计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.200800023

无服务器平台资源调度综述

马泽华[1]　刘　波[1]　林伟伟[2]　李加伟[1]
1 华南师范大学计算机学院　广州 510631
2 华南理工大学计算机科学与工程学院　广州 510641
(mzh.scnu@qq.com)

摘　要　无服务器(Serverless)计算正成为部署云应用程序的一种具有广阔发展前景的范式。其实现了一种真正的现收现付的计费方式,并且不会浪费资源。开发人员无需担心计算平台的底层细节,只需在处理请求或事件时付费,从而降低了开发人员的门槛。无服务器模式的转变虽然带来了机遇,但也带来了平台函数冷启动延迟、资源利用不足等问题。为此,文中对无服务器计算平台的资源调度技术做了深入的调查和分析,重点阐述了面向资源利用、响应时间延迟以及多目标优化的无服务器平台资源调度的技术原理和相关研究现状,并在此基础上分析总结并指明了无服务器平台资源调度未来的主要研究方向:即面向不同应用类型负载的调度优化、响应时间与资源利用率的折衷调度、虚拟机与无服务器平台的联合调度以及无服务器资源调度的混合算法。

关键词:无服务器计算;函数即服务;资源调度
中图法分类号　TP393

Survey of Resource Scheduling for Serverless Platforms

MA Ze-hua[1],LIU Bo[1],LIN Wei-wei[2] and LI Jia-wei[1]
1 School of Computer Science,South China Normal University,Guangzhou 510631,China
2 School of Computer Science and Engineering,South China University of Technology,Guangzhou 510641,China

Abstract　Serverless computing is emerging as a promising paradigm for deploying cloud applications. It really implements pay-as-you-go billing without wasting resources. Developers don't have to worry about the low-level details of the computing platform,they just need to pay when processing requests or events,thus lowering the threshold for developers. Although the change of serverless model brings opportunities,it also brings problems such as cold startup delay of platform function and insufficient resource utilization. Therefore,this paper makes an in-depth investigation and analysis of the resource scheduling technology of the serverless computing platform. The technical principle and research status of resource scheduling in serverless platform based on resource utilization,response time delay and multi-objective optimization are discussed. Finally,this paper points out the trends of future research directions:scheduling for different application types,the tradeoff between response time and resource utilization,joint scheduling between virtual machine and serverless platform,and hybrid algorithm for serverless resource scheduling.

Keywords　Serverless computing,Functions as a services,Resource scheduling

1　引言

近年来,云计算,特别是基础设施即服务(Infrastructure as a Service,IaaS)提供的虚拟机技术已经被各行各业广泛接受并采用,企业越来越多地采用云的主要因素是它的现收现付模式,即客户只需为向云提供商租用的资源付费,并且能够获得所需的任意规模的资源。然而,数据中心[1]报告中云资源使用情况的研究表明:云客户支付的资源(租用虚拟机)与实际资源使用(CPU、内存等)之间存在巨大差距,这造成了资源的浪费,增加了云客户的成本。

无服务器(Serverless)计算的诞生为解决这一问题提供了思路,其正成为部署云应用程序的一种新颖范例。云原生

到稿日期:2020-08-03　返修日期:2020-09-10　本文已加入开放科学计划(OSID),请扫描上方二维码获取补充信息。
基金项目:国家自然科学基金项目(61872084,61772205);广东省基础与应用基础研究重大项目(2019B030302002);广州市科技计划项目(202007040002,201902010040)
This work was supported by the National Natural Science Foundation of China (61872084,61772205),Guangdong Major Project of Basic and Applied Basic Research (2019B030302002) and Guangzhou Science and Technology Plan Project (202007040002,201902010040).
通信作者:林伟伟(linww@scut.edu.cn)

计算基金会(CNCF)[2]给出了无服务器计算的权威定义:无服务器计算描述了一种更细粒度的部署模型,其中包含一个或多个函数的应用程序被捆绑到平台上,然后应用程序根据当前的确切需求执行、伸缩和计费[3]。无服务器计算实现了一种真正的现收现付(以毫秒为粒度)的计费方式,有效地节约了资源,并将所有的操作复杂性委托给云提供商,从而降低了开发人员的门槛。由于其研发交付速度和成本的优势,无服务器计算越来越受欢迎,预计到 2021 年,其市场规模将增长到 77.2 亿美元[4],其中包括 Amazon、IBM、Microsoft、谷歌、腾讯和阿里巴巴等已经发布了无服务器计算能力的云供应商,以及由行业和学术机构共同推动的一些开源库。

无服务器计算使得开发人员无需担心可用性、可伸缩性、容错性、虚拟机资源的供应、服务器管理和其他基础设施问题,而是专注于应用程序的业务方面。但是,在无服务器计算中,提供商必须决定如何调度用户传入的函数,其中一个难题就是用户应用程序集的多样性。无服务器平台的优势在于其不需要明确的配置、自动且几乎无限的扩展,并按使用计费,因此吸引了不同应用社区的兴趣,其中就包括高性能计算[5]。这种多样性使得特定于应用程序的函数调度成为必要,因为不同的应用程序具有截然不同的资源消耗模式,如短期应用程序、具有间歇性活动的应用程序和某些并行服务[6]。

