次完整的用户请求:客户端的 HTTP 请求通过 Nginx 转发到 API 服务,API 服务调用设备服务、用户服务、关系服务并最后将业务数据打包为客户端所需要的格式并返回. 根据以上日志信息,可以通过对日志的离线处理来自动获取各个服务之间的调用关系,对于从而构造出微服务的关系 T.

4.2 微服务收益模型

微服务在单位时间内的请求数量符合这样的特点:(1)请求数量足够大;(2)单次请求对系统性能和占用资源的影响很小;(3)所有请求到达是独立的. 所以可以将单位时间内微服务的请求到达数用泊松(Poisson)分布描述:

$$P(X = k) = \frac{e^{-\lambda}\lambda^k}{k!} \tag{2}$$

其中,λ 表示单位时间内微服务请求到达数的平均值. 为了应对大量的访问请求,每一个逻辑上的微服务,物理上可能同时运行多个实例(即容器),分布在不同的虚拟机资源上. 假设请求会被均衡到每一个微服务运行实例,我们将微服务实例处理访问请求可以看成一个排队系统,排队系统具有如下特性:

(1)请求的到达时间符合泊松分布——指数概率密度分布;

(2)每一个请求的服务时间相同;

(3)对于同时到达的请求,服务器以以分时调度的策略进行处理.

因此请求访问过程符合 M/G/1 队列[18]条件,对于微服务 i,可以计算每一个请求的平均访问时间 γ 的期望为:

$$E[\gamma_i]=\frac{1}{C\mu_i-\lambda_i} \tag{3}$$

μ_i 为微服务实例的服务速率——单位时间内可以处理的请求数量;

C 是微服务所部署的物理单元(容器)的计算处理能力;

λ_i 是每一个物理单元的请求到达速率,等价于总体请求到达速率和物理单元数量的比值.

对于式(2)(3)中所需的参数 λ_i,则需要根据历史数据进行预测,实际应用场景中通常使用服务的历史访问日志等运行时数据,经过聚合分析后得到服务的到达速率. 结合 4.1 节中定义的微服务模型,$\forall i\in S$,定义到达速率集合 $\Lambda_i=\{\lambda_{ie_1},\lambda_{ie_2},\cdots,\lambda_{ie_n}\}$. 微服务的整体到达速率可以表示为:$\lambda_i=\sum_{e\in E_i}\lambda_{ie}$. 对于任一微服务的请求,都可能级联访问他依赖的微服务,结合微服务模型中的调用关系图 T,本节给出基于 T 的到达速率更新算法,如算法 1 所示.

算法 1　计算微服务到达速率

输入:各个微服务的初始到达速率集合
输出:各个微服务的最终到达速率集合
1. function updatePropagate(value,e)
2. λ_{s_e} += value
3. for $(e_t,p_t)\in\{(e_t,p_t)\mid(e,e_t,p_t)\in T\}$ do
4.　update Propagate($p_t\times$value,e_t)
5. end for
6. end function
7. 初始化微服务最终到达速率为 0
8. for $s\in S$ do
9.　for $e\in E_s$ do
10.　updatePropagate(w_{s_e},e)
11.　end for
12. end for

算法 1 中的 updatePropagate 函数通过遍历微服务关系的带权有向无环图,更新图中各个节点的到达速率.

假设所有微服务运行在有限的资源集合 $R=\{1,\cdots,r\}$ 上,微服务 i 在资源 r 上单位时间内的请求到达速率为 λ_{ir},结合 3.1 节中提出的微服务收益计算函数,单位时间内微服务的从资源 r 上获取的收益如下:

$$d(x_r)=\lambda_{ir}(\nu_i+m_kE[\gamma_k]) \tag{4}$$

带入 $E[\gamma_k]$ 后得到:

$$d_r(x_r)=\lambda_{ir}\left(\nu_i+\frac{m_k}{\frac{C\mu_i}{x_r}-\lambda_{ir}}\right)$$

$$=\lambda_{ir}\nu_i+\frac{m_k}{\frac{C\mu_i}{x_r\lambda_{ir}}-1} \tag{5}$$

这里我们用 x_r 表示选择资源 r 的微服务总数,在下一节会将其描述为与决策向量的函数关系. $\frac{\mu_i}{x_r}$表示运行在该资源上的服务平均分享计算资源.

