面向微服务应用的瓶颈感知自动伸缩框架

许树旺1)

摘 要 在具有动态工作负载的云应用中，自动伸缩对于确保最佳性能和资源利用率至关重要。然而，由于微服务负载模式的多样性和微服务之间的复杂交互，传统的自动伸缩技术在基于微服务的应用中通常不再适用。具体来说，性能异常通过交互的传播导致大量的异常微服务，使得识别根性能瓶颈（PB)和制定合适的伸缩策略变得困难。此外，为了平衡资源消耗和性能，现有主流的基于在线优化算法的方法需要多次迭代，导致振荡，提高了性能退化的可能性。为了解决这些问题，本文介绍了一个这几篇文献里最优的瓶颈感知自动伸缩框架。因此提出了一种新的基于拓扑势的随机游走算法TopoRank来减少不必要的缩放。

关键词 微服务、自动伸缩、性能瓶颈、副本管理

A Bottleneck-aware AutoscalingFramework for Microservice-based Applications

Shuwang Xu1)

Abstract Autoscaling is critical for ensuring optimal performance and resource utilization in cloud applications with dynamic workloads.However,traditional auto scaling technologies are typically no longer applicable in microservice-based applications due tothe diverse workload patterns and complex interactions between microservices. Specifically,the propagation of performance anomalies through interactions leads to a high number of abnormal microservices,making it difficult to identify the root performance bottlenecks(PBs)and formulate appropriate scaling strategies. In addition,to balance resource consumption and performance,the existing mainstream approaches based on online optimization algorithms require multiple iterations,leading to oscillation and elevating the likelihood of performance degradation.To tackle these issues, we propose a bottleneck-aware autoscaling framework designed to prevent performance degradation in a microservice-based application. Thus,we propose TopoRank,a novel random walk algorithm based on the topological potential to reduce unnecessary scaling.

Keywords microservice; autoscaling; performance bottleneck; replica management

1 介绍

随着微服务架构的发展，越来越多的云应用从单体架构向微服务架构迁移。这种新的架构通过将单个应用程序分解成多个通过HTTP或RPC协议相互通信的微服务来降低应用程序的耦合性。此外，每个微服务都可以由不同的团队独立开发、部署和扩展，从而实现快速的应用开发和迭代。然而，外部工作负载的不可预测性和微服务之间交互的复杂性会导致性能下降。云提供商必须准备过多的资源来满足应用所有者的服务水平目标（Service Level Objective，SLO)，这通常会造成不必要的资源浪费。因此，满足SLO和最小化资源消耗之间的不平衡成为微服务中资源管理面临的主要挑战。

微服务自动伸缩是指应对工作负载变化而弹性分配资源的能力[13]。通过利用微服务的弹性特性，自动伸缩可以缓解资源成本和性能之间的矛盾。然而，微服务的自动伸缩在短期内会受到性能瓶颈(Performance Bottleneck，PB)的精确伸缩。由于微服务间通信的复杂性，一个PB的退化可能会通过消息传递传播到其他微服务[2],导致出现大量的异常微服务。我

收稿日期：年-月-日；最终修改稿收到日期：年-月-日 *投稿时不填写此项*. 本课题得到… …基金中文完整名称(No.项目号)、… …基金中文完整名称(No.项目号)、… … 基金中文完整名称(No.项目号)资助.作者名1(**通信作者**)，性别，xxxx年生，学位(或目前学历)，职称，是/否计算机学会(CCF)会员（提供会员号） ,主要研究领域为*****、****.E-mail: **********.作者名2 （**通信作者**） ，性别，xxxx年生，学位(或目前学历)，职称，是/否计算机学会(CCF)会员（提供会员号） ,主要研究领域为*****、****.E-mail: **********. 作者名3（**通信作者**），性别，xxxx年生，学位(或目前学历)，职称，是/否计算机学会(CCF)会员（提供会员号） ,主要研究领域为*****、****.E-mail: **********.(**给出的电子邮件地址应不会因出国、毕业、更换工作单位等原因而变动。请给出所有作者的电子邮件**)

