计算机科学
COMPUTER SCIENCE

云边协同环境下面向负载时间窗口的无服务器应用资源分配方法

张铭豪，肖博怀，郑松，陈星

引用本文

张铭豪，肖博怀，郑松，陈星. 云边协同环境下面向负载时间窗口的无服务器应用资源分配方法[J]. 计算机科学，2025，52(6)：336-345.

ZHANG Minghao, XIAO Bohuai, ZHENG Song, CHEN Xing. Resource Allocation Method with Workload-time Windows for Serverless Applications inCloud-edge Collaborative Environment [J]. Computer Science, 2025, 52(6): 336-345.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.240400073

云边协同环境下面向负载时间窗口的无服务器应用资源分配方法

张铭豪[1,2]　肖博怀[1,2]　郑　松[3,4]　陈　星[1,2]
1 福州大学计算机与大数据学院　福州 350108
2 福建省网络计算与智能信息处理重点实验室　福州 350108
3 福州大学电气工程与自动化学院　福州 350108
4 福州大学工业自动化控制技术与信息处理福建省高校重点实验室　福州 350108
(211020024@fzu.edu.cn)

摘　要　随着云边协同环境中的计算需求日益多样化,以虚拟机为最小资源粒度的传统计算架构暴露出灵活性不足、成本效益低下等问题。无服务器计算作为一种具有出色扩展性与灵活性的新兴计算架构,为解决上述问题提供了新的思路。针对云边协同环境下面向负载时间窗口的无服务器应用资源分配问题,提出了一种规则引导下基于协同进化算法的无服务器应用资源分配方法 RARCA。该方法考虑某资源调整时刻及未来一段时间的工作负载情况,运用规则引导的分布式资源更新机制,实现计算资源的动态分配与调整。同时,协同进化机制的信息共享与协同优化能力,使得算法能够高效搜索全局最优的资源分配方案,显著提升了整体资源分配方案的实时性和有效性。实验结果表明,RARCA 能够以秒级的决策时间获得更优质的资源分配方案,相比基准方法,在资源分配的性能上提高了 2.8%～14.5%。

关键词:云边协同;资源分配;负载时间窗口;无服务器计算;协同进化算法
中图分类号　TP393

Resource Allocation Method with Workload-time Windows for Serverless Applications in Cloud-edge Collaborative Environment

ZHANG Minghao[1,2],XIAO Bohuai[1,2],ZHENG Song[3,4] and CHEN Xing[1,2]
1 College of Computer and Data Science,Fuzhou University,Fuzhou 350108,China
2 Fujian Provincial Key Laboratory of Networking Computing and Intelligent Information Processing,Fuzhou 350108,China
3 College of Electrical Engineering and Automation,Fuzhou University,Fuzhou 350108,China
4 Key Laboratory of Industrial Automation Control Technology and Information Processing(Fuzhou University) of Fujian Province,Fuzhou 350108,China

Abstract　With the increasingly diverse computational demands in the cloud-edge collaborative environment,the traditional computing architecture based on virtual machines as the smallest unit of resources exhibits inflexibility and low cost-effectiveness. Serverless computing,as an emerging computing architecture with excellent scalability and flexibility,provides a new perspective to address these issues. In response to the resource allocation problem with workload-time windows for serverless applications in cloud-edge collaborative environments,this study proposes a resource allocation method based on rule-driven co-evolution algorithm(RARCA). This method considers the workload at a certain resource adjustment moment and in the foreseeable future,employing a rule-driven distributed resource updating mechanism to achieve dynamic allocation and adjustment of computing resources. Additionally,by leveraging the information sharing and cooperative optimization capabilities of the co-evolution mechanism,the algorithm efficiently searches for globally optimal resource allocation solutions,significantly improving the real-time and effectiveness of the overall resource allocation method. Experimental results demonstrate that RARCA can achieve superior resource allocation solutions with decision times in seconds,outperforming baseline methods by 2.8% to 14.5% in resource allocation performance.

到稿日期:2024-04-11　返修日期:2024-08-12
基金项目:国家自然科学基金(62072108);福建省科技经济融合服务平台(2023XRH001);福厦泉国家自主创新示范区协同创新平台项目(2022FX5);福建省促进海洋与渔业产业高质量发展专项资金(FJHYF-ZH-2023-02)
This work was supported by the National Natural Science Foundation of China(62072108),Science and Technology and Economic Integration Service Platform of Fujian Province(2023XRH001),Collaborative Innovation Platform Project of Fu Xia Quan National Independent Innovation Demonstration Area(2022FX5) and Special Funds for Promoting the High-quality Development of Marine and Fishery Industry of Fujian Province(FJHYF-ZH-2023-02).
通信作者:陈星(chenxing@fzu.edu.cn)

Keywords Cloud-edge collaborative, Resource allocation, Workload-time windows, Serverless computing, Co-evolutionary algorithm

1 引言

随着物联网(Internet of Things，IoT)的快速发展，数以百万计的智能设备被部署在复杂的网络中，以提供多样化的功能[1]。然而，网络边缘产生的巨大数据流量以及用户对服务质量的更高要求，导致传统的集中式云计算模式[2]面临响应时延高、工作负载量大等挑战。为了解决上述问题，边缘计算成为一种创新策略。它将数据处理和存储部署在更接近终端用户的地方，从而降低云端压力。云计算与边缘计算各有优势，近期许多工作[3-4]将二者结合起来，提出了云边协同的计算模式。在云边协同环境中，靠近终端用户的边缘设备是时间敏感决策的最佳选择，而云数据中心则适用于计算密集型任务[5]。云边协同计算使服务提供商能为 IoT 用户设备的多样性请求提供更加灵活的计算服务。