4.3　微服务博弈模型

拥塞博弈用来描述博弈中的参与者共享资源的场景,在这类场景中,每一个博弈的参与者(Player)的通过选择资源(Resource)来最大化自己的收益,资源产生的收益与选择该资源的参与者数量有关——越多参与者选择,每一名参与者从该资源上能够获得的收益越小. 博弈的形式化定义如下:

$M=\{1,\cdots,n\}$ 表示参与者集合,M 是有限集;

$R=\{1,\cdots,r\}$ 表示资源集合,R 是有限集;

Σ_i 表示博弈参与者 i 决策集合,其中 $i\in M$,$\Sigma_i\subseteq R$,即选择一个资源子集. 为方便描述,本文使用 $\Sigma=\prod_{i=i}^{n}\Sigma_i$ 来表示所有参与者的联合决策空间(Joint Strategy Space)以及 $\boldsymbol{\sigma}=(\sigma_1,\sigma_2,\cdots,\sigma_n)$ 来表示所有参与者的决策向量;

$x_r,r\in R$ 表示资源 r 被选择的次数. 对于一个决策向量 $\boldsymbol{\sigma}=(\sigma_1,\sigma_2,\cdots,\sigma_n)$,$x_j(\boldsymbol{\sigma})=\sum_{i=1}^{n}I_{\sigma_i}(j)$,其中 I 是指示函数,表示资源 j 是否属于参与者 i 的决策集合 $I_{\sigma_i}(j)=\begin{cases}1,j\in\sigma_i\\0,j\notin\sigma_i\end{cases}$;

对于每一个资源 $r\in R$,定义收益函数 $d_r:\mathbb{N}\to\mathbb{R}$,描述该资源被选择的次数 x_r 于每个参与者所能获取的收益之间的函数关系,d_r 是关于 x_r 的单调递减函数;

对于决策向量 $\boldsymbol{\sigma}=(\sigma_1,\sigma_2,\cdots,\sigma_n)\in\Sigma$,参与者 i 的总收益 $S_i=\sum_{r\in\sigma_i}d_r(x_r(\boldsymbol{\sigma}))$;

对于拥塞博弈模型,一个策略向量 $\boldsymbol{\sigma}^*=(\sigma_1^*,\sigma_2^*,\cdots,\sigma_n^*)$是拥塞博弈的纳什均衡点当且仅当:

$$\forall i\in M,\forall\sigma_i\in\Sigma_i,$$
$$\sum_{r\in\sigma_i^*}d_r(x_r(\boldsymbol{\sigma}^*))\geq\sum_{r\in\sigma_i}d_r(x_r(\boldsymbol{\sigma}')) \tag{6}$$

将 4.2 节中的微服务收益模型应用于拥塞博弈,对于任意微服务 i,它的非空决策向量 $\boldsymbol{\sigma}=(\sigma_1,\sigma_2,\cdots,\sigma_n)\in\Sigma$ 的收益函数定义如下:

$$S_i(\boldsymbol{\sigma})=\sum_{r\in\sigma_i}d_r(x_r(\boldsymbol{\sigma})) \tag{7}$$

x_r 表示资源 r 被选中的次数. 进一步地对于任意微服务 i:

$$d_r(x_r(\boldsymbol{\sigma})) = \lambda_{ir}\nu_i + \frac{m_k}{\frac{C\mu_i}{x_r(\boldsymbol{\sigma})\lambda_{ir}} - 1} \tag{8}$$

这里假设当资源 r 被 x_r 个微服务选择后,每一个微服务平均分配其计算能力. 可以看出,$d_r(x_r)$ 是关于 x_r 的单调递减函数($m_i < 0$),满足拥塞博弈模型中的条件.

