

# 一种基于 Lasso 回归的微服务性能建模方法

郑杰生<sup>1</sup>, 谢彬瑜<sup>1</sup>, 吴广财<sup>1</sup>, 陈 非<sup>1</sup>, 花 磊<sup>2</sup>

(1. 广东电力信息科技有限公司, 广东 广州 510000;

2. 苏州博纳讯动软件有限公司, 江苏 苏州 215000)

**摘 要:**微服务技术广泛用于构建多样化的分布式软件,微服务的资源使用取决于其所实现的内部功能和处理的外部负载,负载突增会造成软件的性能衰减,因此需要动态调整微服务的最大访问速率以保证其服务质量。然而,在云计算环境下,软件的部署环境与应用类型具有多样性和复杂性,因而难以准确评估微服务处理请求的能力。为了应对以上问题,提出一种基于 Lasso 回归的微服务性能建模方法。首先将目标微服务放置在独立的 Docker 容器中,而后模拟生成微服务的外部负载并搜集其性能监测数据,进而基于 Lasso 回归建立资源与性能的关联模型以评估微服务的请求处理能力,从而实现微服务的细粒度灵活水平扩展。最后实现了原型系统并进行典型微服务实验,结果表明系统及方法具有较低的预测误差,并能够为软件提供较好的性能保障。

**关键词:**微服务;性能建模;容量规划;Lasso 回归;云计算

中图分类号:TP312

文献标识码:A

文章编号:1673-629X(2020)12-0216-05

doi:10.3969/j.issn.1673-629X.2020.12.038

## A Lasso Regression Based Performance Modeling Method for Microservices

ZHENG Jie-sheng<sup>1</sup>, XIE Bin-yu<sup>1</sup>, WU Guang-cai<sup>1</sup>, CHEN Fei<sup>1</sup>, HUA Lei<sup>2</sup>

(1. Guangdong Electric Power Information Technology Co., Ltd., Guangzhou 510000, China;

2. Suzhou Bona Xundong Software Co., Ltd., Suzhou 215000, China)

**Abstract:**Microservice technologies are widely used to develop various distributed software. The resource utilization of microservices depends on their implemented functions and processed external workloads. The surge of workloads can cause the performance degradation of software, so it is necessary to dynamically adjust the maximum access rate of each microservice to ensure its quality of service. However, since the deployment environment and application types of software are diverse and complex in cloud computing, it is difficult to accurately assess the service capability of each microservice to process requests. We propose a Lasso regression-based method of microservices' performance modeling. First, the target microservice is placed in an independent Docker container, and then the external workload of the microservice is simulated to generate and its performance monitoring data is collected. Then, an association model of resources and performance is established based on Lasso regression to evaluate the request processing capacity of the microservice, so as to realize the fine-grained and flexible horizontal expansion of the microservice. The prototype system is implemented to evaluate the proposed method on a typical microservice. The experiment shows that the proposed method has low prediction error rate and can improve the overall performance.

**Key words:**microservices; performance modeling; capacity plan; Lasso regression; cloud computing

## 0 引言

微服务技术将分布式应用软件划分为多个功能独立、松散耦合的细粒度微服务,通过与语言和平台无关的 API 与其他微服务进行通信,众多微服务协同实现应用功能<sup>[1]</sup>。这些微服务具有独立可自治的特点,从而实现了高度的伸缩性与灵活性,可以由多种不同的

语言和框架(如 Spring Cloud<sup>[2]</sup>, Dubbo<sup>[3]</sup>)来实现。每个微服务可以独立部署、资源调度和运行监测,应用可以启动或扩展特定的微服务,而不必启动整个应用的全部实例,以简化对云应用众多组件及服务的精细化管理<sup>[4]</sup>。微服务为运行时应用管理提供了高度灵活性,可以通过改变微服务副本的数量来扩展特定微服

收稿日期:2020-01-19

修回日期:2020-05-20

基金项目:国家重点研发计划(2018YFB1403004)

作者简介:郑杰生(1986-),男,硕士,高级工程师,通信作者,研究方向为企业信息化建设。

务的容量,从而从整体上有效提升应用性能。微服务可以直接部署在主机上或者包装为容器进行部署,通过在相同物理或虚拟主机上并行运行多个微服务实例以高效利用主机资源<sup>[5]</sup>。

近年来,微服务越来越多地与云计算技术相结合以构建弹性可伸缩的基于互联网的软件应用,大规模微服务通常部署在共享物理资源的云计算平台,广泛应用在众多领域<sup>[6]</sup>。云计算通过互联网方便访问共享计算、存储、网络等物理资源,云计算数据中心的物理服务器由云服务提供商管理及维护,以虚拟机或容器的形式将共享物理资源提供给用户使用,云计算用户仅支付实际使用费用,而无需维护物理设备。