制定与优化资源分配方案以降低能耗或延迟成本，已成为云边协同计算环境下高效处理任务的关键[6]。现有的针对资源分配问题的相关研究工作主要集中在基于虚拟机资源分配的传统计算模式中，但是这种模式存在灵活性低下的问题。无服务器作为一种灵活的云计算范式，为上述问题提供了有效的解决方案。与传统计算模式相比，无服务器计算的主要优势在于开发人员可以透明地使用计算资源，而常见的资源管理任务如资源供应、部署、自动伸缩等由服务提供商负责[7]。在使用无服务器功能时，开发人员不再需要考虑资源管理，而是根据实际使用时长进行资源付费[8]。无服务器计算以其灵活的自动扩展能力和即用即付的收费模式，被视为应用服务部署的一种全新解决方案。主流云服务提供商，如亚马逊云服务和谷歌云平台，都推出了自己的无服务器计算平台，各类应用也正以无服务器函数的方式重构执行[9]。在无服务器计算环境中，制定有效的应用资源分配策略是实现资源动态高效管理、提高应用性能和降低运营成本的关键。

在云边协同环境中，无服务器应用的工作负载波动较大，此时系统需要实时调整计算资源，并尽可能保证资源分配方案的有效性。现有的一些研究工作[10-11]，已经成功实现了对动态工作负载环境的高精度预测，能够精准预估系统在未来某一时间窗口内的负载状况。在这些研究的基础上，通过引入工作负载时间窗口，可以预知未来一段时间内系统可能的工作负载情况，制定出更为精准和高效的资源分配方案[12]。

基于上述考虑，本文提出规则引导下基于协同进化算法的资源分配方法(Resource Allocation Method Based on Rule-driven Co-evolution Algorithm，RARCA)。在云边协同环境中，假定系统已知未来一段时间的工作负载，考虑计算系统中时间窗口内的负载情况，以秒级的决策时间给出合理有效的无服务器应用资源分配方案，在降低任务响应时间的同时，减少租用计算资源的开销，并在二者之间取得良好的平衡。

2 相关工作

近年来，云边协同环境下的应用资源分配问题得到学者们的广泛关注。Zhou 等[13]提出了一种基于反向拍卖的计算卸载和资源分配机制，首先基于约束梯度下降分配方法来确定资源分配策略，然后基于贪婪随机自适应搜索过程的获胜投标调度方法来确定计算卸载策略。Yuan 等[14]提出了一种利润最大化的协同计算卸载和资源分配算法，通过考虑资源限制、负载平衡要求、边缘节点的处理能力以及云数据中心的服务器和能量限制，在满足任务响应时间限制的同时最大化云边协同计算系统的利润。Gan 等[15]提出了一种针对云边协同 IoT 应用的模型和数据双驱动资源优化机制，在离线学习阶段构建基于最优传输的离线优化模型来解决带宽和计算资源分配优化问题，在在线学习阶段构建基于联邦强化学习的优化模型进行在线资源优化，进一步降低延迟和能耗。然而，现有的云边协同环境下的资源分配研究大多针对基于虚拟机资源的传统计算模型，其以虚拟机为最小粒度的资源预留将会导致相当大的资源浪费。以函数容器为最小资源粒度的无服务器计算可以解决这些问题，但目前很少有工作在云边协同环境下考虑基于无服务器计算的应用资源分配问题。

无服务器计算具有快速响应要求和细粒度的特点，以及按需付费的计费模式，无法直接套用基于虚拟机的传统计算模式的资源分配方法。现有的无服务器应用资源分配问题的科研工作较少，且主要聚焦于云或边缘[16-17]。Ascigil 等[18]考虑基于无服务器计算的边缘环境，提出了从完全集中到完全分散的资源分配和供应算法。实验结果表明，去中心化的方法几乎没有协调和通信开销，但依然能够实现与完全集中式方法相当的性能。Kim 等[19]提出了一种基于性能指标的组感知调度算法，其能够在考虑工作负载的波动的前提下，管理多租户无服务器平台的计算资源。该算法可动态调整具有相同/相似性能需求的应用的 CPU 使用限制，以减少资源争用。Raza 等[20]提出的基于贝叶斯优化的算法，为无服务器应用程序中的函数找到了最佳资源配置和位置。该算法利用统计学习技术智能地收集样本，并预测无服务器函数在未知配置值下的成本和执行时间，以此为无服务器应用程序的运行选择最佳配置参数和位置，同时满足客户的要求。在云计算环境中，无服务器架构适合弹性扩展和计算密集的应用场景。在边缘计算环境中，无服务器架构则适合低延迟需求和距离敏感的应用场。因此，单一的云计算或边缘计算环境的无服务器应用资源分配方案，难以满足物联网设备产生的复杂多样的计算需求。

综上所述，目前的研究工作已经在云边协同环境下的应用资源分配问题中取得较大进展，但多数研究仍局限于针对虚拟机资源分配的传统计算模型。与此同时，关于无服务器计算的资源分配研究则聚焦于单一的云计算或边缘计算环境，尚未充分探索其在云边协同场景下的应用潜力和优势。因此，面对网络状况更复杂的云边协同环境以及灵活性与实时性要求更高的无服务器计算模型，应用程序的计算资源分配问题依然是一个具有挑战性和现实意义的问题。

3 问题形式化

本章将介绍云边协同环境下面向负载时间窗口的无服务

器应用资源分配问题，并对该问题进行形式化定义。

考虑在特定区域内部署基于无服务器计算的应用服务。该区域包括数个通信基站和边缘计算节点，并配备远程云计算中心（即云计算节点）。通信基站统一接收来自物联网设备的任务请求，并定期调度这些任务请求。随后，这些请求数据被分发至计算节点，各节点根据自身资源的情况选择适当的容器来执行无服务器函数，以处理这些任务请求。无服务器函数通常很小且无状态，应用将被拆分为不同的无服务器函数[21]，本文考虑由规格相同的无服务器函数容器执行特定函数请求。系统整体环境如图 1 所示。