调度的另一个难点是如何利用容器来托管应用程序。虽然容器有助于打包应用程序并简化函数调用过程,但它们确实会带来不良的性能影响。在大多数函数即服务(Functions as a Service,FaaS)平台上,函数首先会注入容器中,然后该容器将运行在虚拟机或物理机器上。如果没有请求函数的实例,FaaS 平台将派生出新的容器,并注入函数,配置依赖项,这称为冷启动延迟[7]。函数在容器中的冷启动可能会影响用户的服务水平协议(Service Level Agreement,SLA)。理想情况下,云应用程序的所有者希望维护用户的 SLA[8-9],从而不会导致更高的成本或增加系统复杂性。为此,本文对无服务器计算平台的资源调度技术做了深入的调查和分析,重点阐述了面向资源利用、响应时间延迟以及多目标优化的无服务器平台资源调度,并指明了未来研究方向的趋势。

2 相关背景

2.1 云计算与服务器计算

在 IaaS 模型中,开发人员对云中的应用程序代码和基础设施拥有最大的控制权。而在另外的 PaaS 和 SaaS 模型中,开发人员不需要了解任何基础设施,因此不再控制基础设施。相反,开发人员可以访问预先打包的组件或完整的应用程序[10]。

本节将关注图 1 所示的无服务器计算平台模型。在这里,开发人员可以控制部署到云中的代码,这些代码必须以无状态函数的形式编写。开发人员无需担心代码部署和维护,并且期望它具有容错和自动伸缩的能力。无服务器计算模型与平台即服务(PaaS)模型都抽象了对服务器的管理,且无服务器计算模型提供了基于无状态函数的“简化”编程模型[11-13]。但与 PaaS 不同的是,无服务器计算模型的开发人员可以编写任意代码,且不局限于使用预先打包的应用程序。特别地,当不使用用户的函数代码时,对于用户来说没有成本,代码可能被缩放为零。

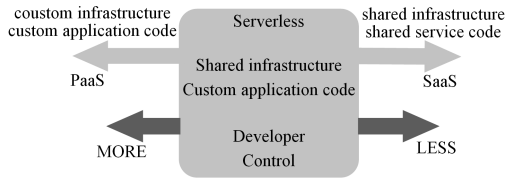

图 1 无服务器模型

Fig. 1 Serverless computing model

2.2 函数及服务

服务器计算最简单的方式是提供一段代码(函数)由无服务器计算平台执行,它促使 FaaS 平台的兴起。代码以函数的形式在 FaaS 平台中运行有限的时间(最多几分钟),这些函数由事件或 HTTP 请求(或其他触发器)触发执行,并且不允许保持持久状态(函数可以在任何时候重新启动)。通过限制执行时间和不允许函数保持持久状态,服务提供者可以很容易地维护和扩展平台[14]。通常,无服务器计算平台使用容器或其他沙箱方法来实例化函数[15]。

图 2 演示了一个典型无服务器计算平台[16]。传入的用户函数通过 HTTP 前端发送到调度器;然后,调度器选择一个可以运行函数的调用程序;根据应用程序的不同,函数可以使用附加的后端服务,如数据库、对象存储等。

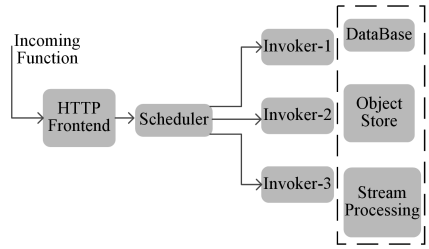

图 2 无服务器计算平台

Fig. 2 Serverless computing platform

无服务器函数适合在流量差异很大的网站上采用,可以在网站没有负载时降低服务成本,同时在需要时提供及时的资源扩展[17]。另一个适合无服务器架构的大型系统的例子是物联网后端。此外,FaaS 适用于需要大量实时数据处理和并行处理的应用程序。这类应用程序的例子包括媒体公司 Thomson Reuters,其使用了无服务器计算方法来更有效地分析产品数据[18]。娱乐公司 Netflix 也在使用 AWS Lambda,如媒体数据的并行处理、备份更新和安全流程验证[19]。此外,Serverless 计算和 CPS 的集成可以帮助系统的网络组件与物理组件进行连接,并促进它们之间的交互。例如,农业机械上的传感器可以触发无服务器平台后端上的函数并自动购买所需的新部件[20]。

3 无服务器平台的资源调度

无服务器计算是一种具有广阔前景的范例,代表云应用

程序开发、编程模型，抽象化平台的演进。其中必须解决的问题就是无服务器平台的资源调度问题。本节将无服务器平台按照调度目标分为以下几类：面向资源利用、面向响应时间延迟以及面向多目标优化的无服务器平台资源调度。