5 博弈算法求解

将微服务资源调度问题转化为拥塞博弈模型后,进一步的,我们将给出寻找拥塞博弈的纳什均衡点的方法. 拥塞博弈的纳什均衡(Nash Equilibrium)的存在性可以通过构造势函数来证明. 证明中势函数构造的方法同样给出了一种通过迭代来寻找纳什均衡状态的方法.

对于 n 个博弈参与者,需要进行 n 次迭代,每一次迭代加入一个新的参与者进入博弈,新加入的参与者基于当前其他竞争对手的最优策略进行策略选择,同时由于新加入的参与者决策后,会导致资源拥挤,其他竞争对手也需要相应的调整自己的策略以寻求收益最大化. 当所有人都不能通过单独改变自己的策略(即其他参与者策略不变)来获取更大收益时,当前迭代就达到了均衡状态. n 次迭代后,便可以求出整个博弈的纳什均衡点.

假设 $\mu_i(\sigma^i,\sigma^{-i})$ 表示参与者 i 的收益函数,其中 σ^i 是参与者 i 的决策变量,σ^{-i} 表示其他参与者 $(1,2,\cdots,i-1)$ 的决策变量集合,也就是 $\sigma^{-i}=(\sigma^1,\sigma^2,\cdots,\sigma^{i-1})$. 当进行第 i 轮迭代时,如果存在一组决策集合 $\sigma_*=(\sigma_*^1,\sigma_*^2,\cdots,\sigma_*^{i-1})$ 满足下列条件:

$$\forall 1 \leqslant j \leqslant i-1, \forall \sigma^j \in \Sigma_j, \quad \mu_j(\sigma^j,\sigma_*^{-j}) \leqslant \mu_j(\sigma_*^j,\sigma_*^{-j}) \tag{9}$$

即每一个参与者无论怎么样改变策略,只要其他参与者不改变策略,都无法获得更大的收益,那么则说明这 $i-1$ 个参与者达到了纳什均衡状态. 在这种状态下,算法 2 给出了如何加入第 i 个新的参与者,使得 i 个参与者仍然保持纳什均衡状态的算法.

算法 2 计算纳什均衡状态

输入:$n-1$ 个博弈参与者的最优决策向量 $\boldsymbol{\sigma}(n-1)=(\sigma_1,\sigma_2,\cdots,\sigma_{n-1})$

输出:加入参与者 n 后,n 个博弈参与者的最优决策向量 $\boldsymbol{\sigma}(n)=(\sigma_1,\sigma_2,\cdots,\sigma_n)$

```
1. 基于 σ(n) 计算参与者 n 的最优策略 σ_n^*
2. σ(n) = σ(n-1) + σ_n^*
3. i = 0
4. while true do
5.     if i = = n then break
6.     end if
7.     if
           ∀σ_i ∈ Σ_i, S_i(σ(n)) ≥ S_i(σ_i, σ_{-i}(n)) then
8.         i = i + 1
9.     else
10.        计算参与者 i 的最优策略 σ_i^*
11.        σ(n)[i] = σ_i^*
12.        i = 0
13.    end if
14. end while
```

根据算法 2,最多经过 $n \times v$ 次迭代可以到达纳什均衡状态,v 表示选择资源的博弈参与者数量. 对于 n 个参与者,只需要调用 n 次算法 1 就可以找到 n 个参与者进行拥塞博弈的纳什均衡状态,算法的时间复杂度为 $O(n^2v)$.

