doi:10.3772/j.issn.1002-0470.2019.01.001

基于服务能力模型的微服务弹性资源供给机制①

刘晓东②* 赵晓芳③* 陈雅静** 陈继红**

(*中国科学院计算技术研究所 北京 100190)

(**中国石油集团东方地球物理公司物探技术研究中心 涿州 072751)

摘 要 针对实时性要求高、服务请求响应时间敏感的应用场景,为解决在满足服务请求响应时间要求的基础上减少资源使用量的难题,本文提出了基于服务能力模型的弹性资源供给算法(SCMARP)。为建立准确的服务能力模型,提出了基于非线性回归的服务能力模型。在资源弹性供给阶段,SCMARP 算法根据当前服务请求负载、服务请求范式和已建立的服务能力模型,提前按需触发伸缩动作,从而减少违反请求响应时间服务等级协议(SLA)的数量和资源使用量。实验结果表明,与经典的基于阈值弹性资源供给算法(TARP)相比,在满足平均请求响应时间的基础上,平均资源使用数减少 6%,服务请求超时失败率降低 10.7%。

关键词 微服务,弹性资源供给,服务能力模型,非线性回归

0 引 言

随着新业务生态如敏捷开发、持续交付、DevOps 以及容器技术的不断发展和成熟,催生出新的应用架构——微服务架构(microservices architecture, MSA)[1,2]。MSA 自 2014 年开始流行,已经成为工业界和学术界研究的热点,被业界公认为云计算时代互联网应用的主要构建方式[1,3]。国内外大型互联网公司,包括阿里巴巴、华为、腾讯、Netflix、Amazon、Uber 等,均采用微服务架构搭建企业级应用。微服务架构采用一套微服务来构建应用,通过轻量级通信机制实现服务互联,服务基于业务功能构建,可以使用不同的编程语言和数据存储技术,独立开发、独立扩展、独立部署。Gartner 的研究表明,截止到 2017 年有超过 20% 的大型机构通过部署微服务以提高应用系统的灵活性和可扩展性。

构建业务应用的软件架构的不断发展,使得业务个体(服务、组件、模块等)对 IT 资源纵向升级(scale-up)的要求降低,但横向伸缩(scale-out)需求增加[4]。弹性云服务的目标是能够根据服务请求的负载大小,快速、准确地扩容或缩容,以保障服务质量,避免违反服务等级协议(service level agreement, SLA),同时减少资源浪费[4,5]。云计算的弹性,促使业务应用向云上转移,云计算为高扩展性的业务应用提供了合适的运行平台[6]。

随着用户服务请求的不断增长,单一的服务实例已经远远不能满足用户对服务质量(quality of service, QoS)(请求响应时间、吞吐量等)的需求[6]。通过部署多服务实例构建服务集群,以聚合的方式对外提供服务,是目前服务提供商应对高负载服务请求的主要解决方式。多个微服务实例提供相同的服务功能,服务请求可以在任意一个服务实例中得到处理。

对服务请求者来说,服务请求的响应时间是最重要的性能指标之一[7],尤其是对实时性要求高、

① 国家重点研发计划(2016YFB0201500)和国家自然科学基金(61202413)资助项目。

② 男,1989 年生,博士生;研究方向:云计算,高性能计算;E-mail: liuxiaodong@ict.ac.cn

③ 通信作者,E-mail: zhaoxf@ict.ac.cn

(收稿日期:2018-04-26)

服务请求响应时间敏感的应用场景(如在线查询)。但是,如何在满足服务平均请求响应时间的基础上,减少资源使用量是一个难题。这是由于互联网环境下的服务请求负载具有自相似特性、突发性和差异性,因此很难预测未来服务请求负载[3,8,9]。资源分配过多,如按照负载峰值时对资源的需求进行微服务部署,则在非峰值阶段会造成大量的资源浪费;而资源分配过少,如按照平均负载时对资源的需求进行微服务部署,则在负载峰值发生时造成服务过载,服务请求的排队等待时间增长,无法满足请求响应时间的 SLA。服务的弹性伸缩能力是应对该难题的关键,但如何快速、准确地伸缩以应对突发性的服务请求负载是一个研究热点和难点[10-12]。

本文提出了一种基于服务能力模型的弹性资源供给算法(service capacity model based auto-scaling of resource provisioning, SCMARP)。同时,借助容器技术具有的秒级资源供给机制,实现微服务的快速部署。为建立准确的服务能力模型,提出了基于非线性回归的服务能力模型算法(nonlinear regression based service capacity model, NRSCM),该算法通过在线学习的方法建立“微服务-容器”组合在特定服务请求范式下范式率与请求响应时间的服务能力模型。在资源弹性供给阶段,SCMARP 算法根据服务请求负载、服务请求范式和已建立的服务能力模型,得到期望的容器数。然后与当前正在运行的容器数进行比较,提前按需触发伸缩动作。大量的实验证明 SCMARP 算法能够快速、准确地进行资源伸缩,从而减少违反请求响应时间 SLA 的数量和资源使用量。

1 相关工作

目前,工业界和学术界对弹性资源供给算法进行了大量的研究[4,13],有多种实现弹性的方案,从方法上主要分为 2 类。

(1) 横向伸缩:也称为复制,是指通过增加/减少实例来实现伸缩,其中实例包括物理机、虚拟机、容器、服务实例等。资源提供者需要在负载上升时增加实例以避免 SLA 违约,在负载下降时减少实例以提高资源利用率。

(2) 纵向伸缩:是指通过增加/减少计算资源来实现伸缩,计算资源包括 CPU 核、内存等。从实现的方法上包括调整和替换,调整是指在线增加/减少计算资源,替换是指通过启动新的资源来替换掉之前的资源。