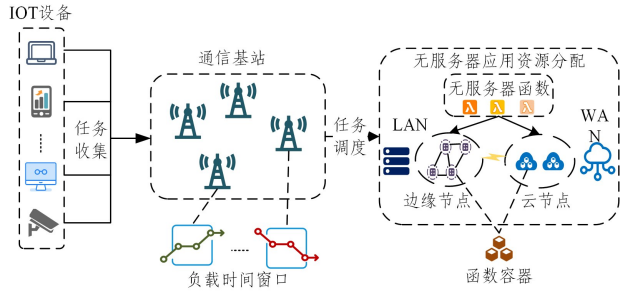


图 1 系统环境示意图

Fig. 1 Schematic diagram of system environment

本文中，区域内的通信基站表示为 $BS=\{1,2,\cdots,n\}$，计算节点表示为 $E=\{0,1,\cdots,m\}$，其中 0 代表云计算节点，1 到 m 代表边缘计算节点。节点的无服务计算资源 CT 由该节点当前部署的可用函数容器数量来表示：

$$CT=\{ct_j(t_k)\mid j\in E,t_0\leqslant t_k\leqslant t_0+T\} \tag{1}$$

其中，$ct_j(t_k)$ 表示计算节点 j 在 t_k 时刻部署的函数容器数量，$t_0\leqslant t_k\leqslant t_0+T$ 表示可调整资源的时间段。此外，定义 Δt 为调整无服务器应用资源分配方案的时间间隔。需要注意的是，区别于持有大量资源的云计算节点，边缘计算节点由于设备限制，提供的计算资源相对有限，因此边缘计算节点中可部署的最大函数容器数量受到该计算节点的设备规格的约束。

在本文的系统环境中，网络带宽保持恒定。基站与计算节点之间的距离决定了数据传输速率，可以用二维矩阵 $\boldsymbol{R}$ 来表示：

$$\boldsymbol{R}=\begin{pmatrix} r_{1,0} & \cdots & r_{1,m} \\ \vdots & \ddots & \vdots \\ r_{n,0} & \cdots & r_{n,m} \end{pmatrix} \tag{2}$$

其中，$r_{i,j}(\forall i\in BS;\forall j\in E)$ 表示基站 i 与节点 j 之间的传输速率。

通信基站的工作负载体现为每个时间周期内收集到的服务范围内的无服务函数请求，这些任务请求将被调度到合适的计算节点上执行。通信基站的工作负载可以表示为：

$$W=\{w_i(t_k)\mid i\in BS,t_0\leqslant t_k\leqslant t_0+T\} \tag{3}$$

其中，$w_i(t)$ 表示基站 i 的工作负载，其在 t_k 时刻的工作负载表示为 $w_i(t_k)$。对于任务 $p\in w_i(t_k)$，其携带的数据量被定义为 $data_i^p(t_k)$。

当制定资源分配计划时，仅依赖当前工作负载进行决策，无法很好地确保资源分配的有效性[22]。因此，本文考虑了负载变化，并引入负载时间窗口的概念，利用未来的工作负载信息支持当前的资源分配决策，从而提高无服务器应用资源分配方案的有效性。负载时间窗口的定义如图 2 所示。

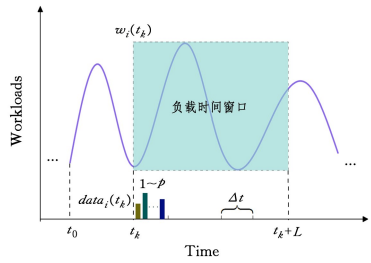


图 2 负载时间窗口的定义

Fig. 2 Definition of workload-time window

假设可预测工作负载的时间长度为 $L(L\ll T)$，则负载时间窗口中 t_k 时刻的工作负载可表示为：

$$W^L=\{w_i(t_k)\mid i\in BS,t_k\leqslant t\leqslant t_k+L\} \tag{4}$$

资源供应成本与用户任务请求的响应时间是影响资源分配策略的主要因素。一个有效的资源分配策略能够在降低整体资源成本和缩短用户响应时间之间取得平衡。

一方面，在面向负载时间窗口的场景下，资源成本受时间窗口内的容器部署方案以及无服务函数容器的租赁单价影响。计算节点接收来自各个通信基站的无服务器函数请求，并提供函数容器来执行这些任务。针对到达计算节点的任务请求，节点采用先来先服务（First Come First Service，FCFS）的方式执行。计算节点维护一个任务等待队列，根据任务的到达顺序将任务按序置入队列，并依次执行。在任务执行过程中，函数容器不会被其他任务抢占，其执行速率基于节点的容器规格。本文假设所有计算节点统一容器规格，函数容器的计算能力表示为 γ，容器初始化时间（即冷启动时间）表示为 T_{cold}。

服务提供商根据计算节点的地理位置等因素制定不同的资源收费标准，导致各个节点的无服务函数容器的租赁价格各不相同。对于节点 $j\in E$，其容器租赁价格表示为 λ_j。计算节点的无服务器应用资源分配方案体现为当前时刻各个节点租赁的容器数，具体来说，t_{k+1} 时刻节点 j 的容器数量 $ct_j(t_{k+1})$ 为 t_k 时刻保留的容器数 $ct^j_{\text{reserve}}(t_k)$ 加上 t_{k+1} 时刻新增的容器数 $ct^j_{\text{add}}(t_{k+1})$。因此对于节点 $j\in E$，根据节点的资源分配方案，容器保留量 $ct^j_{\text{reserve}}(t_k)$ 和新增量 $ct^j_{\text{add}}(t_{k+1})$ 可通过式(5)和式(6)计算得到。

$$ct^j_{\text{reserve}}(t_k)=\begin{cases} ct_j(t_k), & ct_j(t_{k+1})\geqslant ct_j(t_k) \\ ct_j(t_{k+1}), & \text{otherwise} \end{cases} \tag{5}$$

$$ct^j_{\text{add}}(t_{k+1})=\begin{cases} ct_j(t_{k+1})-ct_j(t_k), & ct_j(t_{k+1})\geqslant ct_j(t_k) \\ 0, & \text{otherwise} \end{cases} \tag{6}$$