6 实验分析

6.1 实验环境

本节基于开源项目 Pwitter[19] 进行实验. Pwitter 是仿照 Twitter 产品实现的社交应用,是一个 Python 开发的三层应用,运行在 Gunicorn 应用服务器上,Redis 和 MySQL 用于存储数据. 本文对 Pwitter 进行了扩展:(1)将运行于单一进程中的 Pwitter 应用依据微服务体系结构风格,拆分为多个微服务,如:用户服务、账户服务、文章服务、关注关系服务、收藏服务、计数服务等;(2)扩展 Pwitter 功能,增加了活动流服务、视频服务、推荐服务、搜索服务、通知服务、私信服务等等.

Pwitter 经过微服务化后,由 30 个相互关联的微服务组成,每一个微服务暴露 10 ~ 20 个基于 HTTP 的 API 端点. 这些微服务运行在阿里云提供的虚拟机实例上,实例规格如表 2 所示.

表 2 实验环境使用的虚拟机规格描述

实例规格	ecs. c5. x2large
操作系统	CentOS 7. 4 64 位
CPU	8 核 2. 5GHz
内存	16G
硬盘	50G
公网带宽	10Mbps

表 2 中,每台虚拟机实例上都安装了 Docker 应用程序,服务以 Docker 容器的方式运行在虚拟机实例中. 服务通过 Docker 官方提供的集群管理工具 Swarm 进行管理,不同的虚拟机实例可以分为 Manager 和 Worker 两类节点,Manager 节点负责服务发现、服务管理、服务调度等管理功能,Worker 节点负责服务容器的实际运行环境. 部署服务时,我们可以只需要向 Manager 发起请求,Manger 会根据当前的资源管理算法的调度结果,

将服务的运行时容器放置在合理的虚拟实例上,以满足微服务收益最大化.

6.2　实验过程

使用 Docker Swarm 进行集群资源调度时内置了三种调度策略:

(1) Spread　默认策略,有限选择占用资源(如 CPU,内存)最少的节点,保证节点资源均匀使用;

(2) Binpack　与 spread 相反,尽可能填满一个节点,保证有更多的空余节点;

(3) Random　随机选择节点运行容器.

为了验证拥塞博弈调度策略的性能,本节分别以 Spread、Binpack 以及拥塞博弈三种调度策略对 Pwitter 的 30 个微服务进行部署,部署完成后,模拟发送访问请求,请求频率从每秒 500 请求逐步提升,直至每秒 5000 请求. 在请求发送过程中,本节讲记录请求的平均响应时间、失败比例,每台机器的 CPU 使用率、内存使用率以及服务实例总数,通过这些指标的对比分析,验证不同策略的性能.

同时,由于微服务数量众多,每一个微服务还包含 10 ~ 20 个访问端点,为了完整覆盖所有微服务的所有端点,我们使用基于日志的分布式追踪系统中的日志信息,从在线运行的生产环境的日志中提取各个微服务的调用链,解析历史请求的上下文信息并恢复请求并进行请求重放——再次对系统发起完全相同的请求,这个过程中针对业务逻辑我们会自动地修改一些请求信息,例如对于用户注册请求,从日志中提取出来的请求信息如果重放后会出现失败的情况(用户名、手机号、邮箱重复)从而无法达到覆盖整条服务调用链路的目的.

为了验证不同策略在不同资源条件下的效果,本文分别在 5 台、10 台、15 台、20 台、30 台虚拟机上进行了重复了上述实验.

6.3　实验结果

6.3.1　策略性能比较

为了验证不同策略的实际效果,本节从服务的平均响应时间、请求失败比例和服务收益三个方面对参与实验的三种策略进行比较. 图 4 展示了三种策略在每秒 5000 请求下的实验结果对比.

6.3.2　不同级别的服务性能比较

根据前文的描述,不同服务的重要性是有区别的,重要服务的访问性能越高,应用的实际收益也将越大. 这里我们将服务按照重要程度分为 1 级、2 级、3 级 3 个等级,其重要性随着等级的增加而降低. 图 5、图 6 分别展示了不同等级服务使用不同调度策略在每秒 1000、2000、3000、4000、5000 请求访问频率下的平均访问时间和请求失败比例的对比.