从何时触发伸缩动作以及如何计算伸缩的容量上进行分类,分为被动式和主动式。

(1) 被动式:根据当前负载和性能指标(如 CPU 利用率,SLA 等)判断是否超过制定的阈值,如果是,则触发伸缩动作。由于该方法基于当前值而不是预测值进行触发,因此依赖监控器定期的收集性能指标,以作为弹性伸缩的依据。

(2) 主动式:通过负载预测模型来计算即将出现的负载,如果当前的资源不满足预测负载对资源的需求,则提前对资源进行扩容。通常情况下,预测算法基于历史负载信息。

被动式的伸缩算法只有在达到触发阈值时才进行伸缩,具有滞后性。主动式的伸缩算法克服了被动式算法的滞后性,但依赖于负载预测模型的准确性,是建立在负载具有一定规律基础之上,限制了其应用范围。目前,被动式、横向伸缩是云计算环境下主流的弹性资源供给方法[13]。

针对云计算环境下的应用,除以上弹性资源供给算法的分类标准,从算法实现上主要分为基于阈值的资源供给算法、基于自适应环的资源供给算法、基于强化学习的资源供给算法等。

(1) 基于阈值弹性资源供给算法(threshold based auto-scaling of resource provisioning, TARP)[14-16] 具有简单、直观的优点,在云提供商中被广泛使用,如亚马逊的 EC2、RightScale 和 Kubernet。阈值包含一个或多个性能指标,如服务请求率、CPU 利用率、平均请求响应时间等。阈值规则包括扩容的上限阈值和缩容的下限阈值。为了避免突发性负载造成的伸缩抖动,在达到阈值之后需要维持一段时间,如当 CPU 利用率超过 80%,且持续 1 min 才进行扩容。除此之外,需要考虑扩容/缩容的梯度,如每次伸缩一个或多个资源。如何设置合理的阈值、持续时间、伸缩梯度,需要对应用及其工作负载有深入的了解

和研究。

(2) 基于自适应环的资源供给算法[17-19]包括4个环节:监视(Monitoring,M)、分析(Analysis,A)、规划(Planning,P)和执行(Execution,E)。在监视阶段,监视器收集资源和云应用的运行状态信息;分析阶段利用收集到的状态信息对即将出现的工作负载及资源需求进行预测和分析;规划阶段做出资源供给的伸缩决策;在执行阶段,执行伸缩决策。知识库(Knowledge,K)是自适应环所需要的知识基础,被以上4个环节所共享和更新。通过定期的执行自适应环MAPE,来适应动态变化的工作负载。

(3) 基于强化学习的资源供给算法[20,21],将伸缩决策建模为马尔科夫决策过程。使用强化学习算法在给定的马尔科夫决策过程中寻找最优策略,通过智能体与其环境的直接交互来学习,不需要任何的先验知识,即基于给定的工作负载、性能指标等变量,找到最优的伸缩动作。在执行完伸缩动作之后,资源提供者会得到一个回报值(如请求响应时间缩短),决策过程的目标是选择最佳动作,使得回报值最大。但实际上,该算法有两大不足:较差的初始化表现;长时间的训练过程和大规模的状态空间。因此,对于SLA(如请求响应时间、吞吐量等)要求严格的应用场景,基于强化学习的资源供给算法将导致大量的SLA违约。

2 微服务弹性资源供给机制

2.1 微服务弹性资源供给机制的系统架构

本文提出的弹性资源供给机制包括监控(Monitor)、建模(Modeling)、伸缩决策(Auto-scaling Maker)和执行(Exector)4个环节,简称MMAE。容器云环境下的微服务弹性资源供给机制的系统架构如图1所示。

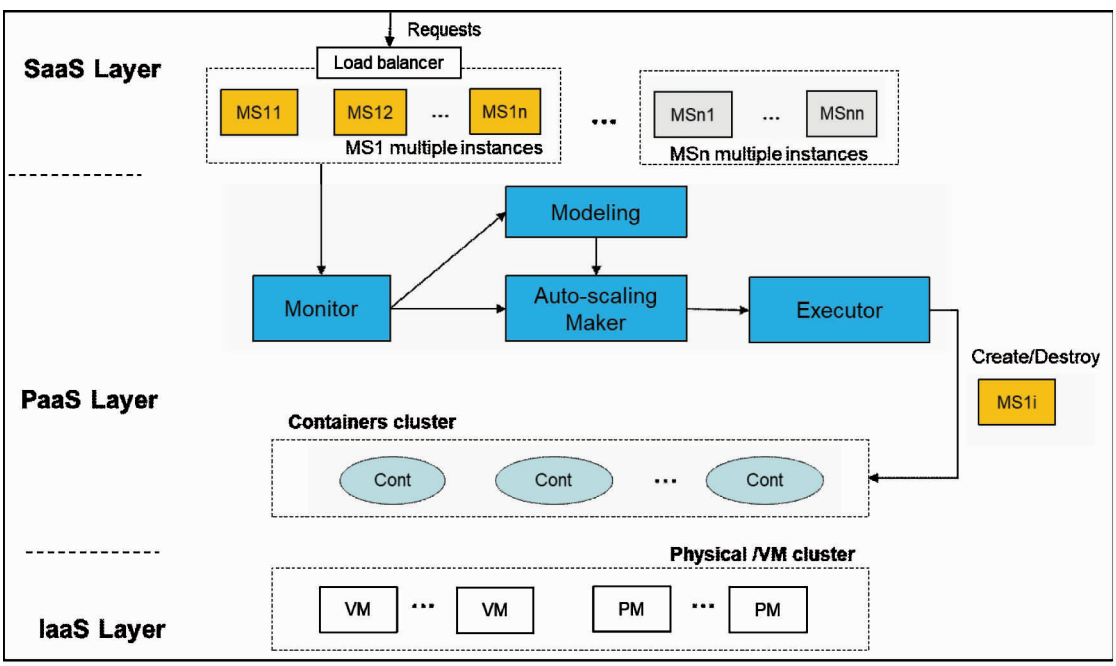


图1 微服务弹性资源供给机制的系统架构

(1) 监控

对于每一个微服务实例集群,分别收集微服务实例集群的服务请求信息,用于微服务和其运行所在的容器的服务能力建模,以及伸缩决策。基于服务能力模型的弹性资源供给机制需要收集的服务请求信息包括:

聚合服务请求率。一个微服务实例集群(如MS1)接收到的各个服务请求类型的请求率 $\boldsymbol{\lambda} = <\lambda_1, \lambda_2, \cdots, \lambda_N>$。服务请求经过负载均衡器(Load balancer)被均衡地分发到微服务实例。$\boldsymbol{\lambda}$ 被

发送到伸缩决策模块,作为当前微服务集群的负载。

参考服务实例的请求率与平均请求响应时间。默认情况下,一个微服务实例集群下的容器具有相同的服务能力,因此参考实例可以是其中的任意一个实例,如 MS11。由于一种微服务提供多种类型的服务请求 APIs,对于请求类型感知的服务能力模型,要求监控模块收集所有服务请求类型的请求率 $\boldsymbol{\alpha} = <\alpha_1, \alpha_2, \cdots, \alpha_N>$ 以及平均请求响应时间 τ。样本 $<\boldsymbol{\alpha}, \tau>$ 被发送到建模模块,用于构建“微服务-容器”的服务能力模型。

(2) 建模

利用监控模块收集的样本数据,通过在线学习的方法构建“微服务-容器”的服务能力模型。服务能力模型是弹性资源供给机制的基础,其模型的准确性、稳定性将直接影响弹性伸缩效果(包括请求响应时间和资源使用数)。

(3) 伸缩决策

实现基于服务能力模型的弹性资源供给算法。基于平均请求响应时间的 SLA,结合当前的请求负载、服务能力模型,进行服务容量规划,得到期望的服务实例数。然后,与当前服务实例数比较,做出伸缩决策。

(4) 执行

执行伸缩决策,包括创建新的服务实例-容器、下线正在运行的服务实例-容器。在执行创建上线决策时,资源供给模块选择一个虚拟机或物理机启动容器,容器启动后微服务会自动进行服务注册,注册完成之后负载均衡器会发现新的服务实例,从而将其加入可用服务实例列表。执行下线决策需要容器资源供给模块和负载均衡器的配合,负载均衡器收到下线通知之后,停止将服务请求分发到即将下线的服务实例,当即将下线服务实例完成所有服务请求时,资源供给模块才能销毁容器。

2.2 请求范式

一种微服务提供多种类型的服务请求 APIs,不同类型的服务请求具有不同的服务响应时间和资源需求。服务请求类型的差异性将会直接影响平均请求响应时间,例如有两种服务请求:请求 1 和请求 2,服务请求 1 的服务响应时间为 10 ms,服务请求 2 的服务响应时间为 30 ms。当两种请求类型的请求率为 3:1 时,其平均响应时间为 15 ms,而当请求率为 1:3 时,其平均响应时间为 25 ms。通过这个简单的例子可以看出,请求类型和请求率的比值(也称请求模式)对平均请求响应时间的影响很大。因此,基于平均请求响应时间的服务能力模型能够感知请求类型和请求模式的差异性。

由于服务请求具有自相似性,每秒到达的服务请求千差万别,因此请求模式数非常大。假设有 m 种服务请求类型,请求率范围为 $[0, n]$(请求/s),则所有可能的请求模式数为 n^m。如果为每种请求模式建立其服务能力模型,则服务能力模型变为一个包含 n^m 条记录的查询表。即使对于普通的服务场景(如 $m = 4, n = 1000$),查询效率将成为弹性伸缩的性能瓶颈,不能满足实时性的要求。

为提高服务能力模型的查询效率,提出了请求范式的概念,即请求模式的标准化。其基本思想是将请求模式进行分类,每一类对应一个请求范式。将 n^m 个模式数分为 n^χ 个类,其中 $\chi << m$,称为模式差异度。设 m 种服务请求类型,请求率为 $\boldsymbol{\alpha} = <x_1, x_2, \cdots, x_m>$,则对应的请求范式 $\boldsymbol{\sigma}$ 为

$$\begin{aligned} &\boldsymbol{\sigma} = \boldsymbol{\alpha}/\delta \\ &\delta = \max(\boldsymbol{\alpha}) / \chi \end{aligned} \tag{1}$$

其中 δ 称为范式率。如 $\chi = 4, \boldsymbol{\alpha} = <10, 20, 30, 40>$,则 $\delta = 40/4 = 10, \boldsymbol{\sigma} = <1, 2, 3, 4>$。

基于请求范式的服务能力模型的定义是:对于“微服务-容器”组合,在特定请求范式下,范式率与平均请求响应时间的关系模型。

通过模式差异度 χ 进行请求模式的标准化,可以有效地降低请求模式数,但在标准化过程中会带来误差,如 $\chi = 4, \boldsymbol{\alpha} = <1, 2, 3, 5>$,则 $\delta = 1.25$,标准化之后 $\boldsymbol{\sigma} = <1, 2, 2, 4>$。因此,需要根据具体的服务场景,在查询效率和精准度之间权衡。