同时，t_{k+1} 时刻的资源成本表示为 $Cost(t_{k+1})$，可以由式(7)计算得到。

$$\begin{aligned} Cost(t_{k+1})&=Cost_{\text{reserve}}(t_k)+Cost_{\text{add}}(t_{k+1}) \\ &=\lambda_j\sum_{j=0}^{m}\left[ct^j_{\text{reserve}}(t_k)\times\Delta t+ct^j_{\text{add}}(t_{k+1})\times(\Delta t+T_{\text{cold}})\right] \end{aligned} \tag{7}$$

此外，在负载时间窗口内，可选资源分配方案定义为：

$$CT^L_{\text{opt}}=\{CT_{t_k}\mid t_k\leqslant t\leqslant t_k+L\}_{\text{opt}} \tag{8}$$

基于计算节点的无服务器应用资源分配方案，通信基站在每个时间周期将收集到的无服务器函数任务请求调度至合适的计算节点中执行。本文所提出的方法主要解决无服务器计算环境下的资源分配问题，在任务调度策略方面，采用基于

贪心算法的决策方法。基于贪心算法的任务调度策略综合考虑了任务响应时间和资源总成本，在决策过程中，总是做出当前状态下的最优选择。具体而言，在考虑第 i 个任务调度时，基于前 $i-1$ 个任务的决策结果，贪心算法将任务调度至能够使系统整体性能最高的计算节点。在最大效益驱动的多任务调度问题中，贪心算法的近似比为 2[23]，表明在最坏情况下，算法的性能与最优解的差距不超过 2 倍。根据任务调度结果，任务 $p\in w_i(t_k)$ 的执行节点可以表示为 $sn_i^p(t_k)\in E$。

另一方面，用户任务请求的响应时间由计算节点的容器资源分配方案与任务调度策略确定。本文定义了以下几个关键时间，并据此推导任务的平均响应时间。

1）传输时间：t_k 时刻，任务 $p\in w_i(t_k)$ 从基站传输至其执行节点 $j=sn_i^p(t_k)$ 的数据传输时间由任务 p 的数据量以及通信基站与计算节点之间的传输速率决定。

$$T_{\text{trans}}(t_k,i,p)=\frac{data_i^p(t_k)}{r_{i,j}} \tag{9}$$

2）执行时间：t_k 时刻，任务 $p\in w_i(t_k)$ 在容器中的执行时间由任务 p 的数据量与容器的计算能力 γ 决定。

$$T_{\text{exec}}(t_k,i,p)=\frac{data_i^p(t_k)}{\gamma} \tag{10}$$

3）响应时间：t_k 时刻，任务 $p\in w_i(t_k)$ 经过 $T_{\text{trans}}(t_k,i,p)$ 时间后，到达其执行节点的等待队列中，一旦任务开始执行，将花费 $T_{\text{exec}}(t_k,i,p)$ 的时间结束任务。任务的响应时间记为 $T_{\text{resp}}(t_k,i,p)$，其数值取决于任务的传输时间、执行时间以及节点资源量和等待队列中的前置任务。

4）平均响应时间：t_k 时刻，任务的平均响应时间 $avgT_{\text{resp}}(t_k)$ 可以通过式(11)计算得到。

$$avgT_{\text{resp}}(t_k)=\frac{\sum_{i=1}^{n}\sum_{p=1}^{P_i(t_k)}T_{\text{resp}}(t_k,i,p)}{N(t_k)} \tag{11}$$

其中，$N(t_k)$ 表示 t_k 时刻的任务总量。

本文使用适应度函数来评估资源分配方案的有效性。t_k 时刻的适应度函数值 $Fitness$ 可以通过式(12)计算得到。

$$Fitness=\alpha\times avgT_{\text{resp}}(t_k)+\beta\times Cost(t_k) \tag{12}$$

其中，α 和 β 是由云工程师预先定义的权重，分别对应任务平均响应时间和资源成本。

因此，在负载时间窗口 L 内，资源分配方案在 t_k 时刻的适应度函数 $Fitness^L$ 可以通过式(13)计算得到。

$$Fitness^L=\alpha\times\int_{t_k}^{t_k+L}avgT_{\text{resp}}(t)\,\mathrm{d}t+\beta\times\int_{t_k}^{t_k+L}Cost(t)\,\mathrm{d}t \tag{13}$$

在云边协同的无服务器计算环境中，基于负载时间窗口的应用资源分配问题可以定义为：给定系统参数 (E,BS,W) 和时间窗口 L，在 t_k 时刻寻找合适的容器资源分配方案 CT_{t_k}，使得适应度函数最小：