如图 5 所示,当请求频率较低时,三种策略的平均响应时间基本相等. 随着请求频率的增加,三个级别服务的平均请求时间都越来越高,但是,Spread 和 Binpack 策略下三个级别服务的增长基本没有规律,有时会出现 1 级服务请求响应时间高于 3 级微服务请求响应时间的情况,而拥塞博弈策略下,重要程度高的服务始终保持着较低的平均响应时间. 当请求频率达到每秒 5000 次,Spread 和 Binpack 策略下各级服务的平均响应时间时间接近 4000ms,并且 Binpack 策略下 1 级服务的平均响应时间高达 4600ms,而拥塞博弈策略下 1 级服务的平均响应时间仅为 1600ms,远远低于其他两种策略.

如图 6 所示,与平均响应时间类似,当请求频率较低,三种策略的请求失败比例较为接近. 当请求频率大于每秒 3000 次时,开始出现大量的失败请求. 当请求频率为每秒 3000 次,Binpack 和 Spread 策略下 1 级服务的失败请求比例分别为 4.3% 和 2.3%,拥塞博弈策略的请求失败比例最低,为 1.0%. 随着请求频率的增加,Binpack 和 Spread 策略下 1 级服务的失败比例大幅增加,而拥塞博弈策略下 1 级服务的失败比例增加幅度远小于另外两种策略,于此同时,拥塞博弈策略下 1 级服务的失败比例仍然远低于 2 级和 3 级服务,也就是说,拥塞博弈策略在保持整体性能提升的同时,兼顾了重要程度较高服务的性能.