2.3 基于请求范式的服务能力模型

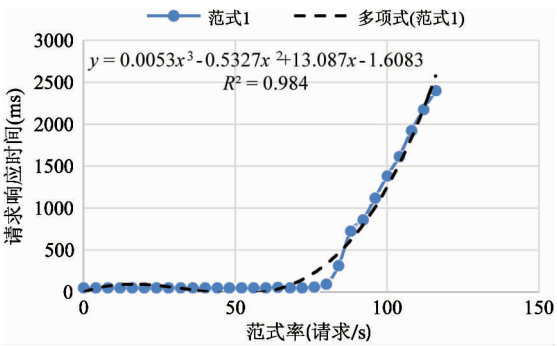
对于一种请求范式,范式率越大,平均响应时间越长。为深入研究范式率与平均请求响应时间的正比关系,设计了如下实验。

- 微服务提供 4 种服务请求类型,服务时间均为 50 ms,但具有不同的 CPU 使用率[1%, 30%,

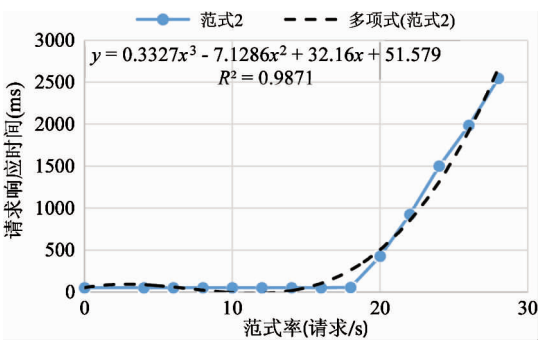
60%, 100%],微服务内有 4 个工作线程处理服务请求,微服务运行在 1 CPU @2.4 GHz 1 GB 内存的节点上。

- 为更直观地说明不同请求范式对服务能力模型的影响,设置了 3 种典型的请求范式:[1, 0, 0, 0],[0, 0, 0, 1]和[1, 1, 1, 1]。
- 对于每种请求范式,范式率不断增长直到平均请求响应时间达到 2 s。对于 1 个范式率,客户端连续调用 10 s。

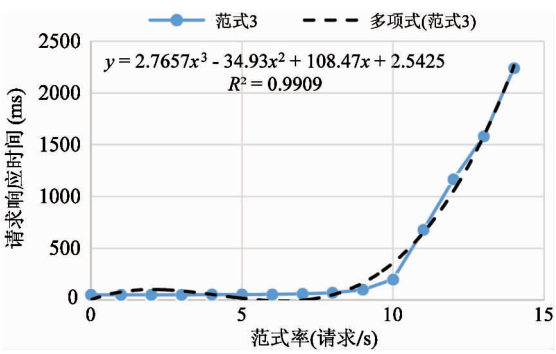
范式率与平均请求响应时间的实验结果如图 2 所示。



(a) 范式 1 及其多项式拟合曲线



(b) 范式 2 及其多项式拟合曲线



(c) 范式 3 及其多项式拟合曲线

图 2　范式率与平均请求响应时间的关系

从图 2 可以看出:(1)对于每 1 种请求范式,当范式率小时,平均请求响应时间基本保持不变,3 种请求范式的平均请求响应时间均在 50 ms 左右。当范式率超过一定值(分别为[80, 16, 9])之后,平均请求响应时间随范式率迅速增长。(2)通过对比 3 种请求范式可以得到,当平均请求响应时间达到 2 s 时,3 种请求范式的范式率分别为[110, 26, 13],不同请求范式下范式率与平均请求响应时间的关系相差很大,即服务模型受请求范式的影响很大,因此说明基于请求范式感知的服务能力模型的有效性。

基于请求范式的服务能力模型的仿真实验结果表明,平均请求响应时间随范式率成非线性增长。因此,本文将服务能力模型定义为非线性回归问题,更确切的是一元非线性回归问题,并通过机器学习的方法解决此问题。非线性回归模型的算法复杂度低,模型建立迅速,非常适合实时性要求高的在线应用场景,如微服务弹性资源供给。

根据图 2 中数据点的分布规律,范式率与平均请求响应时间的关系可以用一元多项式函数表示:

$$Y = f(x) = \alpha_0 + \alpha_1 x + \alpha_2 x^2 + \cdots + \alpha_n x^n \quad (2)$$

其中,Y 为平均请求响应时间,x 为范式率,n 为多项式的度,α_i 为 i 次项的系数。

一元非线性回归,也称一元曲线回归,是将有非线性关系的两个随机变量进行适当的变换,转化成线性关系的一类回归分析。通常称为化曲线为直线的回归问题,其转化过程主要包括如下步骤。

步骤 1 确定变量与因变量之间的内在关系的函数类型。常用的函数类型有幂函数,多项式函数,指数函数,抛物线函数,对数函数等。在选择函数类型时有 2 种方式:第 1 种是通过理论推导,确定函数类型。第 2 种是根据实验数据的散布图,选择适当的曲线来拟合实验数据,如果拟合效果好,则用作函数类型。

步骤 2 通过变量替换,化曲线方程为直线方程,然后用线性回归方法进行求解。

步骤 3 还原到原来的变量,即可得到所要求的一元非线性回归方程。

需要注意的是回归方程的适用范围,一般情况下只局限于已观测数据的变动范围,而不能随意外

推。

为在线建立服务能力模型,提出了基于非线性回归的服务能力模型算法 NRSCM,该算法基于 scikit-learn library。训练集包含 n 条样本,一条样本数据包含 N 个特征向量以及对应的平均请求响应时间,样本数据格式 $<\boldsymbol{\alpha}, \tau>$,其中 $\boldsymbol{\alpha} = <x_1, x_2, \cdots, x_N>$ 表示对应 N 种请求类型的请求率。