3.1 面向资源利用的无服务器平台资源调度

由于现有平台的调度机制不能满足无服务器应用程序的独特特性，如突发、可变、无状态等，当前主要的方法是通过动态优化分配资源来提高无服务器平台的资源利用情况。Kaffes[21]等通过维护集群资源的全局视图来提高调度器的资源利用情况。他们使用集中的方式消除了队列不平衡，采用以核心为粒度的方式减少了干扰，并提出了一种集中式的内核粒度调度器。如图 3 所示[21]，每个调度器核心维护一个空闲的工作内核心引用列表，调度器使用负载平衡机制将传入的请求分配给调度程序内核(Scheduler Core)中的任意一个(i)，调度器内核从它的引用列表(ii)中取出一个空闲工作内核(Worker Core)，并将函数实例化请求调度到该内核中(iii)。当函数完成执行时，将通知调度器把这个工作内核添加到调度器内核的引用列表中(iv)(不一定是原来的引用列表)，如果一个请求到达调度程序内核，其引用列表为空(v)，它可以从另一个调度器内核(vi)的列表中窃取一个工作内核，并在那里调度函数执行(vii)。这种设计的好处在于，把准备执行的函数直接分配给空闲的工作内核，避免不必要的排队，消除队列不平衡，进而提高平台的资源利用率。

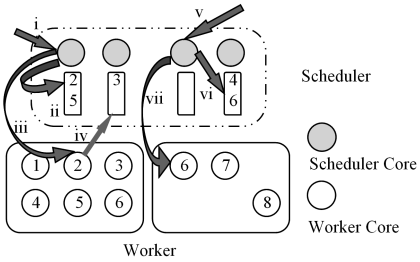

图 3 集中式内核粒度调度器

Fig. 3 Centralized core-granular scheduler

Jiang 等[22]运用混合作业分派方法提高了工作流的资源利用情况。他们同时考虑了工作流不同执行阶段的资源消耗模式，提出了一个无服务器的工作流执行系统。该系统旨在解决大规模科学工作流的执行和资源未充分利用等问题。该系统利用对象存储服务进行二进制数据分段，能够有效地处理 FaaS 执行环境中的最大执行时间限制、内存限制和存储空间限制。在大规模测试中，与传统的集群执行模型相比，其大幅度简化了在公共云上执行大规模科学工作流所需的工作。

3.2 面向响应时间延迟的无服务器平台资源调度

在无服务器资源调度上，当前的主要方法是通过减少冷启动时间或冷启动次数来降低函数的响应时间。FaaS 平台通常依赖于容器技术[23]：当开发人员部署函数时，这个函数代码与容器绑定在一起并存储在容器存储库中；当请求进入时，网关组件检查是否已经有一个空闲的容器实例可以服务于请求；当没有空闲容器时，网关分配一个新的容器并将请求定向到各自的机器。从确定没有空闲容器可用到请求提供服务容器之间的时间被称为冷启动延迟或冷启动时间[24-26]。在许多情况下，从客户端的角度来看，额外的冷启动增加了两倍的函数执行延迟。

小型云函数可以快速启动，因为它们运行在预先分配的虚拟机和容器中。但是，当这些函数依赖于大型包时，函数的启动就会变慢，从而影响应用程序的弹性，同时降低了应用程序对负载突发的快速响应能力[27]。一种解决方案是在工作节点上缓存包，而不是将它们与函数绑定在一起。现有的 FaaS 调度器只是普通的负载平衡器，在给工作节点分配函数时，调度器不会尝试包的最大化缓存。为了解决这个问题，Abad 等[28]尝试将需要相同包的函数分配给相同的工作节点，提出了包感知调度算法。在该算法中，调度器为每个工作器 $W=(w_1,\cdots,w_n)$ 跟踪一个先进先出的调度队列 $Q=(q_1,\cdots,q_n)$，并使用散列函数 H_1，H_2 根据任务所需要的最大的包 p_l 计算哈希值：

$$t_1=H_1(p_l)\%|W|+1$$

$$t_2=H_2(p_l)\%|W|+1$$

每个哈希函数将任务映射到不同的队列，并将任务分配到这些队列中最短的一个。当一个工作器有空闲容量时，它会联系调度程序请求任务分配，如果它跟踪的队列为空，调度器将选择最大负载的队列进行任务窃取。Abad 等[28]提出的算法提高了包缓存的命中率，降低了云函数的延迟；同时考虑了工作节点所承受的负载，避免超出可配置的阈值。

Oakes 等[29]试图通过将函数所需包缓存在工作节点来减少云函数的启动时间，并提出了共享包缓存(如图 4 所示)。该缓存器维护一组 Python 解释器，这些解释器的包是预先导入且处于睡眠状态。当一个云函数被分配给一个工作节点时，缓存器检查所需的包是否被缓存了。如果一个云函数需要的包在两个不同的休眠解释器中，那么只能使用其中一个，丢失的包必须加载到该容器的子容器中。为了处理具有多个包依赖关系的云函数，Pipsqueak 支持树缓存，其中一个条目可以缓存包 a，另一个条目可以缓存包 b，这两个条目的子条目都可以缓存 a 和 b。