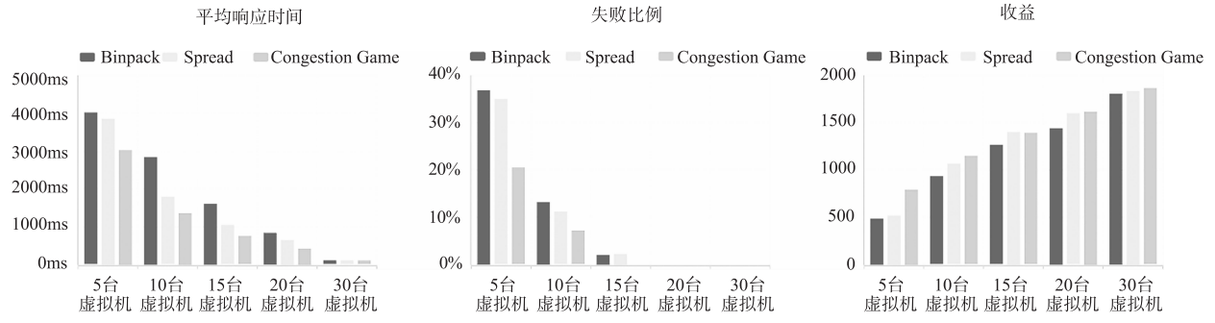


图4　不同策略在不同数量虚拟机下的平均响应时间、请求失败比例和收益对比

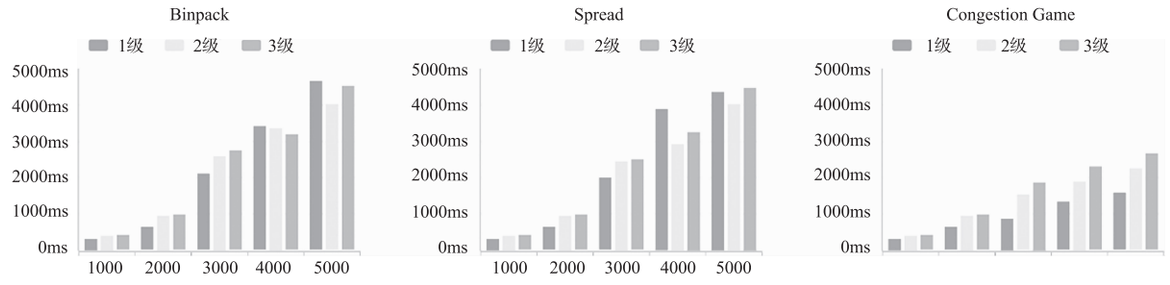


图5 不同等级服务在不同策略下的平均响应时间

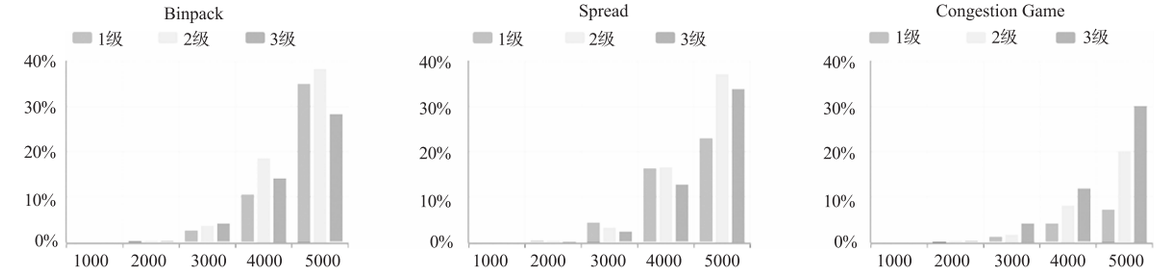


图6 不同等级服务在不同策略下的请求失败比例

6.4 结果分析

经过上述对比可以发现,当虚拟计算实例个数较少时,本文提出的基于拥塞博弈的调度策略优化效果较好,相较于 Docker Swarm 提供的 Binpack 策略,有较大的提升. 而随着虚拟计算实例个数的提升,两种算法的差距越来越小. 增加虚拟计算实例,一方面使得每一个实例的拥挤程度变小,从而增大收益;另一方面,为了使得大量请求的完成时间减小,可以在运行更多的容器个数来保障完成时间.

同时,在拥塞博弈策略下,重要程度较高的服务性能高于重要程度较低的服务. 这是因为拥塞博弈在进行调度时,优先考虑了重要程度较高的服务,也就是说,随着资源越来越少,重要程度高的服务获得资源的可能性高于重要程度低的服务,从而保障了重要程度较高的服务在访问高峰时期的性能. 这种特性使得应用在服务器资源紧张时可以获得更高的收益.

7 总结

本文针对云环境下的微服务应用内部资源竞争的场景,提出了一种基于博弈的资源管理方法. 首先,根据微服务的特点,提炼了基于微服务的应用模型,描述刻画了组成应用的微服务及其相互之间的依赖关系;其次,针对微服务内资源竞争的场景,提出了基于负载的微服务收益估计方法以及拥塞博弈模型;最后,考虑竞争关系,使用博弈论纳什均衡点的求解算法来解决应用整体收益的最优化问题,并实现了原型来验证算法的效果,根据实验结果可以看到,本文提出的算法在计算资源有限的情况下能够较好地提升应用的整体性能.

参考文献

[1] 王怀民,史佩昌,丁博,尹刚,史殿习. 软件服务的在线演化[J]. 计算机学报,2011,34(02):318 – 328.

[2] 梅宏,黄罡,张路,张伟. ABC:一种全生命周期软件体系结构建模方法[J]. 中国科学:信息科学,2014,44(05):564 – 587.

[3] Dragoni N,Lanese I,Larsen S T,et al. Microservices:How to make your application scale[A]. International Andrei Ershov Memorial Conference on Perspectives of System Informatics[C]. Springer,Cham,2017. 95 – 104.