训练集是包含多种请求范式的样本集合,因此需要按照请求范式分成多个子训练集,然后分别对子训练集进行训练,得到不同请求范式下的服务能力模型。在建立服务能力模型时,首先要解决的问题是如何确定一元多项式函数的度,多项式度数太小则曲线拟合效果差,度数太大则增加服务能力模型的复杂性。为此,提出了基于决策系数 R^2 的多项式度数求解方法,其基本思想是在保障曲线拟合程度满足决策系数要求的前提下,最小化多项式度数,从而简化模型的复杂度。决策系数 R^2 的计算公式如下:

$$R^2(y,\hat{y}) = 1 - \frac{\sum_{i=0}^{i=N_{sps}} (y_i - \hat{y}_i)^2}{\sum_{i=0}^{i=N_{sps}} (y_i - \bar{y}_i)^2} \tag{3}$$

其中,y_i 是真实值,$\hat{y}_i$ 是预测值,$\bar{y}$ 是所有真实值的平均值。R^2 的最高得分是 1,也可能为负,因为建立的模型可能非常差。

对于一个子训练集,服务能力建模的算法描述如算法 1 所示。

算法 1 基于非线性回归的服务能力模型算法 NRSCM

输入:样本数据 $\boldsymbol{Sps} = [<\boldsymbol{\alpha}_1, \boldsymbol{\tau}_1>, <\boldsymbol{\alpha}_2, \boldsymbol{\tau}_2>, \cdots, <\boldsymbol{\alpha}_n, \boldsymbol{\tau}_n>]$,
模式差异度 χ,
决策系数 R^2 的最小值 Min R^2,
多项式度的最大值 MaxDeg,

输出:样本的请求范式 σ,
多项式的度 deg,
多项式系数 coef,

```
From sklearn. linear _ model import LinearRegression
as lr
begin
initialize
deg = 3
X = [ ] Y = [ ]
/ * 假设训练集中仅有一种请求范式,任取一条样本数据即可得到请求范式 * /
δ = max([Sps['x1']][0], Sps['x2']][0], …, Sps['xN']][0]])/χ
σ = [Sps['x1']][0], Sps['x2']][0], … , Sps['xN']][0]]/δ
/ * 请求模式标准化 * /
for i in range(0, len(Sps))
    δ = max([Sps['x1']][i], Sps['x2']][i], … , Sps['xN']][i]])/χ
    y = Sps['τ']][i]/δ
    X. append(δ)
    Y. append(y)
  endfor
/ * 建立服务能力模型 * /
for degree in range(deg, MaxDeg)
    linreg = lr (fit _ intercept = False)
    linreg. fit(X, Y)
    tmpR2 = linreg. score(X, Y)
      if tmpR2 > = MinR2
        coef = linreg. coef _
        deg = degree
        break
      endif
  endfor
end
```

2.4 基于服务能力模型的弹性资源供给算法

弹性资源供给算法在制定伸缩决策时,依赖的输入信息包括当前请求负载、服务能力模型和平均请求响应时间的 SLA。3 个输入信息对应 3 个时间概念:负载统计周期、样本统计周期和平均请求响应时间的 SLA。3 个时间长短的设置与相互之间的关系,对弹性伸缩的效果具有重要影响。

- 负载统计周期

请求负载在服务能力模型中是指服务请求率,而服务请求率是最近一段时间服务请求的平均值,如将 1 s 作为统计周期。如何设置统计周期的长短是一个难点。

统计周期过短,如毫秒级,则统计的服务请求率波动性变大,这是由于服务请求具有的突发性、波动性特征造成的。服务请求率的波动性会直接影响弹性伸缩的波动性,造成资源伸缩决策对当前请求负

载过于敏感。此外,由于服务请求负载的突发性,弹性伸缩的幅度会大大增加。这种高频率、高幅度的资源伸缩不但造成大量的计算开销(服务部署、配置等),而且降低了应用系统的稳定性。而实际上有效的伸缩决策很少,这是由于服务请求负载相对稳定,不会出现如此高频、高幅度的波动。

统计周期过长,如大于 10 s,则统计的服务请求率对请求负载的敏感性降低,加大伸缩决策的滞后性,增加请求响应时间 SLA 违约的数量。被动式的弹性伸缩算法是典型的统计周期过长导致的,直到出现响应时间 SLA 违约才会采取伸缩动作,但该算法的优点是伸缩依据简单直观、伸缩决策准确。

- 样本统计周期

样本统计周期的长短会对服务能力模型的准确性造成影响,这是由于在线采样过程中,采样点之前的请求负载会对采样时刻的请求响应造成影响,例如采用时刻之前微服务内有排队等待的请求,则必然会加长采样时刻请求的响应时间。又如,当服务请求负载超过微服务的服务能力时,无论超过多少,在请求负载保持不变的情况下,随着时间的增长,平均响应时间会不断增长,也就是说对应同一个服务请求率,平均服务响应时间是不同的。

样本统计周期过短,采样点之前的请求负载对采样点请求负载的影响比例增大,导致样本数据误差增大。

样本统计周期过长,由于请求负载具有波动性,以长时间的平均请求率及对应的请求响应时间代表该请求率下的请求响应时间,同样会导致样本数据的误差增大。这是由于平均请求率是所有请求的个数除以统计周期,而平均响应时间是所有请求的响应时间除以请求数,前者与统计周期负相关,而后者无关。因此,对于平均请求率,其平均请求响应时间随着样本统计周期的增大而增长,即服务能力模型曲线上移。

(1) 微服务-容器扩容

通过对负载统计周期、样本统计周期的分析,为提高资源伸缩决策的有效性、服务能力模型的准确性,减少请求响应时间 SLA 违约的数量,对于平均请求响应时间 SLA 是秒级的应用场景(也是目前基本的微服务应用场景,不考虑特殊的应用场景,如服务请求响应时间的 SLA 是毫秒级的),本文对两者之间的关系约束定义如下:

$$样本统计周期 >= 负载统计周期 + 微服务上线时间 \tag{4}$$

其中,微服务上线时间是指从执行伸缩决策开始到微服务提供服务的时间,包括 Docker 容器启动时间、服务注册、环境配置等时间。负载统计周期默认情况下是 1 s,即大多数监控器的负载统计周期,在实际应用场景下,可以根据具体的请求负载情况进行调整,周期变长能提高伸缩决策的有效性,但与之矛盾的是增加请求响应时间 SLA 违约的数量,因此需要权衡有效性和 SLA 违约。