第1作者手机号码(投稿时必须提供，以便紧急联系，发表时会删除): … …, E-mail: … …*此部分6号宋体*

们通过在Google开发的开源微服务应用Online Boutique 1中为特定微服务注入突发工作负载来证明这一点。PB推荐中的性能下降可以扩散到上游的Checkout、主轴前端等微服务。为了进一步验证精确缩放PB的重要性，我们分别对不同的微服务进行了压力测试和缩放。异常微服务(主轴前端)伸缩并不能缓解SLO违规行为。然而，当我们识别并扩展PB推荐时，微服务应用的性能得到了提升。遗憾的是，定位PB通常比较耗时，偶尔也会出错[3]。

近年来，在自动伸缩之前，已经提出了几种识别关键微服务的方法。例如，Kubernetes 2的默认自动换算装置根据计算资源的静态阈值过滤用于直接扩展的微服务。通过计算服务功率，即第50百分位响应时间(P50)与第90百分位响应时间(P90)的比值，定义了弹性伸缩的边界。此外，还可以通过分析各种尾延迟的比例来提取关键路径。这些研究虽然缩小了自动伸缩的范围，但仍然考虑到了可能影响伸缩策略的非瓶颈微服务，尤其是当应用中大量微服务同时出现异常时。

2 现状分析

近年来，微服务场景下的瓶颈分析方法层出不穷，这些方法大多依赖于日志、痕迹和度量三种类型的数据。

Jia等[4]和Nandi等[5]首先从正常状态日志中提取模板和流，将其与目标日志进行匹配，过滤掉异常日志。Trace是一种基于事件跟踪的记录，它再现了微服务之间的请求过程。Yu等[6]通过结合频谱分析和PageRank算法在由轨迹构建的依赖图上定位瓶颈，而Mi等[7]提出了一个无监督的机器学习原型来学习微服务的模式并过滤异常微服务。然而使用痕迹会对代码产生干扰，并且需要操作员对微服务的结构有深入的了解。一些方法[8]利用图随机游走算法米模拟元吊的传播过程，然后通过整合度量的统计特征和微服务之间的依赖关系来发现瓶颈。此外，MicroCause[9]等方法侧重于构建带有因果推断的度量因果关系图，这些方法通常涉及度量之间隐藏的间接关系。

由于在监控度量值时很少修改工作流代码，因此仅集微服务的度量值通常比使用跟踪更经济。此外，由于度量指标在后一种场景中被广泛使用，使用度量指标作为主监测数据可以降低瓶颈分析和自动缩放集成的成本。尽管这些方法各有优势，但大多数方法在异常回溯起始点的选择上并无偏好。相比之下，本文的方法从具有更大异常潜力的微服务开始随机游走，加快了收敛速度，提高了瓶颈定位精度。

3 系统设计

我们提出了一个以PB为中心的自动缩放控制器，用于定位PB并为其优化副本。由三个部分组成:

1)度量采集器:为了提供对应用程序状态的实时洞察，我们设计了一个度量采集器，在固定的时间间隔内捕获并整合来自普罗米修斯4的监控度量。

2)性能瓶颈分析:在度量采集器的辅助下，该组件执行SLO违例检测和冗余检查，以识别具有异常行为的微服务。接下来，瓶颈定位过程将被触发以精确定位异常微服务中的PB。

3)规模决策(Scaling Decision):该组件旨在使用进化算法确定PB的最优副本数量。最后，生成具有优化策略的配置文件，并将其提交给kubernetes-client5，该文件规范微服务的副本数量。

3.1公制收集器

自动伸缩系统依赖于内存使用数据、系统负载和尾延迟等指标的实时访问，以确定在微服务应用中是否应该执行弹性伸缩以及应该分配多少资源。与需要深入理解程序和代码注入的基于跟踪的监控器不同,Metric Collectionor报告基于服务网格的度量，以最小化对业务流的干扰。如表1所示，本框架使用普罗米修斯和kube-statemetrics6对这些指标进行收集和分类，包括响应延迟、微服务之间的调用关系、资源消耗和微服务工作负载。