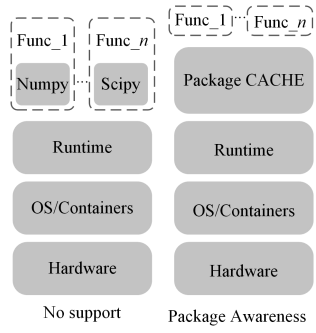

图 4 共享包缓存

Fig. 4 Caching of shared packages

当研究人员开始为 FaaS 平台解决包感知调度和其邻近数据调度优化问题时，需要有一个共同的框架来实现和评估他们的想法。Totoy 等[30]为了实现实验重现性，提出了

olscheduler，这是一个用于 OpenLambda[31] 平台的简单可扩展函数调度器，比 OpenLambda 使用的 ngninx 负载均衡器更容易修改和扩展。olscheduler 向系统开发人员公开有关平台的详细信息，以便其轻松实现其他调度策略。Totoy 等还在 olscheduler 中添加了包感知调度算法，试图提高包的亲和性，同时避免了过载。初步仿真结果表明，该方法可以将函数延迟降低 65%以上。

虽然使用一致性哈希算法[32-33]将需要特定包的所有函数请求路由到同一工作节点上可以最大限度地提高缓存命中率。然而，特别是在倾斜的工作负载[34]下，这种方法面临着负载不平衡的问题。Aumala 等[35]进一步研究了包感知调度的情况，试图在调度过程中寻找包的亲和性，从而使工作节点能够重用预加载包的执行环境，并提出了一种新的调度算法 PASch。在 PASch 算法中，存在全局工作器列表 $W=(w_1,\cdots,w_n)$及其负载上限阈值 $T=(t_1,\cdots,t_n)$，利用一致性哈希算法根据函数中最大需求包 p 计算哈希值：

$$a_1=consistenHash(p)$$

$$a_2=consistenHash(p+salt)$$

其中，$salt$ 为盐值。Aumala 等通过对比所对应工作器的负载 $\min(load(w_{a1}),load(w_{a2}))$，将任务分配到较小负载工作器中（如果工作器负载小于上限阈值），否则将任务分配到列表中的最小负载工作器中。

3.3 面向多目标优化的无服务器平台资源调度

3.3.1 面向响应时间和成本的调度

对于多数无服务器平台，无服务器函数都会在容器中先实例化然后再运行，Mahmoud 等[7]对无服务器函数的放置问题进行了研究，致力于最小化运行成本的同时优化运行函数的性能。他们提出并评估了一种基于神经网络的函数放置算法，与目前 FaaS 平台使用的放置算法相比，其可以提高函数的性能，而容器编排过程中的开销和延迟则可以忽略不计。Mahmoud 等研究了 3 种表示 CPU、内存和 I/O 密集型函数的通用函数，所提出的自适应功能放置算法很容易被无服务器计算提供商采用。

Bermbach 等[36]利用函数组合方面的应用知识来减少冷启动的次数。他们提出了客户端的互补方法并提供了一个轻量级的多平台编排中间件，其可以与无服务器函数一起部署。最后，Bermbach 等提出并讨论了 3 种基于减少冷启动次数的方法，作为实现上述中间件的一部分，同时将 FaaS 平台视为黑盒。大量的实验证明，该方法在较低成本的情况下平均冷启动时间可减少约 30%～40%（在某些情况下可减少高达 80%）。

在科学任务的工作流中，科学应用程序的工作流可以表示为有向无环图（DAG），其中每个计算任务都用一个节点表示，约束任务之间的关系用有向边表示[37-38]。特别地，节点（任务）表示计算作业，而边（关系）定义节点之间的依赖关系。具有依赖项约束的任务在父任务执行完成之前不能启动。Alqaryouti 等[39]将科学任务的工作流根据调度目标进行分类，即最小化时间、最小化执行成本和多目标（时间和成本）。最后其分析了使用 FaaS 对工作流进行任务调度的优势：可以远程运行小任务，并且只关注调度大任务，这有助于在减少响应时间的同时减少执行成本。

3.3.2 面向响应时间和资源利用的调度

Stein[40]回顾了 Apache OpenWhisk[41]无服务器事件负载平衡和分布式系统中响应时间最小化的非合作博弈论负载平衡方法。关联调度在高吞吐量计算中很常见，但是当提供商的目标是高资源利用率，而客户的目标是最小响应时间时，本地化就会带来挑战。资源利用率最大化和响应时间最小化之间的平衡是一个多目标问题。因此 Stein[40]又提出了一种非合作的在线分配启发式方法，允许调整响应时间和每个无服务器函数资源成本之间的平衡。而 Suresh[16]致力于解决在裸机服务器上调度用户功能，在提供可接受延迟的情况下，同时最小化提供商的开销，并提出了 FnSched 函数级别的调度器。FnSched 的工作方式是调节每个调用程序上协同函数的资源使用，并将负载集中在少数调用程序上以响应不同的流量，从而实现自动调标能力。FnSched 通过将无服务器函数划分为不同的类别来考虑其资源消耗和生存期模式。然后，根据传入函数的类别可伸缩地确定函数的位置。为了提供可接受的延迟，FnSched 通过在运行时动态地调节其 CPU 共享资源来减少共域函数之间的资源争用。除了单个主机之外，FnSched 还可以根据传入的工作负载需求弹性地添加和删除主机，从而实现自动伸缩。当执行中的函数的性能下降到某个阈值时，FnSched 会添加一个额外的主机。该算法显著地提高了资源效率，提升幅度高达 36%～55%，同时提供了可接受的应用程序延迟。