[4] Altman E,Boulogne T,El-Azouzi R,et al. A survey on networking games in telecommunications[J]. Computers & Operations Research,2006,33(2):286 – 311.

[5] Stolyar A L,Zhong Y. An infinite server system with general packing constraints:Asymptotic optimality of a greedy randomized algorithm[A]. The 51st Annual Allerton Conference on Communication,Control,and Computing (Allerton)[C]. US:IEEE,2013. 575 – 582.

[6] Li Y,Tang X,Cai W. Dynamic bin packing for on-demand cloud resource allocation[J]. IEEE Transactions on Parallel and Distributed Systems,2016,27(1):157 – 170.

[7] Chaisiri S,Lee B S,Niyato D. Optimal virtual machine placement across multiple cloud providers[A]. 2009 IEEE Asia-Pacific Services Computing Conference (APSCC)[C]. US:IEEE,2009. 103 – 110.

[8] Kansal S, Kumar H, Kaushal S, et al. Genetic algorithm-based cost minimization pricing model for on-demand IaaS cloud service[J]. The Journal of Supercomputing, 2018:1-26.

[9] Gupta A, Garg R. Load balancing based task scheduling with ACO in cloud computing[A]. 2017 International Conference on Computer and Applications (ICCA)[C]. US:IEEE, 2017. 174-179.

[10] Sheikholeslami F, Navimipour N J. Service allocation in the cloud environments using multi-objective particle swarm optimization algorithm based on crowding distance[J]. Swarm and Evolutionary Computation, 2017, 35:53-64.

[11] Hoenisch P, Schulte S, Dustdar S, et al. Self-adaptive resource allocation for elastic process execution[A]. 2013 IEEE Sixth International Conference on Cloud Computing[C]. US:IEEE, 2013. 220-227.

[12] Baresi L, Guinea S, Leva A, et al. A discrete-time feedback controller for containerized cloud applications[A]. Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering[C]. US:ACM, 2016. 217-228.

[13] 俞岭,谢奕,陈碧欢,等. 基于前馈和反馈控制运行时虚拟资源动态分配[J]. 计算机研究与发展,2015,52(4):889-897.

[14] Ardagna D, Ciavotta M, Passacantando M. Generalized-Nash equilibria for the service provisioning problem in multi-cloud systems[J]. IEEE Transactions on Services Computing, 2017, 10(3):381-395.

[15] Wei G, Vasilakos A V, Zheng Y, et al. A game-theoretic method of fair resource allocation for cloud computing services[J]. The Journal of Supercomputing, 2010, 54(2):252-269.

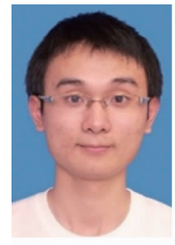
[16] Milchtaich I. Congestion games with player-specific payoff functions[J]. Games and Economic Behavior, 1996, 13(1):111-124.

[17] Carnell J. Spring Microservices in Action[M]. Manning Publications Co, 2017.

[18] Bolch G, Greiner S, De Meer H, et al. Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications[M]. John Wiley & Sons, 2006.

[19] Remi Cadene, David Panou. LI328-Simpler Twitter with JavaEE, JQuery, MySQL, MongoDB [OL]. https://github.com/Cadene/Pwitter, 2018.

作者简介



罗睿辞　男,1988 年生于湖南株洲. 北京大学信息科学技术学院博士研究生,主要研究领域为基于 Web 的软件工程和软件体系结构.

叶　蔚　男,1985 年生于江西全南. 北京大学软件工程国家工程中心副教授. 主要研究领域为基于 Web 的软件工程、软件体系结构、应用集成等.
E-mail:wye@pku.edu.cn

刘学洋　男,1978 年生于湖北钟祥. 北京大学软件工程国家工程研究中心副教授,主要研究领域为软件工程、信息安全等.
E-mail:liuxueyang@pku.edu.cn