关系约束定义的依据是服务能力的建模根据样本统计周期(负载统计周期 + 微服务上线时间)采集到的数据,相比于负载统计周期来说周期增大,因此服务能力模型曲线上移。假设在负载统计周期内检测到的请求率在接下来这段时间(微服务上线时间)保存不变,那么依据该请求率的扩容动作,在出现请求平均响应时间 SLA 违约时,新的微服务已上线。样本统计周期与两者之和的差值越大,触发伸缩动作的服务请求率越小。

(2) 微服务-服务缩容

微服务-容器缩容的主要目标是提高资源利用率。缩容对实时性的要求远没有扩容要求高,这是因为缩容缓慢不但不会造成请求响应时间的 SLA 违约,反而缩短了服务请求响应时间。但是,为了在满足请求响应时间的同时,提高资源利用率,资源供给算法需对服务进行缩容。目前,Kubernetes 中常用的伸缩条件是通过判断所有 Pod 的平均 CPU 利用率进行缩容,如 CPU 利用率低于某个阈值(如小于 50%),并持续一段时间(如 1 min)。此外,Openstack 也采用了基于 CPU 利用率和持续时间阈值的弹性伸缩机制。

(3) 服务容量规划

基于服务能力模型的微服务-容器伸缩是建立在服务容量规划基础上。服务容量规划是指根据微服务集群的请求负载 *sumR* 和服务能力模型,对所需要的微服务实例数进行规划,规划方式有多种。

最简单的规划方式是从服务能力模型曲线中找到平均请求响应时间是 SLA 的那点所对应的服务请求率 *oneR*,将其作为一个微服务实例的请求吞吐量。期望的微服务实例数 *targetNum*:

$$targetNum = sumR/oneR \tag{5}$$

选择小于平均请求响应时间 SLA 的点所对应的服务请求率进行容量规划,则期望的微服务实例数增大。因此,会缩短平均请求响应时间,减少请求响应时间 SLA 违约的数量,但降低资源利用率。

服务扩容和缩容的触发条件,根据服务能力模型曲线中不同的点进行容量规划。为了减少服务-容器伸缩的频率,扩容选择缩容右边的点,如扩容阈值为 2 s,缩容阈值为 1 s。

基于服务能力模型的弹性资源供给算法如算法 2 所示。

算法 2 基于服务能力模型的弹性资源供给算法 SCMARP

输入:服务能力模型集合 $\boldsymbol{modelMap} = [<\sigma_0, F_0>, <\sigma_1, F_1>, \cdots, <\sigma_n, F_n>]$,
集群请求负载模式 $\boldsymbol{\alpha}_{\text{cluster}} = <x_1, x_2, \cdots, x_N>$,
模式差异度 χ,
正在运行的微服务实例数 N_{run},
负载统计周期 *loadCyc*,
触发扩容(expend)的平均请求响应时间阈值 TH_{expd},
触发缩容(reduce)的平均请求响应时间阈值 TH_{redu} (要求 $TH_{\text{expd}} >= TH_{\text{redu}}$)
输出:扩容实例数 N_{expd},
缩容实例数 N_{redu}

```
begin
  /* 计算当前集群请求负载模式的请求范式,并找到对应的服务能力模型函数 */
  δ_cluster = max(α_cluster)/χ
  σ = α_cluster/δ_cluster
  targetF = modelMap.find(σ)
  /* 计算服务能力模型曲线的扩容点和缩容点对应的范式率 */
  δ_expd = targetF^-1(TH_expd)
  δ_redu = target F^-1(TH_redu)
  /* 服务容量规划 */
  C_expd = δ_cluster/δ_expd
  C_redu = δ_cluster/δ_redu
  if (C_expd > N_run)
  N_expd = C_expd − N_run
  elseif (C_redu < N_run)
  N_expd = N_run − C_redu
  else
  N_expd = 0
  N_redu = 0
  endif
end
```

其中,服务能力模型集合 ***modelMap*** 中的每一项包含一种服务请求范式 σ 与其对应的服务能力模型曲线函数 F。$\boldsymbol{\alpha}_{\text{cluster}} = <x_1, x_2, \cdots, x_N>$ 表示微服务集群接收到对应 N 种服务请求类型的请求率。微服务-容器触发的时间阈值可以选择不同的值,但是要求扩容的时间阈值大于或等于缩容的时间阈值,否则会产生即扩容又缩容的伸缩决策。targetF^{-1} 为单值函数,是指 targetF^{-1} 的反函数。$\text{targetF}^{-1}(TH_{\text{expd}})$ 是求对应请求响应时间为 TH_{expd} 时的范式率。此外,在求请求响应时间对应的范式率时,可以通过不断增大范式率来逐渐逼近请求响应时间的方法进行求解。

3 实验结果与分析

为评价本文提出的基于服务能力模型的弹性资源供给算法 SCMARP 的弹性伸缩性能,将 SCMARP 算法与经典的基于阈值的资源供给算法 TARP 进行了对比,包括使用的服务节点数、平均请求响应时间和请求截止时间失败率。基于阈值的资源供给算法 TARP 的原理是当前一秒的所有服务请求的平均响应时间超过 2 s 时进行扩容,每次增加一个服务实例。当前一秒的所有服务请求的平均响应时间小于 1 s 时,进行缩容,每次减少一个服务实例。截止时间失败是指服务请求的响应时间超过一个阈值,即请求超时,实验中超时阈值设为 5 s。同时,设定平均服务响应时间的 SLA 为 2 s(±10%),容器的启动时间为 1 s。