3.3.3 其他面向多目标优化的研究

Hoseiny 等[42]通过预测未来传入事件的速率和考虑终端用户请求的服务质量（QoS）来强制动态地扩展资源。他们提出了一种闭环（反馈）的资源分配控制器，控制器连续求解目标函数的公式如下：

$$\min J_\gamma=\sum_{t=\gamma+1}^{\gamma+f}\sum_{pk}(\gamma_1 C_{(U)}+\gamma_2 D_{pk,t}+\gamma_3 SW_t)$$

其中，$C_{(U)}$为资源利用率的剩余值，D_{pk}为 QoS 的违例值，SW 为更改当前配置的代价，f 为预测水平长度，γ 系数为权重。Hoseiny 等通过粒子群算法（PSO）求解目标函数，并使用两种现有的启发式方法对资源利用、QoS 冲突和可伸缩性这 3 个不同指标进行了评估研究。实验结果表明，该控制器对系统的整体资源利用率平均提高了 21%，同时 QoS 违例减少了近 3/4。

文献[43]提出了一种在无服务器环境下的实时伸缩平台，其可以在容器资源运行时自动伸缩，且不需要重新启动。Enes 等[43]使用大数据负载应用对其资源利用、响应时间和可伸缩性进行评估。实验结果表明，该无服务器平台在响应时间开销仅为 6%的情况下可将 CPU 利用率提高达 77%，同时在使用 32 个容器集群的情况下保持可伸缩性。

3.4 无服务器平台调度的比较

表 1 列出了平台调度方法的分析比较。

表1 无服务器平台调度方法分析比较

Table 1 Analysis and comparison of scheduling methods for serverless computing platform

文献	目标	方法描述	关键实现	工具	特点
[16]	资源利用、响应时间	试图调节每个调用程序上协同函数的资源使用，并将负载集中在少数调用程序上	在OpenWhisk之上实现了Fn-Sched调度器并测试评估所提贪婪自缩放算法	Apache OpenWhisk	在提供可接受延迟的情况下，同时最小化提供商的开销。但是，实验假设函数执行时间是可变的(这种情况不现实)
[7]	响应时间、执行成本	提出并评估了一种新的神经网络的函数放置算法	使用智能传播算法测试评估函数在Docker中的放置情况	Docker，Metricbeat，Cybera cloud	在最小化运行成本的同时优化运行函数的性能。但是，没有考虑到安全性、容错性等影响因素
[21]	资源利用、可伸缩性	使用集中方式消除队列不平衡，采用以核心为粒度的方式来减少干扰	实现了集中式的内核粒度调度器	eRPC，IX	可以克服可伸缩性的挑战，在短时间内产生大量的函数
[42]	资源利用、响应时间、QoS冲突、可伸缩性	通过预测未来传入事件的速率和考虑终端用户请求的服务质量，强制动态地扩展资源	使用Lambda平台测试提出的闭环(反馈)的资源分配控制器	Xen hypervisor，python2.7	提高了资源利用情况、伸缩性以及减少了QoS违例。但是，争用共享资源可能会导致整体性能下降，还会增加整个系统的功耗
[22]	资源利用	利用混合作业分派方法，并考虑了工作流不同执行阶段的资源消耗模式	评估DEWE v3在云函数上的性能，进行了初始评估、性能调优策略和大规模评估	AWS Lambda	解决了大规模科学工作流的执行和资源未得到充分利用等问题，从而减少了云成本。但是，没有考虑到减少函数的响应延迟
[28]	响应时间	尝试将需要相同包的函数分配给相同的工作节点	在Python中实现了模拟器，并测试包感知调度算法	SimPy simulation framework	提高了包缓存的命中率，从而降低了云函数的延迟。但是，对最佳过载阈值没有自动调优功能
[29]	响应时间	通过将函数所需包缓存在工作节点来减少云函数的启动时间	提出并评估可感知包的计算平台Pipsqueak	OpenLambda	提高了包缓存的命中率，从而降低了云函数的延迟，同时支持树缓存，提高了共享包的缓存能力
[30]	响应时间	重构OpenLambda调度器并添加了文献[26]的包感知调度算法	在OpenLambda添加负载均衡器olscheduler，并测试评估包感知算法	OpenLambda	促进了实验的重现性
[35]	响应时间	试图寻找包的亲和性，从而使工作节点能够重用预加载包的执行环境	实现调度器PASch，并使用模拟和真实实验对其进行评估	OpenLambda	提高了共享包的缓存能力。但是，没有考虑到不同动态工作负载下的性能
[36]	响应时间、执行成本	利用函数组合方面的应用知识来减少冷启动的次数	提出轻量级多平台编排中间件并在AWS Lambda和OpenWhisk上进行测试评估	AWS Lambda，Apache OpenWhisk	减少了函数响应延迟，同时降低了执行成本。但是，没有考虑到其他QoS因素
[40]	资源利用、响应时间	提出了非合作的在线分配启发式方法，实现调整响应时间和每个无服务器函数资源成本之间的平衡	开发了无服务器的模拟工具来模拟Apache OpenWhis体系结构并测试评估提出的NOAH算法	SimPy simulation framework，Apache OpenWhisk	对响应时间和资源利用之间进行平衡，且允许对其进行调整
[43]	资源利用、响应时间、可伸缩性	提出了可伸缩性的无服务器平台，可自动伸缩CPU和内存，以适应实际使用	在可伸缩平台上使用大数据负载应用(流应用、批处理应用)进行评估	BDWatchdog，OpenTSDB	在有限的时间里大幅提高了资源利用率，且在32个容器集群中保持良好的伸缩性能