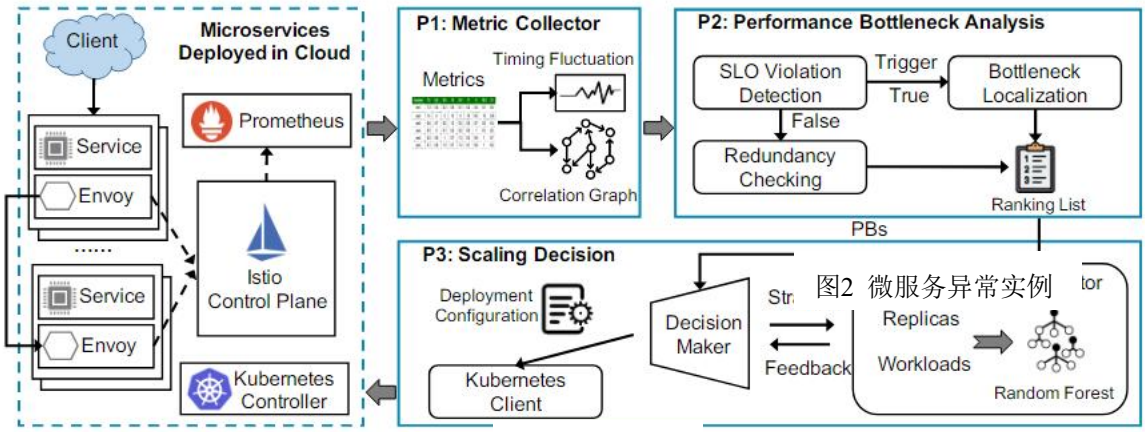
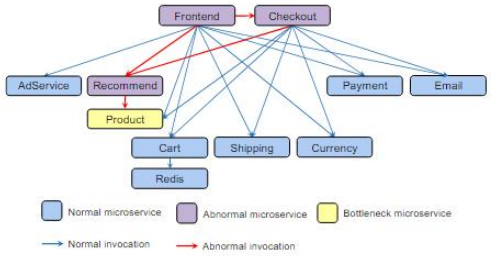


图1 伸缩框架

图2 微服务异常实例



3.2性能瓶颈分析

性能瓶颈分析(Performance Bottleneck Analysis，PBA)是一种旨在发现微服务应用中性能下降和资源浪费的过程，推断当前问题的PB。

1）Slo违规检测

为了检测微服务中的异常，使用服务水平目标(sLOs)与特定的指标进行比较。如果一个微服务有大量的SLO违规行为，即性能下降，则被认为是不正常的。度量收集器收集的调用关系可以用来构建微服务关联图Gc。每隔15秒检测Gc中所有调用边的P90tail延迟，及时发现性能下降。如果调用的尾部延迟超过预定阈值，则调用的被调用微服务将被添加到异常微服务集合(S)中，瓶颈定位过程将被激活。考虑到微服务延迟中偶尔存在的噪声，将阈值设置为SLO×(1 +α 2)，其中α用来调整对噪声的容忍度。

2）冗余性检查

在没有性能异常的情况下，一些微服务可能会被分配到比需求更多的资源。然而，仅通过度量指标很难识别这类情况，可能导致浪费有限的硬件资源。为了避免这种情况，需要识别哪些微服务分配了多余的资源。本框架使用微服务每秒的工作负载变化率来判断资源是否冗余。这种策略比仅仅依靠资源消耗更有效，因为不同的微服务对异构资源的敏感度可能不同。冗余检测的主要思想是通过假设检验来检测微服务当前的工作负载是否显著低于其过去的工作负载。

3）瓶颈定位

由于微服务应用中存在复杂的交互[10]，并不是每一个异常的微服务都需要进行扩展。例如，图5展示了瓶颈微服务的性能退化如何沿着调用链传播到其上游微服务(如推荐、主轴前端、Checkout等)，即使上游微服务没有过载。因此，只有瓶颈微服务必须进行扩容，其他异常微服务才会被牵连进来。