$$\min\ Fitness^L \tag{14}$$

4 方法描述

本章详细介绍了规则引导下基于协同进化算法的资源分配方法 RARCA，方法整体架构如图 3 所示。

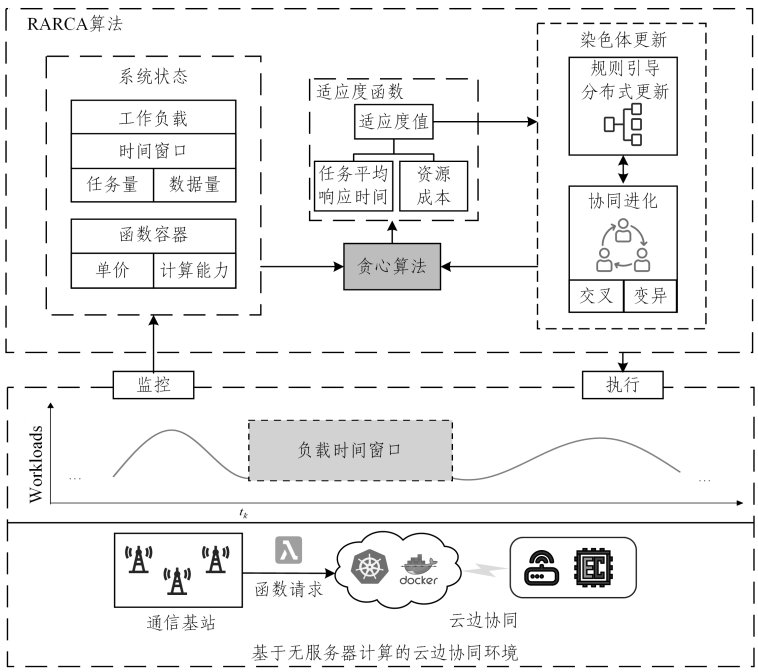


图 3 方法概览图

Fig 3 Overview of the proposed method

该方法引入协同进化机制，通过计算节点的信息共享，协同优化每个计算节点函数容器部署方案的更新方向，以提高算法搜索全局资源分配方案的性能。同时，采用规则引导的分布式更新策略，计算节点根据自身信息调整时间窗口内的函数容器部署方案，能够达到接近实时的决策速度。RARCA 中的染色体对应每个计算节点在时间窗口内的函数容器部署方案的编码，分布式更新策略会根据计算节点的系统状态与规则更新染色体的基因片段；而协同进化机制能够根据全局信息，指导每个计算节点的染色体执行变异操作与交叉操作，从而在保证算法整体有效性的同时，满足系统对算法决策实时性的要求。

4.1 问题编码与初始化

计算节点中每个染色体代表该节点的一个候选容器部署方案，本文采用离散编码方式对染色体进行编码，节点的染色体编码表示当前节点在时间窗口内部署的无服务器函数容器数量。节点 j 的染色体在第 t 次迭代时的编码表示为：

$$X_j^t=\{x_{j1}^t,x_{j2}^t,\cdots,x_{jl}^t\} \tag{15}$$

其中，$l=L/\Delta t$ 表示时间窗口内调整资源分配方案的次数。

边缘节点将自身染色体上传至云计算节点后，云节点将染色体整合为系统整体的无服务器应用资源分配方案。第 t 次迭代云计算节点的系统整体资源分配方案的编码 X^t 如式(16)所示：

$$X^t=\{X_j^t \mid j\in E\} \tag{16}$$

为了在搜索开始时更好地调整染色体基因，避免算法快速陷入局部最优，本文在迭代更新前将随机生成的预定义的解作为初始染色体。初始染色体将满足其所在节点的计算资源限制约束，并确保其可行性。

4.2 适应度函数

适应度值是评价整体资源分配方案有效性的重要指标，对协同进化过程起指导性作用，通常认为适应度值较小的资源分配方案为更优解。在算法迭代过程中，可能出现计算节点的资源不足以支撑所有任务完成的情况，此时的资源分配方案为不可行解。在资源分配方案评估的过程中，若二者都是可行解，则选择适应度值更低的方案，并更新每个计算节点的全局最优染色体；若新的方案为不可行解，则选择保留旧的资源分配方案。由于算法初始化时保证了初始解的可行性，并且算法总是选择更好的资源分配方案，因此不会出现二者都为不可行解的情况。用于比较两个候选染色体质量的适应度函数定义为：

$$F(X^t)=Z(Fitness^L(X^t)) \tag{17}$$

其中，Z 表示染色体选择操作，本文将选取适应度值 $Fitness^L$ 更低的可行染色体。

4.3 染色体更新策略

计算节点经过一定轮次的资源调整，将自身资源分配方案与工作负载情况上传至云节点，云节点计算全局适应度值，并将计算结果分发至边缘节点。各节点根据适应度值更新记录全局最优个体，并执行遗传操作更新染色体。染色体 j 在第 $t+1$ 次迭代时的更新方式如式(18)所示：

$$X_j^{t+1}=c\otimes(w\odot X_j^t,gBest^t) \tag{18}$$

其中，$\otimes$ 和 $\odot$ 分别表示交叉算子和变异算子，$gBest^t$ 表示全局最优染色体；w 表示惯性因子，它决定了算法的收敛能力；c 被称为加速常数，体现了染色体对全局最优值的学习能力。

采用变异算子对计算节点的染色体编码进行更新的操作如式(19)所示：

$$A_j^{t+1}=w\odot X_j^t=\begin{cases}Mu(X_j^t), & r_1<w\\ X_j^t, & \text{otherwise}\end{cases} \tag{19}$$

其中，r_1 是取值于[0,1]区间的随机数，当 $r_1<w$ 时，X_j^t 发生变异操作，$Mu(X_j^t)$ 会随机更改 X_j^t 的一个基因座的资源编码；若 $r_1\geqslant w$，则不发生变异操作。

染色体的变异操作过程如图 4 所示。随机选取染色体的 k 个基因座，每个基因座的编码在资源限制区间内进行随机变异，从而生成新的染色体编码 A_j^{t+1}。

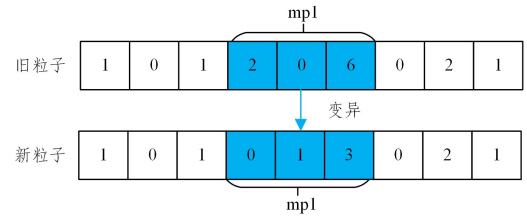


图 4 染色体变异操作

Fig. 4 Mutation operation of chromosomes

采用交叉算子对计算节点的染色体编码进行更新的操作如式(20)所示：

$$X_j^{t+1}=c\otimes A_j^{t+1}=\begin{cases}Co(A_j^{t+1},gBest^t), & r_2<c\\ A_j^{t+1}, & \text{otherwise}\end{cases} \tag{20}$$