当前云计算平台通常由虚拟机和容器等两个虚拟化层组成。容器是轻量级可独立部署执行的软件包,包含运行容器所需的依赖组件及运行环境,可以直接部署在主机或虚拟机上运行。基于容器的虚拟化技术可以在多个容器中实例化微服务,使用多个内核并行运行容器,以增加服务器的资源利用率,这样单个操作系统上可以运行多个隔离的微服务实例<sup>[7]</sup>。例如,大规模容器调度系统 Kubernetes 使用配置文件通过 Pod 对容器进行分组及资源约束,以共享物理或虚拟资源,调度和协调容器运行。但是由于物理设备配置、虚拟机类型、运行应用的差别,在云计算环境下,实际部署应用的性能与预期存在显著差异,因此难以为所有云应用创建通用的性能模型。

外部负载是影响交互式应用性能的关键因素,当并发数量超过某个数值后,服务质量会显著下降,通常表现为用户请求的响应时间超出服务提供商所定义的阈值。该文将微服务容量定义为在不违反服务质量约束的条件下,微服务可以处理的最大请求速率。使用微服务容量指标可有效检测应用性能瓶颈,实现自动化、智能化、精准化的应用容量规划。因此,使用能满足服务质量约束的负载并发量表示服务的容量,该值通过对应用或服务进行压力测试来确定,即不断增加负载直到违反服务质量约束,而后计算此时的并发数量。

应用容量规划是实现云计算的动态灵活伸缩,提升云应用的服务承载能力,实现高效资源利用的关键技术之一<sup>[8]</sup>。性能瓶颈检测用于分析造成应用性能衰减的关键微服务,通过水平扩展该微服务实例以提高应用的整体性能。应用容量规划是建立在性能瓶颈检测基础上,预测负载变化<sup>[9]</sup>,制定应用资源分配计划,以实现自动化的容量规划。性能建模是准确发现应用的性能瓶颈,有效进行容量规划的关键,而现有技术难以对不同类型的虚拟机资源和部署环境进行性能评估。

文献[10]提出了面向多层应用的瓶颈检测和性能预测的分析模型。文献[11]使用容量分析的方法识别潜在的资源瓶颈,并提出了性能优化的建议。Root 系统<sup>[12]</sup>自动检测部署在 PaaS 云计算环境中的 Web 应用性能异常。但现有方法多关注于层次较少、部署环境单一、规模受限的应用,且需要人工参与解释、分析结果,难以应对环境复杂的云计算环境下,规模巨大、类型多样的微服务应用。

为了应对云计算环境下微服务瓶颈检测和容量规划所面临的挑战,该文提出一种基于 Lasso 回归的微服务性能建模方法。首先将目标微服务放置在独立的 Docker 容器中,而后使用 Istio 为微服务模拟生成外部负载并搜集其性能监测数据,进而基于 Lasso 回归建立资源与性能的关联模型以评估微服务的请求处理能力,从而实现微服务的细粒度灵活水平扩展。

## 1 建模方法

### 1.1 基于 Istio 的微服务性能测试

Docker 容器技术将目标微服务与依赖的系统资源或服务进行隔离,以便对目标服务进行有针对性的测试与分析<sup>[13]</sup>。Istio 为微服务开发与管理提供服务网格基础平台,能够高效运行分布式微服务应用,并提供了统一的连接、监测和保护微服务的方式,以降低微服务部署的复杂性以减轻开发团队的工作量。该文将目标微服务实例放置在 Docker 容器环境中,通过 Istio 的服务代理机制使其与依赖的微服务隔离,而无需对原微服务做改动。Istio 以边车模式为每个微服务部署服务代理,微服务接收对原微服务的 API 调用请求并响应以评估该微服务的性能。服务代理的 URL 作为环境变量形式传递给微服务,以保证能够截获每个请求并返回响应。

该文通过 Kubernetes 将具有服务代理的微服务 Docker 容器部署在服务器集群中,线性增加外部负载,收集资源利用率和性能指标。当检测到性能度量违反服务质量约束时,将收集的数据持久化存储到数据库中,作为目标微服务的容量。在不同部署配置下,重复以上负载生成和数据搜集过程,不断修改容器配置对模型进行训练,直到生成的性能模型的精度不再明显提高为止。