实验包括 1 个客户端、4 个可用微服务实例和 1 个 MMAE,微服务使用 C++ 语言开发,非线性回归的服务能力模型算法 NRSCM 基于 scikit-learn library 库的 Python 语言开发。客户端、微服务实例和

MMAE 均运行在独立的节点上,其配置参数如表 1 所示。所有的计算节点通过 1000 Mbps 带宽的网络进行互联。

表 1　节点配置参数

	CPU	Mem	OS	Work threads
Server-1	1core 2.4GHz	1GB	Centos6.5	4
Server-2	1core 2.4GHz	1GB	Centos6.5	4
Server-3	1core 2.4GHz	1GB	Centos6.5	4
Server-4	1core 2.4GHz	1GB	Centos6.5	4
MMAE	2cores 2.4GHz	2GB	Centos6.5	–
Client	1core 2.4GHz	1GB	Centos6.5	–

选择范式 1 进行实验,根据图 2 设定触发扩容的平均请求响应时间阈值 $TH_{expd}=2$ s,触发缩容的平均请求响应时间阈值 $TH_{redu}=2$ s,此时范式率为 110 请求/s。对于一个范式率客户端连续访问10 s,以保证与服务能力模型的建模负载特征相符,服务请求负载如图 3 所示,

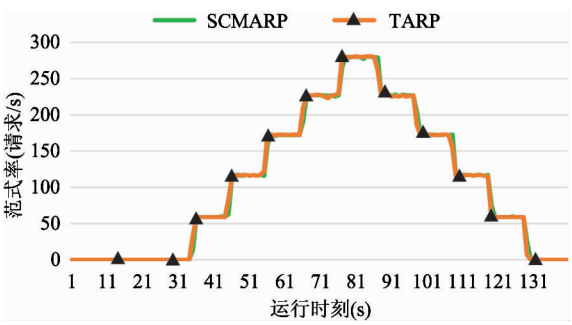


图 3　服务请求负载

在整个服务运行时间段,两种算法的服务实例变化情况如图 4 所示。

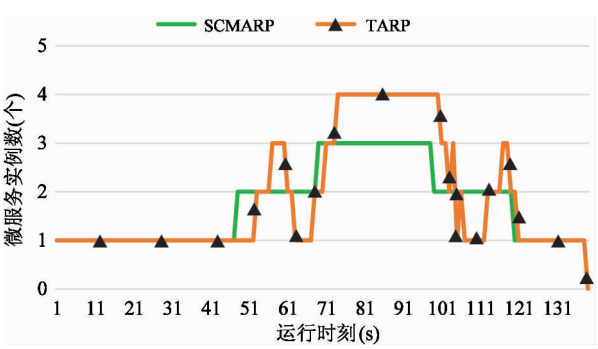


图 4　服务实例变化情况

从图 4 可以明显地看出, SCMARP 算法使用的服务实例数比图 3 中的负载吻合程度更高,最大使用服务实例数为 3,平均使用资源数为 1.74;在 TARP 算法中,服务实例数抖动比较严重,最大使用服务实例数为 4,平均使用资源数为 1.87。2 种算法相比,SCMARP 算法的平均资源使用数减少 6%。

每秒的平均请求响应时间、所有服务请求的请求响应时间分别如图 5、图 6 所示。

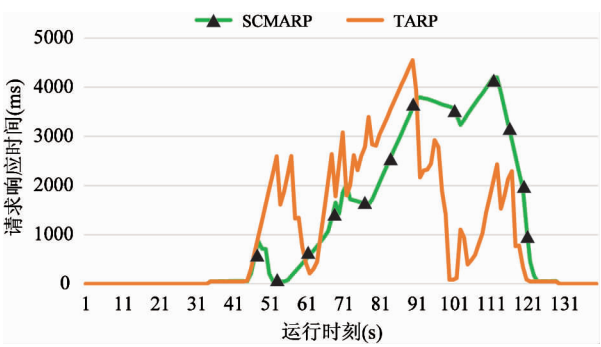


图 5　平均请求响应时间

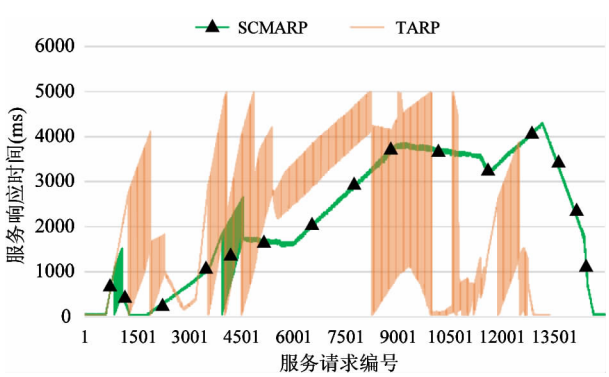


图 6　所有服务请求的响应时间

从图 5 和图 6 可以看出,在服务实例发生伸缩时,平均请求响应时间随着发生变化,SCMARP 算法中平均请求响应时间抖动较小。通过统计所有服务请求的响应时间可以得到 SCMARP 算法一共成功地完成 15 000 条服务请求,即失败率为 0,平均请求响应时间为 2 195 ms;TARP 算法一共完成 13 394 条服务请求,即请求截止失败率为 10.7%,已完成服务请求的平均请求响应时间为 1 937 ms,即两种算法的平均请求响应时间基本相同。