其中，r_2 是取值于[0,1]区间的随机数，当 $r_2<c$ 时，A_j^{t+1} 发生交叉操作，$Co(A_j^{t+1},gBest^t)$ 会随机选择 A_j^{t+1} 的一个基因片段与 $gBest^t$ 对应的基因片段进行交叉操作；若 $r_2\geqslant c$，则不发生交叉操作。

染色体的交叉操作过程如图 5 所示。随机选取两个交叉点执行交叉操作，将粒子 A_j^{t+1} 和 $gBest^t$ 在两个交叉点之间的编码位交叉，并生成新的编码粒子 X_j^{t+1}。

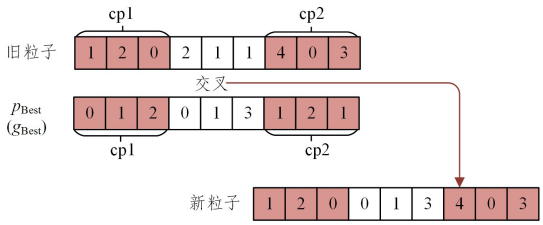


图 5 染色体交叉操作

Fig. 5 Crossover operation of chromosomes

4.4 协同进化机制

为了在时间约束内充分搜索解空间，RARCA 引入了一种多点协同进化机制，对多节点目标进行协同优化。首先，每个计算节点寻找较优的解决方案，根据自身时间窗口和周边节点的工作负载情况，以及任务响应时间，调整自身容器部署方案。然后，为了避免任意计算节点对自身进行极端优化，引入协同优化的思想，对节点的方案进行遗传操作。此过程由云计算节点主导，根据全局优化目标计算适应度值，指导所有计算节点执行遗传操作。

同时，为了在协同进化过程中提升节点自身的搜索能力，引入了规则引导的分布式更新策略，即计算节点在执行遗传操作后，进行一定轮次的迭代更新，鼓励计算节点自行调整容器部署策略，加快整体算法的收敛速度。经过实验测试，在分布式更新策略中使用复杂算法(如粒子群优化算法等)会使计算节点自身的资源方案收敛过快，容易陷入局部最优，从而导致 RARCA 整体的有效性不佳。因此，算法采用了基于规则的策略来调整节点的计算资源。

在制定计算节点的资源策略调整规则时，根据计算节点的相邻节点状态、时间窗口状态、任务平均响应时间 3 个要素更新资源方案。为方便描述，设当前时刻执行分布式资源调整的计算节点的负载量为 W_{cur}，其相邻节点的平均工作负载量为$\overline{W_N}$，其时间窗口内的平均负载量为$\overline{W^L}$，其任务平均响应时间为 RT。具体调整方式如下：

1）计算节点的相邻节点状态。$\overline{W_N}$与 W_{cur}的比值越小，表示当前节点的工作负载量相比其他计算节点越大，因此需要更多的计算资源来支持高负载的任务状态；反之，则需要减少计算资源来降低不必要的开销。

2）计算节点时间窗口内的负载状态。$\overline{W^L}$与 W_{cur}的比值较大，则需要增加计算资源以保证时间窗口内的资源平衡，降低因大幅调整资源数量产生的成本；反之，则需要减少计算资源。

3）任务平均响应时间。RT 直接体现了当前节点的资源情况，RT 较大，则需要增加计算资源；反之，则需要减少计算资源。

4.5 算法流程

RARCA 的具体流程图 6 所示。算法首先将随机生成的预定义的解作为初始资源分配方案，并开始更新迭代过程。每轮迭代将通过规则引导的分布式资源更新机制调整计算节点资源，并在算法达到协同轮次时进行协同进化操作。首先，由云节点整合所有计算节点上传的资源方案，并计算全局适应度值；然后，边缘节点将下载全局适应度值，并执行遗传操作进行资源方案更新；最后，当算法达到最大迭代轮次时，将记录的全局最优方案作为算法的最终资源分配方案进行输出。

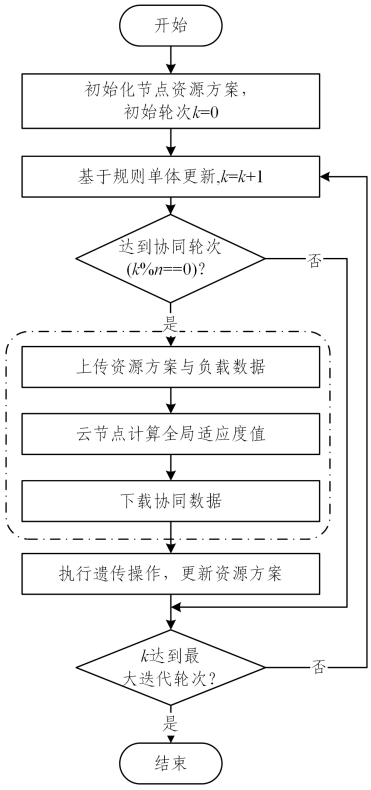


图 6 RARCA 流程图

Fig. 6 Flowchart of RARCA

5 实验结果与分析

本章将对 RARCA 方法进行实验评估。所有的模拟仿真实验都是在配备有 16GB 内存和 Intel i5-9500F 3.0GHz 处理器的 Windows 10 操作系统下进行，本文提出的方法与所有基准方法均在 Python 3.10 环境下实现。

5.1 实验设置

实验评估中所采用的工作负载配置与基础设施分布来自于上海电信基站数据集，该数据集包含上海市的基站位置信息以及互联网访问情况[24]。本文基于不同地区用户对互联网服务的不同使用程度，从该数据集中选取了 5 个具有代表性的区域，并使用区域内的通信数据来模拟仿真实验。区域大小均为 2000 m * 2000 m，各个区域内通信基站与计算节点的分布情况如图 7 所示。5 个区域体现了不同的互联网服务需求程度以及通信基站和计算节点的覆盖率，具体信息如表 1 所列。