4 未来研究的趋势

无服务器计算是一个新的领域，同时也具有一些独有的特性，如突发性、无状态性、没有明确的配置、自动且几乎无限的扩展。这些特性吸引了业界和学术界的许多学者对其进行研究，其中热点问题就是无服务器函数的资源调度问题，但当前的研究仍然存在不少问题亟待解决。我们认为无服务器平台的资源调度的研究趋势包含以下几个方面。

4.1 面向不同类型应用负载的调度优化

当前提出的无服务器资源调度方式大多是面向无服务器平台本身的。然而，已有的方法很少考虑到面向不同类型的应用负载，而对于不同的应用程序需要考虑完全不同的资源消耗模式。

未来需要进一步研究面向不同类型应用负载的无服务器函数资源调度方法和机制。例如，基于边缘计算的无服务器调度平台需要考虑网络边缘的有限资源造成的超载问题[44-46]；基于分布式机器学习的无服务器调度平台需要在训练过程中动态调整资源配置。多数模型的资源需求随着训练迭代次数的增加而减少，投入到机器学习训练中的资源数量应该随着工作的进行而动态调整，以最大限度地提高模型质量[47]。

4.2 响应时间与资源利用率的折衷调度

当前绝大多数无服务器函数的资源调度方法很少同时考虑到响应延迟时间和资源利用能力。虽然提供商通过提高资源利用率来获取更大的收益，但这往往会导致响应时间的延长，违背了客户的SLA。理想情况下，云应用程序的所有者希望维护用户的SLA[8-9]，从而不会导致更高的成本或增加系统复杂性。为此，响应时间与资源利用率折衷的无服务器资源调度是无服务器调度的重中之重。Suresh等[16]提出的FnSched函数级别调度器显著提高了资源效率，同时提供了可接受的应用程序延迟，但是算法的执行效率有待进一步提高。而Hendrickson等[40]提出的非合作的在线分配启发式方法(NOAH)允许调整响应时间和每个无服务器函数资源成本之间的平衡，但是他们假设函数执行时间是可变的，这种假设在公有云的情况下显然是不成立的，需要进一步考虑公有云中响应时间与资源利用的折衷调度。

4.3 虚拟机和无服务器平台的联合调度

多数 FaaS 平台上，函数首先会注入容器中，然后该容器将运行在虚拟机或物理机器上。随着无服务器技术的逐渐成熟，无服务器平台和虚拟机并存已经成为一种必然现象。目前多数调度技术都是针对无服务器平台本身。对于虚拟机和无服务器联合调度是否能更好地提高资源利用率、降低能耗或者最小化客户成本的问题，有待未来的进一步研究。

4.4 无服务器资源调度的混合算法

无服务器资源调度的特点和各类指标的性质，使得很难有某种算法能够同时完全满足各种需求。现有的调度算法往往只能针对调度中的某几项指标进行优化，因此越来越多的研究中采取了复合算法。例如，粒子群算法(PSO)具有容易实现、收敛较快的优点，但同时存在容易陷入局部最优解的缺点。将 PSO[48] 和其他群智能算法(SA，ACO[49]，BSAS[50] 等)相结合，可以分别进行不同侧重点的优化，同时也提高了多目标优化中目标之间的最佳平衡性。

结束语 无服务器计算平台是一种新型云计算范式，它由于不需要明确的配置、自动且几乎无限的扩展性，并按需计费的优势而更接近于云计算最初的期望。该模式的转变虽然带来了机遇，同时也带来了平台函数冷启动延迟、资源利用不足的问题。本文对无服务器计算平台的资源调度技术做了深入的调查和分析，重点阐述了面向资源利用、响应时间延迟以及多目标优化的无服务器平台资源调度，并指明了未来研究的趋势。

参 考 文 献

[1] KILCIOGLU C, RAO J M, KANNAN A, et al. Usage patterns and the economics of the public cloud[C]// Proceedings of the 26th International Conference on World Wide Web. 2017: 83-91.