为了确定瓶颈微服务，我们引入了异常势的概念，它聚合了给定位置上所有微服务的异常影响。PB的异常潜力通常较高，因为它被许多受其影响的异常微服务所包围。我们设计了一种新颖的瓶颈定位算法TopoRank，该算法在随机游走中引入拓扑势理论(TPT)来计算所有异常微服务的得分，最后输出一个排名列表。表中得分最高的微服务可以被识别为PB。

3.3尺度决策

考虑到性能瓶颈分析识别出的PB，将对PB的副本进行缩放以最小化应用程序的资源消耗，同时保证微服务的端到端延迟

满足SLO。尽管丰富的副本可以缓解性能下降问题，但同时也消耗了大量的资源。因此，在性能保证和资源消耗之间保持平衡是至关重要的。缩放决策的过程将被建模为一个约束优化问题来实现这种平衡。

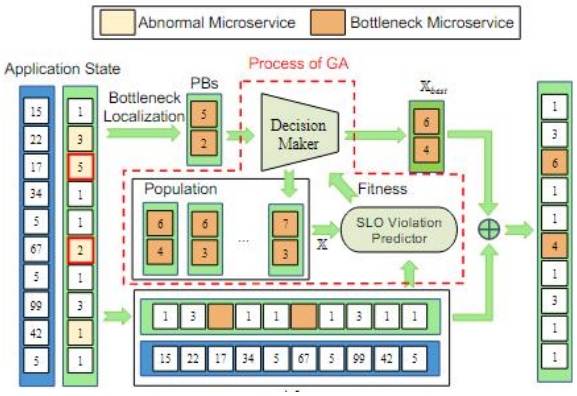


图3 自动缩放优化过程示意图

4 评价

在本节中，我们将详细介绍自动缩放的实验场景，包括本框架与来自学术界和工业界的几种最先进的自动缩放算法的比较。

总的来说，在两个微服务系统中，在6个工作负载下，本框架在减少SLO违例和最小化资源开销方面优于竞争方法。特别地，本框架在火车票上的SLO违反率平均比基准方法降低了4.96%，而资源开销平均降低了0.24$。这些结果表明，本框架能够快速、准确地对大规模微服务系统中的瓶颈微服务进行弹性伸缩，从而减少SLO违例，节约资源。对于online Boutique中的6个工作负载，本框架在其中的4个工作负载中也实现了最低的SLO违例率，并且在3个模拟工作负载中实现了资源消耗的最小化。

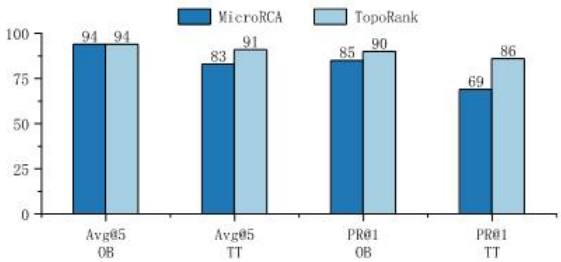


图4 性能比较

	SVM	Decision Tree	Random Forest	MLP
Precision(A)	0.819	0.891	**0.919**	0.799
Recall(A)	0.961	0.918	**0963**	0.930
Precision(B)	0.865	0.927	**0.956**	0.831
Recall(B)	0.915	0.941	**0.969**	0.907

表1 预测模型

6种工作负载下不同方法的延迟分布箱线图，探究了每种方法对微服务系统性能的影响。可以看出，绝大多数的自动缩放方法都能将潜伏期分布的中位数保持在红色虚线(SLO)以下。然而，对于所有的工作负荷，只有本框架更进一步，显著地降低了SLO之下的第三个四分位数。

为了评估使用本框架进行弹性缩放的时间成本，收集并统计了本框架中每个模块所需的平均时间。如表3所示，Online Boutique中所有本框架模块的总时间开销小于一个监控间隔(即5s)，而火车票的总时间开销小于两个监控间隔。得益于PBA缩小了决策范围，尽管微服务数量增加，但当应用从Online Boutique切换到Train Ticket时，决策者的时间成本(不超过6.6%)增加不多。