(a)区域一 (b)区域二 (c)区域三

(d)区域四 (e)区域五

图 7 5 个上海市代表性区域的计算节点与通信基站分布图

Fig. 7 Distribution maps of computing nodes and base stations in five representative regions of Shanghai

表 1 各区域的分布信息

Table 1 Distribution information in each region

区域编号	计算节点数	通信基站数
区域一	5	9
区域二	7	11
区域三	9	16
区域四	11	21
区域五	13	24

计算节点的容器租赁价格基于亚马逊提供的 aws lambda 以及 aws lambda@edge 的定价标准[25]，其中边缘计算节点的单价与区域基站密集程度成正相关，并在一定范围内浮动。计算节点提供同种规格的无服务器函数容器，其计算频率设置为 4GHz，容器的冷启动时间为 250 ms[26]。单个函数请求的数据量在区间(0，1000]kB 随机取值，计算复杂度在[50，500] MCycles 区间取值。区域中通信基站与边缘计算节点间的数据传输速率可以描述为自由空间路径损耗模型[27]，参数如表 2 所列，模拟实验中的数据传输速率约为 2～10 MB/s。

边缘计算节点的最大容器租用限制为 10。工作负载-时间窗口的长度设为 30 s，每次资源调整的时间间隔为 5 s。适应度函数中的任务平均响应时间权重 α 与资源成本权重 β 分别表示云工程师对 QoS(Quality of Service)和资源成本的不同偏好。例如，α 越大表示对 QoS 的偏好越敏感，而较大的 β 表示对资源成本的偏好更敏感。参考文献[28-29]，为了平衡 QoS 和资源成本，本文设置 α 和 β 分别为 320 和 10。

表 2 自由空间路径损耗模型的参数

Table 2 Parameters of free-space path loss model

参数	值	描述
B/MHz	10	信道带宽
$p_i, \forall i$	0.1	通信基站传输功率
g_0	1.42×10^{-4}	信道增益
Σ/(dBm/Hz)	-90	噪声功率

基于文献[29-30]，RARCA 算法最大迭代次数为 1 000 轮，其中，每 5 轮进行一次协同进化，遗传操作的惯性因子 $w=0.5$，加速常数 $c=0.5$。计算节点的分布式更新策略的资源调整规则的设置如表 3 所列，云节点没有相邻节点，其资源调整规则仅依据基于时间窗口状态与响应时间的调整规则，而边缘节点的资源调整则需要叠加基于时间窗口状态与响应时间的调整规则和基于相邻节点状态与响应时间的调整规则执行两次操作。

表 3 无服务器计算资源调整规则

Table 3 Resource adjustmentrules for serverless computing

	条件一	条件二	操作
基于时间窗口状态与响应时间的调整规则	$\frac{\overline{W^L}}{W_{cur}}>1.1$	$RT>250$ ms	容器数加一
		$RT\leqslant250$ ms	容器数不变
	$0.9<\frac{\overline{W^L}}{W_{cur}}\leqslant1.1$	$RT>250$ ms	容器数加一
		150 ms$<RT\leqslant$250 ms	容器数不变
		$RT\leqslant150$ ms	容器数减一
	$0\leqslant\frac{\overline{W^L}}{W_{cur}}\leqslant0.9$	$RT>250$ ms	容器数不变
		$RT\leqslant250$ ms	容器数减一
基于相邻节点状态与响应时间的调整规则	$0\leqslant\frac{\overline{W_N}}{W_{cur}}\leqslant0.9$	$RT>250$ ms	容器数加一
		$RT\leqslant250$ ms	容器数不变
	$0.9<\frac{\overline{W_N}}{W_{cur}}\leqslant1.1$	$RT>250$ ms	容器数加一
		150 ms$<RT\leqslant$250 ms	容器数不变
		$RT\leqslant150$ ms	容器数减一
	$\frac{\overline{W_N}}{W_{cur}}>1.1$	$RT>250$ ms	容器数不变
		$RT\leqslant250$ ms	容器数减一

5.2 基准方法

为了验证 RARCA 在面向负载时间窗口的场景中求解云边协同环境下无服务器应用资源分配问题的有效性，本文改进了以下 3 种方法，以适用于同样的资源分配场景，并将其作为基准方法，比较 RARCA 和其他基准方法在资源分配问题上的性能表现。

1)RACPG(Resource Allocation Method Based on Co-evolutionary Particle Swarm Optimization Employing Genetic Operators)[31]：该方法采用与 RARCA 相同的资源编码方案，在分布式更新策略中采用传统 PSO(Particle Swarm Optimization)算法对粒子编码进行更新以寻找较优的资源分配方案，在一定轮次的分布式更新后，由云计算节点整合例子编码并评估适应度值，并指导计算节点执行 PSO 算法的粒子更新以及遗传算法的交叉、变异过程。算法的最大迭代次数与协同策略的设置与 RARCA 相同，其他参数根据文献[31]进行设置。

2)RADR(Resource Allocation Method Based on Distributed Rule)[29]：该方法使用与 RARCA 相同的资源编码方案。在资源分配过程中，计算节点根据预定的规则调整容器租赁方案，并将每次分布式更新后的方案上传至云节点进行评估，同时记录迭代过程中的最优解。相比 RARCA，该方法仅有相邻计算节点的工作负载信息，而没有利用全局信息进行协同进化的过程。RADR 的最大迭代次数以及资源调整规则的设置与 RARCA 相同。

3)RADRS(Resource Allocation Method Based on Distributed Random Searching)[32]：该方法使用与 RARCA 相同的资源编码方案，节点分布式更新策略基于随机搜索策略生成新的容器租赁方案，并将生成的方案上传至云计算节点进行评估，同时记录随机搜索过程中的最优解，最终输出全局最优解为最终资源分配方案。RADRS 最大迭代次数的设置与 RARCA 相同。