### 1.2 基于 Lasso 回归的微服务性能建模

该文使用负载测试产生的 Docker 容器中微服务的监测数据,基于 Lasso 回归模型<sup>[14]</sup>刻画微服务容量与资源使用(虚拟 CPU、内存、网络等度量)的关联关系。与其他回归模型相比,如支持向量回归(SVR)、简单线性回归、多项式回归,Lasso 回归模型具有更高的准确性,并且能够在更短的时间内拟合收敛。该文

基于该模型预测在特定部署配置条件下目标微服务的容量,判断处理特定数量请求所需的微服务副本数量。模型的输入为多维资源度量,输出为微服务容量,基于 Lasso 回归的性能建模过程具体如下:

向量  $X_t = (x_{t1}^i, x_{t2}^i, \dots, x_{tq}^i)^T$  表示在时刻  $t$ , 微服务  $i$  的  $q$  个资源度量(如 CPU 利用率、网络带宽、磁盘读写速率、内存使用量等)的监测值,对其做标准化处理,作为 Lasso 回归模型的解释变量。用  $Y_t$  表示在时刻  $t$  的微服务  $i$  的容量,作为 Lasso 回归模型的响应变量,构建的回归模型为:

$$Y_t = \sum_{j=1}^q \omega_j^i x_{tj}^i + \alpha$$

其中,  $q$  表示该服务内监测度量的个数,  $x_{tj}^i$  表示时刻  $t$  微服务  $i$  资源度量  $j$  的监测值,  $\omega_j^i$  为  $x_{tj}^i$  对应的回归系数,  $\alpha$  为随机误差项。在约束条件  $\sum_{j=1}^q |\omega_j^i| \leq c$  下,通过坐标下降法极小化误差项  $\sum_{t=1}^T (Y_t - \sum_{j=1}^q \omega_j^i x_{tj}^i - \alpha)^2$ , 其中  $T$  是用于回归建模数据的时间长度。

约束条件的参数  $c$  通过广义交叉验证法来选择,其形式为:

$$GCV(c) = \frac{1}{N} \frac{RSS(c)}{[1 - p(c)/N]^2}$$

其中,  $RSS(c)$  表示残差平方和:  $RSS = \sum_{i=1}^N (Y_i - Y_i(c))^2$ ,  $p(c)$  为 Lasso 回归模型中有效回归系数的个数,  $N$  为所监测度量的个数。

## 2 建模工具

该文根据以上性能建模方法,设计并实现了性能建模工具,用来部署基于 Kubernetes 的微服务集群、生成工作负载、监测运行状态、建模微服务性能及实现微服务容量规划。

如图 1 所示,建模工具采用微服务架构,由配置及部署、容量规划、运行监管、负载生成、Docker 容器等模块构成,通过 Restful API 进行模块间通信,将数据存储在 MySQL 数据库以供查询、分析。

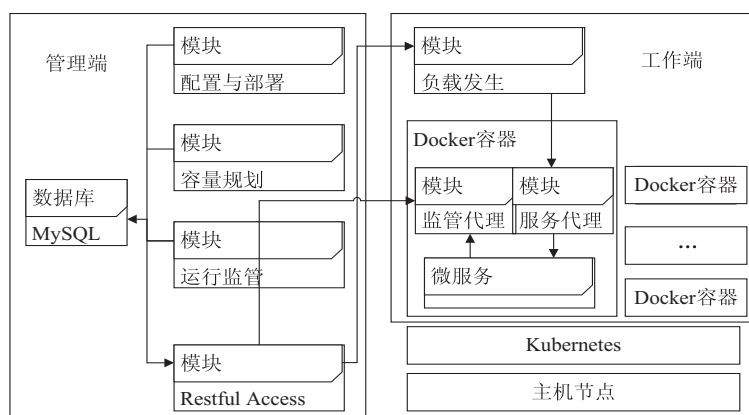


图 1 建模工具系统架构

配置及部署模块使用 Yaml 文件配置并构建微服务应用,将微服务包装在 Docker 容器中进行部署,并创建相应的 Docker 容器作为 Docker 容器;配置分配给每个微服务的副本数量,以及 CPU 和内存的物理资源限制,并部署 Kubernetes 集群。容量规划模块根据测试阶段收集的监测数据使用拟合的 Lasso 回归构建微服务的性能模型,基于 Lasso 回归模型刻画部署特征、资源消耗和性能之间的关联以估计在一定部署配置条件下的微服务容量;提供 Restful API 以设置配置参数,用户通过调用 API 可以启动测试,查看拟合的性能模型、估计的微服务容量、微服务副本数量。

运行监管模块使用监管代理监测每个微服务占用的物理资源,并将监测数据存储在 MySQL 数据库中;将 Yaml 文件、微服务名称和 API 作为配置信息以创建 Docker 容器。负载生成模块在 Kubernetes 集群上部署微服务后,使用 JMeter 生成线性增长的负载,用