[2] CNCF Serverless Whitepaper v1.0 [EB/OL]. (2020-05-23) [2020-08-31]. https://github.com/cncf/wg-serverless # whitepaper.

[3] MCGRATH G, BRENNER P R. Serverless computing: Design, implementation, and performance[C]// 2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW). IEEE, 2017: 405-410.

[4] Global Forecast to 2021: Increasing shift from DevOps to serverless computing to drive the overall Function-as-a-Service market [EB/OL]. (2017-06-15) [2020-08-31]. https://www.businesswire.com/news/home/20170227006262/en/7.72-Billion-Function-as-a-Service-Market-2017---Global.

[5] DELIMITROU C, KOZYRAKIS C. Quasar: resource-efficient and QoS-aware cluster management[J]. ACM SIGPLAN Notices, 2014, 49(4): 127-144.

[6] 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 the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems. 2019: 3-18.

[7] MAHMOUDI N, LIN C, KHAZAEI H, et al. Optimizing serverless computing: introducing an adaptive function placement algorithm[C]// Proceedings of the 29th Annual International Conference on Computer Science and Software Engineering. 2019: 203-213.

[8] CASTRO P, ISHAKIAN V, MUTHUSAMY V, et al. The rise of serverless computing[J]. Communications of the ACM, 2019, 62(12): 44-54.

[9] ISHAKIAN V, MUTHUSAMY V, SLOMINSKI A. Serving deep learning models in a serverless platform[C]// 2018 IEEE International Conference on Cloud Engineering (IC2E). IEEE, 2018: 257-262.

[10] BALDINI I, CASTRO P, CHANG K, et al. Serverless computing: Current trends and open problems[M]// Research Advances in Cloud Computing. Springer, Singapore, 2017: 1-20.

[11] HELLERSTEIN J M, FALEIRO J, GONZALEZ J E, et al. Serverless computing: One step forward, two steps back[J]. arXiv: 1812.03651, 2018.

[12] FOX G C, ISHAKIAN V, MUTHUSAMY V, et al. Status of serverless computing and function-as-a-service(faas) in industry and research[J]. arXiv: 1708.08028, 2017.

[13] ADZIC G, CHATLEY R. Serverless computing: economic and architectural impact[C]// Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering. 2017: 884-889.

[14] JONAS E, SCHLEIER-SMITH J, SREEKANTI V, et al. Cloud programming simplified: A berkeley view on serverless computing[J]. arXiv: 1902.03383, 2019.

[15] WANG L, LI M, ZHANG Y, et al. Peeking behind the curtains of serverless platforms[C]// Annual Technical Conference. 2018: 133-146.

[16] SURESH A, GANDHI A. FnSched: An Efficient Scheduler for Serverless Functions[C]// Proceedings of the 5th International Workshop on Serverless Computing. 2019: 19-24.

[17] LEE H, SATYAM K, FOX G. Evaluation of production serverless computing environments[C]// 2018 IEEE 11th International Conference on Cloud Computing (CLOUD). IEEE, 2018: 442-450.

[18] WAGNER T. Optimizing Enterprise Economics with Serverless Architectures[Z]. AWS Web Serviced, 2017.

[19] Netflix & AWS Lambda Case Study[EB/OL]. (2020-08-18) [2020-08-31]. https://aws.amazon.com/cn/solutions/case-studies/netflix-and-aws-lambda/.

[20] WU D, ROSEN D W, WANG L, et al. Cloud-based design and manufacturing: A new paradigm in digital manufacturing and design innovation[J]. Computer-Aided Design, 2015, 59: 1-14.

[21] KAFFES K, YADWADKAR N J, KOZYRAKIS C. Centralized Core-granular Scheduling for Serverless Functions[C]// Proceedings of the ACM Symposium on Cloud Computing. 2019: 158-164.

[22] JIANG Q, LEE Y C, ZOMAYA A Y. Serverless execution of scientific workflows[C]// International Conference on Service-Oriented Computing. Springer, Cham, 2017: 706-721.

[23] PÉREZ A, MOLTÓ G, CABALLER M, et al. Serverless computing for container-based architectures[J]. Future Generation Computer Systems, 2018, 83: 50-59.

[24] BALDINI I, CHENG P, FINK S J, et al. The serverless trilem-

ma: Function composition for serverless computing[C]// Proceedings of the 2017 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software. 2017: 89-103.

[25] LLOYD W, VU M, ZHANG B, et al. Improving Application Migration to Serverless Computing Platforms: Latency Mitigation with Keep-Alive Workloads[C]// 2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion). IEEE, 2018: 195-200.

[26] MANNER J, ENDREβ M, HECKEL T, et al. Cold start influencing factors in function as a service[C]// 2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion(UCC Companion). IEEE, 2018: 181-188.

[27] VAN E E, IOSUP A, ABAD C L, et al. A SPEC RG cloud group's vision on the performance challenges of FaaS cloud architectures[C]// Companion of the 2018 ACM/SPEC International Conference on Performance Engineering. 2018: 21-24.