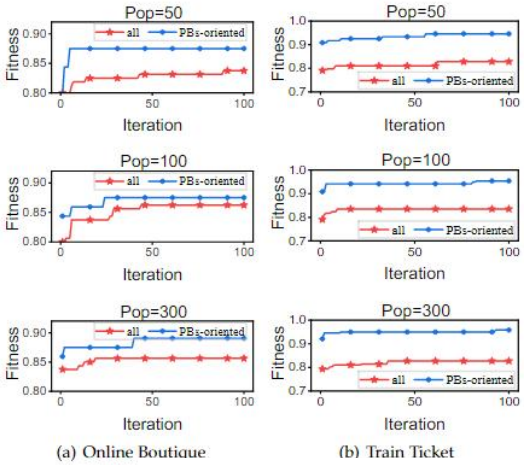


图5 遗传算法迭代过程

参 考 文 献

[1] Y. Al-Dhuraibi, F. Paraiso, N. Djarallah, and P. Merle, “ Elasticity incloud computing: state of the art and research challenges, ” IEEETransactions on Services Computing, vol. 11, no. 2, pp. 430–447, 2017.

[2] M. Kim, R. Sumbaly, and S. Shah, “Root cause detection ina service-oriented

architecture, ” ACM SIGMETRICS PerformanceEvaluation Review, vol. 41, no. 1, pp. 93–104, 2013.

[3] P. Chen, Y. Qi, and D. Hou, “Causeinfer: automated end-to-endperformance diagnosis with hierarchical causality graph in cloudenvironment, ” IEEE transactions on services computing, vol. 12, no. 2,pp. 214–230, 2016.

[4] T. Jia, P. Chen, L. Yang, Y. Li, F. Meng, and J. Xu, “An approachfor anomaly diagnosis based on hybrid graph model with logs fordistributed services, ” in 2017 IEEE international conference on webservices (ICWS). IEEE, 2017, pp. 25–32.

[5] A. Nandi, A. Mandal, S. Atreja, G. B. Dasgupta, and S. Bhat-tacharya, “Anomaly detection using program control flow graphmining from execution logs,” in Proceedings of the 22nd ACMSIGKDD International Conference on Knowledge Discovery and DataMining, 2016, pp. 215–224.

[6] G. Yu, P. Chen, H. Chen, Z. Guan, Z. Huang, L. Jing, T. Weng,X. Sun, and X. Li, “Microrank: End-to-end latency issue local-ization with extended spectrum analysis in microservice environ-ments,” in Proceedings of the Web Conference 2021, 2021, pp. 3087–3098.

[7] H. Mi, H. Wang, Y. Zhou, M. R.-T. Lyu, and H. Cai, “Toward fine-grained, unsupervised, scalable performance diagnosis for pro-duction cloud computing systems,” IEEE Transactions on Paralleland Distributed Systems, vol. 24, no. 6, pp. 1245–1255, 2013.

[8] L. Wu, J. Tordsson, E. Elmroth, and O. Kao, “Microrca: Rootcause localization of performance issues in microservices, ” inNOMS 2020-2020 IEEE/IFIP Network Operations and ManagementSymposium. IEEE, 2020, pp. 1–9.

[9] Y. Meng, S. Zhang, Y. Sun, R. Zhang, Z. Hu, Y. Zhang, C. Jia,Z. Wang, and D. Pei, “Localizin g failure root causes in a mi-croservice through causality inference, ” in 2020 IEEE/ACM 28thInternational Symposium on Quality of Service (IWQoS). IEEE, 2020,pp. 1–10.

[10] X. Zhou, X. Peng, T. Xie, J. Sun, C. Ji, D. Liu, Q. Xiang, and C. He, “Latent error prediction and fault localization for microserviceapplications by learning from system trace logs, ” in Proceedings ofthe 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering,2019, pp. 683–694.