4 结 论

针对实时性要求高、服务请求响应时间敏感的应用场景,为解决在满足服务请求响应时间要求的基础上减少资源使用量的难题,本文提出了基于服务能力模型的弹性资源供给算法 SCMARP。为建立准确的服务能力模型,提出了基于非线性回归的服务能力模型算法 NRSCM。基于建立的服务能力模型,SCMARP 算法能够进行快速、准确的弹性伸缩。实验结果表明,与经典的基于阈值弹性资源供给算法 TARP 相比,在保障平均请求响应时间的基础上,平均资源使用数减少 6%,请求超时失败率降低 10.7%。

参考文献

[1] Thönes J. Microservices[J]. *IEEE Software*, 2015, 32(1):116-116

[2] Nadareishvili I, Mitra R, Mclarty M, et al. Microservice Architecture: Aligning Principles, Practices, and Culture [M]. Sebastopol: O'Reilly Media, Inc. 2016

[3] 郝庭毅, 吴恒, 吴国全,等. 面向微服务架构的容器级弹性资源供给方法[J]. 计算机研究与发展, 2017, 54(3):597-608

[4] Lorido-Botran T, Miguel-Alonso J, Lozano J A. A review of auto-scaling techniques for elastic applications in cloud environments[J]. *Journal of Grid Computing*, 2014, 12(4): 559-592

[5] 魏豪, 周抒睿, 张锐,等. 基于应用特征的 PaaS 弹性资源管理机制[J]. 计算机学报, 2016, 39(2):223-236

[6] Armbrust M, Fox A, Griffith R, et al. A view of cloud computing[J]. *Communications of the ACM*, 2010, 53(4): 50-58

[7] Salah K, Boutaba R. Estimating service response time for elastic cloud applications[C]. In: Proceedings of the 2012 IEEE 1st International Conference on Cloud Networking (CLOUDNET), Paris, France, 2012. 12-16

[8] Ali-Eldin A, Kihl M, Tordsson J, et al. Efficient provisioning of bursty scientific workloads on the cloud using adaptive elasticity control[C] In: Proceedings of the 3rd Workshop on Scientific Cloud Computing, Delft, Nertherlands, 2012. 31-40

[9] 殷波, 张云勇, 王志军,等. 基于弹性资源调整的云计算资源分配方法[J]. 通信学报, 2014

[10] 李勇, 张章, 孟丹,等. 面向混合负载的集群资源弹性调度[J]. 高技术通讯, 2014, 24(8):782-790

[11] Galante G, Bona L C E. A survey on cloud computing elasticity[C]. In: Proceedings of the 2012 IEEE/ACM 5th International Conference on Utility and Cloud Computing, Chicago, USA, 2012. 263-270

[12] Lee Y C, Lian B. Cloud bursting scheduler for cost efficiency[C]. In: Proceedings of the 2017 IEEE 10th International Conference on Cloud Computing (CLOUD), Honloulu, USA, 2017. 774-777

[13] Coutinho E F, de Carvalho Sousa F R, Rego P A L, et al. Elasticity in cloud computing: a survey[J]. *Annals of Telecommunications-annales Des Télécommunications*, 2015, 70(7-8): 289-309

[14] Dutreilh X, Moreau A, Malenfant J, et al. From data center resource allocation to control theory and back[C]. In: Proceedings of the 2010 IEEE 3rd International Conference on Cloud Computing (CLOUD), Miami, USA, 2010. 410-417

[15] Hasan M Z, Magana E, Clemm A, et al. Integrated and autonomic cloud resource scaling[C]. In: Proceedings of the Network Operations and Management Symposium (NOMS), Maui, USA, 2012. 1327-1334

[16] Koperek P, Funika W. Dynamic business metrics-driven resource provisioning in cloud environments[C]. In: Proceedings of the International Conference on Parallel Processing and Applied Mathematics, Torun, Poland, 2011. 171-180

[17] Ritter T, Mitschang B, Mega C. Dynamic Provisioning of System Topologies in the Cloud[M]. London: Enterprise

Interoperability V. Springer, 2012. 391-401

[18] Pop F, Potop-Butucaru M. ARMCO: Advanced topics in resource management for ubiquitous cloud computing: An adaptive approach[J]. *Future Generation Computer Systems*, 2016, 54:79-81

[19] Casalicchio E, Silvestri L. Mechanisms for SLA provisioning in cloud-based service providers[J]. *Computer Networks*, 2013, 57(3): 795-810

[20] Dutreilh X, Kirgizov S, Melekhova O, et al. Using reinforcement learning for autonomic resource allocation in clouds: towards a fully automated workflow[C]. In: Proceedings of the 7th International Conference on Autonomic and Autonomous Systems, Venice, Italy, 2011: 67-74

[21] Xu C Z, Rao J, Bu X. URL: a unified reinforcement learning approach for autonomic cloud management[J]. *Journal of Parallel and Distributed Computing*, 2012, 72(2): 95-105

Service-capacity-model based auto-scaling of resource provisioning mechanism for microservices

Liu Xiaodong*, Zhao Xiaofang*, Chen Yajing**, Chen Jihong**

(*Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190)

(**GeoPhysical Technique Research Center, BGP, CNPC, Zhuozhou 072751)

Abstract

In order to solve the problem of meeting the requirement of service request response time and reducing resource usage under high real-time requirements and sensitive to service request response time, a service capacity model based auto-scaling of resource provisioning (SCMARP) is proposed. In order to establish an accurate service capability model, nonlinear regression based service capacity model (NRSCM) is proposed. In the resource supply phase, SCMARP quickly and accurately scales resources according to the current service request load, service request paradigm, and established service capability model, thereby reducing the number of violations of request response time SLAs and resource usage. The experiment results indicate that compared with the threshold based auto-scaling of resource provisioning (TARP), on the basis of meeting the requirement of response time, the average resource usage decreased by 6%, and the service request timeout failure rate decreased by 10.7%.

Key words: microservices, auto-scaling of resource provisioning, service capacity model, nonlinear regression