以测试在特定部署配置条件下的微服务。Docker 容器模块利用 Docker 容器以隔离每个微服务,使用 Istio 为每个容器创建服务代理以接受请求并返回响应,并启动监管代理线程对 Docker 容器进行监测和管理。

用户通过 Restful API 设定目标应用及参数,如果应用包含多个微服务,则使用 Docker 容器包装每个微服务,并自动生成测试负载;而后,对于每个 Docker 容器中的微服务进行性能建模;最后,根据性能建模得到各微服务的容量,给出应用中各微服务运行实例数量的建议。

具体执行步骤包括:建模工具运行部署与配置模块将 Kubernetes 和 Istio 部署到云计算平台,设置 Pod 副本数量范围及资源约束条件;启动包装微服务的 Docker 容器镜像,并在容器启动时为微服务启动服务代理;负载生成模块产生线性增加的负载,监管代理监测容器的资源及性能度量,当检测到违反服务质量约



束时,记录微服务的负载、性能、容量等监测数据;建立以部署配置及环境为输入,以容量为输出的基于 Lasso 回归的性能模型;在特定配置下预测微服务容量,根据当前微服务负载状况,规划各微服务副本的数量。

### 3 实验评价

#### 3.1 实验环境

实验部署环境包括 4 台物理主机,其中 1 台为管理节点以部署建模工具的管理端程序,另外 3 台为工作节点以部署微服务 Docker 容器和建模工具模块。物理主机配置为 Intel Xeon E5-2620 CPU, 16 GB RAM 内存, 500 GB 磁盘, 100 Mbps 网络连接, 操作系统为 CentOS 6.5, 容器为 Docker 1.13, 容器编排为 Yaml 1.11.1, 容器管理系统为 Kubernetes 1.11.0。

在目标应用方面,该文使用典型微服务架构应用 MediaService<sup>[15]</sup>,选取其中 4 个典型微服务评价方法及工具的有效性。其中,VideoStreaming 为 I/O 密集型微服务,用以从后端 NFS 中读取流媒体格式的视频数据;Rating 为数据库访问型微服务,连接到后端数据库 MySQL,用以查询电影信息;ComposePage 为 Web 访问型微服务,用户接受用户请求并返回相应字符串;php-fpm 为服务网关微服务,用以接收用户请求,并将其分派给后端微服务,在收到所有微服务的响应后,组合响应信息并汇总返回结果。

在负载生成方面,实验根据每种微服务类型处理请求的资源利用情况,确定不同的负载生成率,负载数量引起资源利用率变化设定为每分钟约为 0.05%。负载测试从单个请求开始,请求数量呈线性增加,VideoStreaming 每分钟增加 12 次访问,Rating 每分钟增加 24 次访问,ComposePage 每分钟增加 36 次访问,Php-fpm 每分钟增加 48 次访问。

#### 3.2 实验结果

在容量预测准确性方面,该文使用平均绝对百分比误差(MAPE)将实际监测与模型预测的微服务容量进行比较,实验中 VideoStreaming, Rating, ComposePage 和 Php-fpm 的误差分别为 2.71%, 9.28%, 9.74% 和 6.48%,实验结果表明建模工具具有较高的预测准确性。

在应用性能保障方面,该文对于是否使用建模工具进行微服务水平扩展,应用的整体性能变化进行对比。性能指标使用 90% 请求响应时间,即在给定时间间隔内,处理 90% 请求的响应时间。

如图 2 所示,实验首先未使用建模工具进行容量规划,以整个应用为管理单元,设置副本数量为 3,监测应用性能变化。而后,使用建模工具进行容量规划,以微服务为管理单元,微服务副本数量根据微服务容量与负载数量动态变化,监测应用性能变化。

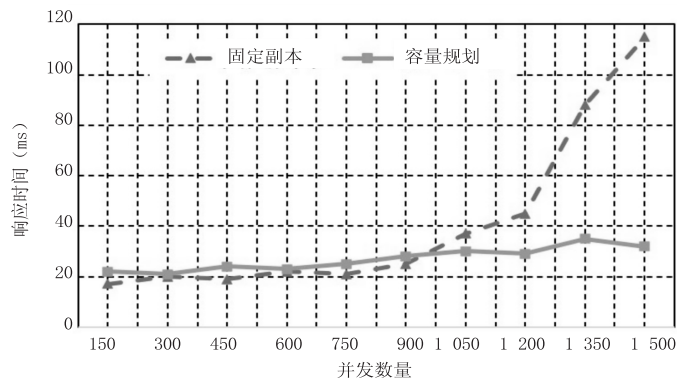


图 2 性能比较

实验结果表明,负载数量在 900 以下时,由于应用未达到性能瓶颈,二者的响应时间差别不大。当负载数量大于 1 050 时,对于未使用建模工具进行容量规划的应用,由于在 Rating 微服务出现了性能瓶颈,应用的响应时间呈指数级增加。对于使用建模工具进行容量规划的应用,由于能够根据负载数量动态扩展 Rating 微服务实例数量,响应时间保持平稳,并未出现大幅上升。