[28] ABAD C L, BOZA E F, VAN EYK E. Package-aware scheduling of FaaS functions[C]// Companion of the 2018 ACM/SPEC International Conference on Performance Engineering. 2018: 101-106.

[29] OAKES E, YANG L, HOUCK K, et al. Pipsqueak: Lean Lambdas with large libraries[C]// 2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW). IEEE, 2017: 395-400.

[30] TOTOY G, BOZA E F, ABAD C L. An Extensible Scheduler for the OpenLambda FaaS Platform[C]// Min-Move workshop(co-located with ASPLOS). 2018.

[31] HENDRICKSON S, STURDEVANT S, HARTER T, et al. Serverless computation with openlambda[C]// 8th (USENIX) Workshop on Hot Topics in Cloud Computing (HotCloud 16). 2016.

[32] KARGER D, SHERMAN A, BERKHEIMER A, et al. Web caching with consistent hashing[J]. Computer Networks, 1999, 31(11-16): 1203-1213.

[33] STOICA I, MORRIS R, KARGER D, et al. Chord: A scalable peer-to-peer lookup service for internet applications[J]. ACM SIGCOMM Computer Communication Review, 2001, 31(4): 149-160.

[34] NASIR A, MORALES G D F, GARCíA-SORIANO D, et al. The Power of Both Choices: Practical Load Balancing for Distributed Stream Processing Engines[J]. arXiv: 1504.00788v1, 2015.

[35] AUMALA G, BOZA E, ORTIZ-AVILES L, et al. Beyond Load Balancing: Package-Aware Scheduling for Serverless Platforms [C]// 2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID). 2019: 282-291.

[36] BERMBACH D, KARAKAYA A S, BUCHHOLZ S. Using application knowledge to reduce cold starts in FaaS services[C]// Proceedings of the 35th Annual ACM Symposium on Applied Computing. 2020: 134-143.

[37] RODRIGUEZ M A, BUYYA R. Deadline based resource provisioningand scheduling algorithm for scientific workflows on clouds[J]. IEEE transactions on cloud computing, 2014, 2(2): 222-235.

[38] RYAN T, LEE Y C. Effective resource multiplexing for scientific workflows[C]// 2015 17th Asia-Pacific Network Operations and Management Symposium(APNOMS). IEEE, 2015: 232-237.

[39] ALQARYOUTI O, SIYAM N. Serverless Computing and Scheduling Tasks on Cloud: A Review[J]. American Scientific Research Journal for Engineering, Technology, and Sciences (ASRJETS), 2018, 40(1): 235-247.

[40] STEIN M. The Serverless Scheduling Problem and NOAH[J]. arXiv: 1809.06100v1, 2018.

[41] QUEVEDO S, MERCHÁN F, RIVADENEIRA R, et al. Evaluating Apache OpenWhisk-FaaS[C]// 2019 IEEE Fourth Ecuador Technical Chapters Meeting(ETCM). IEEE, 2019: 1-5.

[42] HOSEINYFARAHABADY M R, LEE Y C, ZOMAYA A Y, et al. A QoS-aware resource allocation controller for function as a service(FaaS) platform[C]// International Conference on Service-Oriented Computing. Springer, Cham, 2017: 241-255.

[43] ENES J, EXPÓSITO R R, TOURIÑO J. Real-time resource scaling platform for Big Data workloads on serverless environments[J]. Future Generation Computer Systems, 2020, 105: 361-379.

[44] GAND F, FRONZA I, EL IOINI N, et al. Serverless Container Cluster Management for Lightweight Edge Clouds[C]// CLOSER. 2020: 302-311.

[45] NASTIC S, RAUSCH T, SCEKIC O, et al. A serverless real-time data analytics platform for edge computing[J]. IEEE Internet Computing, 2017, 21(4): 64-71.

[46] CICCONETTI C, CONTI M, PASSARELLA A. An architectural framework for serverless edge computing: design and emulation tools[C]// 2018 IEEE International Conference on Cloud Computing Technology and Science(CloudCom). IEEE, 2018: 48-55.

[47] ZHANG H, STAFMAN L, OR A, et al. Slaq: quality-driven scheduling for distributed machine learning[C]// Proceedings of the 2017 Symposium on Cloud Computing. 2017: 390-404.

[48] MARINI F, WALCZAK B. Particle swarm optimization(PSO). A tutorial[J]. Chemometrics and Intelligent Laboratory Systems, 2015, 149: 153-165.

[49] GUNTSCH M, MIDDENDORF M. A population based approach for ACO[C]// Workshops on applications of evolutionary computation. Berlin, Heidelberg: Springer, 2002: 72-81.

[50] WANG J, CHEN H. BSAS: Beetle swarm antennae search algorithm for optimization problems[J]. arXiv: 1807.10470, 2018.

MA Ze-hua, born in 1996, postgraduate. His main research interests include cloud computing and container scheduling.

LIN Wei-wei, born in 1980, Ph.D, professor, is a member of China Computer Federation. His main research interests include cloud computing, big data technology and AI application technology.