### 4 结束语

微服务技术将应用划分为多个功能独立的微服

务,并使用与语言 and 平台无关的 API 实现微服务间的通信,近年来在业界得到广泛应用。微服务的资源使用取决于其所实现的功能和外部负载,负载突增会造成应用违反服务质量约束,因此需要限制每个微服务的最大访问速率以保证其服务质量。然而,在云计算环境下,部署环境与应用种类的多样性与复杂性造成了难以准确评估每个微服务的服务能力。为了应对该挑战,提出一种基于 Lasso 回归的微服务应用性能建模方法。

该方法首先将目标微服务放置在 Docker 容器环境,而后生成外部负载并搜集微服务的性能及资源的

监测数据,基于 Lasso 回归模型建立配置、资源与性能之间的关联关系,进而评估每个微服务的容量以实现细粒度灵活扩展。基于以上方法,实现了原型系统,基于云计算平台使用典型微服务进行验证,实验结果表明该方法具有较低的预测误差,能够有效保证系统性能。

#### 参考文献:

- [1] JAMSHIDI P, PAHL C, MENDONA N C, et al. Microservices; the journey so far and challenges ahead [J]. IEEE Software, 2018, 35(3): 24–35.
- [2] 范迪, 朱志祥. 一种 Dubbo 框架的授权认证方案 [J]. 计算机技术与发展, 2017, 27(11): 115–118.
- [3] 罗钦凯, 倪成章. 基于微服务的工作流技术在云管平台的应用 [J]. 计算机技术与发展, 2019, 29(9): 122–127.
- [4] NEWMAN S. Building Microservices [M]. Sebastopol: O'Reilly, 2015: 81–90.
- [5] 曹伟杰, 贺建民, 孙志丹. IaaS 模式下虚拟机部署机制研究 [J]. 计算机技术与发展, 2012, 22(10): 105–108.
- [6] 吴磊, 湛健, 宋丽华. 微服务架构在智能家居网关系统中的应用研究 [J]. 计算机技术与发展, 2019, 29(11): 200–205.
- [7] JARAMILLO D, NGUYEN D V, SMART R. Leveraging microservices architecture by using Docker technology [C]//Proceedings of SoutheastCon. Norfolk, VA, USA: IEEE, 2016: 1–5.
- [8] 李超, 花磊, 宋云奎. 面向云计算的分布式应用自动部署框架 [J]. 计算机技术与发展, 2018, 28(6): 12–16.
- [9] BAUER A, HERBST N, KOUNEV S. Design and evaluation of a proactive, application-aware auto-scaler; tutorial paper [C]//Proceedings of the 8th international conference on performance engineering. New York, NY, USA: ACM, 2017: 425–428.
- [10] URGAKONKAR B, SHENOY P, CHANDRA A, et al. Dynamic provisioning of multi-tier internet applications [C]//Proceedings of the 2nd international conference on autonomic computing. Seattle, WA, USA: IEEE, 2005: 217–228.
- [11] LEYMAN F, BREITENBÜCHER U, WAGNER S, et al. Native cloud applications; why monolithic virtualization is not their foundation [C]//Proceedings of cloud computing and services science. Rome, Italy: Springer, 2017: 16–40.
- [12] JAYATHILAKA H, KRINTZ C, WOLSKI R. Performance monitoring and root cause analysis for cloud-hosted web applications [C]//Proceedings of the 26th international conference on world wide web. Republic and Canton of Geneva, Switzerland: ACM, 2017: 469–478.
- [13] PREVELAKIS V, SPINELLIS D. Sandboxing applications [C]//Proceedings of annual technical conference. Berkeley, CA, USA: USENIX, 2001: 119–126.
- [14] HASTIE T, TIBSHIRANI R, FRIEDMAN J. The elements of statistical learning [M]. Cham, German: Springer, 2017: 143–180.
- [15] GAN Y, ZHANG Y, CHENG D, et al. An open-source benchmark suite for microservices and their hardware-software implications for cloud & edge systems [C]//Proceedings of architectural support for programming languages and operating systems. New York, NY, USA: ACM, 2019: 3–18.