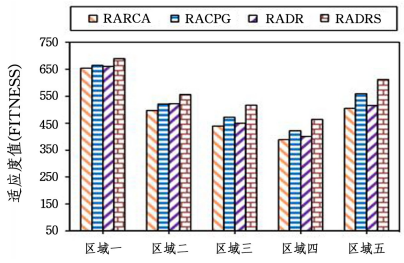
此外，为了加强对方法的有效性评估，本节将 RARCA 与理想方案进行了对比。不同于分布协同式的方法，理想方案由基于集中式的搜索方法进行求解。由于云边协同环境下面向负载时间窗口的无服务器应用资源问题的问题空间太大，遍历搜索全部可行解需要消耗巨大的时间成本和算力成本，在本文的实验环境中难以实现，因此求解理想方案的方法基于集中式 PSO-GA(Particle Swarm Optimization-Genetic Algorithm)算法，并对该方法的粒子编码进行了调整，将每个计算节点中时间窗口内的容器部署方案整合为一个粒子，针对该粒子执行对应的资源分配方案的搜索策略。理想方法实验参数的设置参考文献[31]。

5.3 实验结果与分析

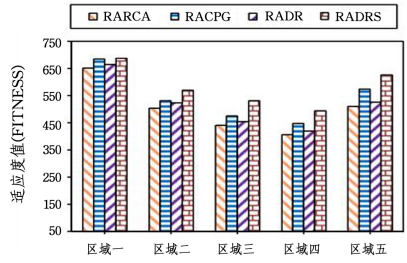
为了比较 RARCA 与其他基准方法在云边协同环境下面向负载时间窗口的无服务器应用资源分配问题中的性能表现，基于上海电信基站的工作负载数据，在不同规模的工作负载情况下，针对上述 5 个上海市实验区域的无服务器应用资源分配分别进行了 3 组实验，包括小规模工作负载、中等规模工作负载和大规模工作负载 3 种工作负载规模，并对每种资源分配方法进行了 10 次独立重复实验，取其平均值作为实验结果。

目标函数方面，图 8 展示了 4 种面向负载时间窗口的资源分配方法在不同工作负载规模下的适应度值。整体来看，本文提出的资源分配方法 RARCA 在各种场景下都得到了最优的资源分配方案，即资源分配方案的适应度值最小。RARCA 的协同进化机制提高了全局搜索目标资源分配方案的能力，基于规则的分布式更新策略为算法提供了一定的局部搜索能力，使其能够在全局搜索目标的一定范围内进一步深入搜索，最终得到最优资源分配方案。RACPG 在该优化问题中的表现较弱，虽然拥有相同的协同机制，但其分布式更新策略使用了基于传统 PSO 的粒子更新策略，每次更新对粒子的修改幅度较大，局部搜索过程收敛过快反而对算法的整体性能产生了负面影响。RADR 的整体表现仅次于 RARCA，该方法只关注对计算节点自身的资源分配方案的探索，规则驱动的搜索策略能够更细致地搜索问题的解空间，拥有较强的局部搜索能力，但缺乏有效的手段来避免算法陷入局部最优。

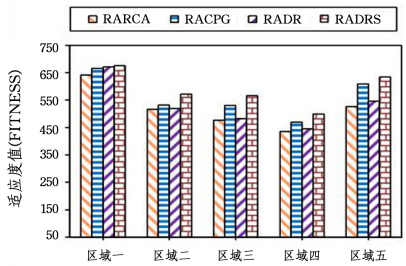
而 RADRS 所采用的随机搜索策略效率较低，其在解空间内进行无差别的探索，因此在规定的资源调整时间间隔内难以搜索到具备高有效性的资源分配方案。



(a)小规模工作负载



(b)中等规模工作负载



(c)大规模工作负载

图 8 不同资源分配方法的适应度值比较

Fig. 8 Comparison of fitness values for different resource allocation methods

具体评价指标方面，表 4 列出了 4 种资源分配方法在 5 个实验区域下对 3 种工作负载规模求平均后的任务平均响应时间(单位：s)与资源成本(单位：1×10^{-3} \$)，以评估算法在不同评价指标上的具体表现，每个区域内的最优指标用粗体标出。RARCA 在 5 个区域中表现出了最短的任务平均响应时间和相对较低的资源成本，其资源成本虽然在某些区域低于 RADR，但是差距并不大。这表明，RARCA 在用户服务质量保障和资源成本控制方面具有更好的平衡性，能够适应工作负载对资源需求的变化，并有效控制容器开关所带来的额外成本。RACPG 能够保持相对合理的响应时间，但在资源成本控制上不够理想。这是因为，相比基于规则引导的分布式更新策略，基于 PSO 的分布式更新策略无法有效地对时间窗口内的动态资源调整做出合理的判断，从而产生资源浪费。RADR 在任务平均响应时间的保证和资源成本控制上的表现仅次于 RARCA。这是因为，基于规则的资源调整策略能够根据时间窗口、相邻计算节点的工作负载情况和当前任务平均响应时间做出合理的资源调整操作，但其依赖于本地信息进行决策，在整体表现上不如基于全局信息协同的 RARCA 有效。RADRS 在所有方法中产生了最高的资源成本，过量的资源分配使得任务平均响应时间较长。RADRS 虽然在理论上能够探索广泛的解空间，但是在资源分配时间间隔为 5 s 的情况下，无法找到问题的全局最优解，所得解往往导致较大的资源浪费，在所有方法中的整体性能表现最差。

表 4 不同资源分配方法的任务平均响应时间与资源成本

Table 4 Average response time and cost of different resource allocation methods

区域编号	资源分配方法	任务平均响应时间	资源成本
区域一	RARCA	**0.32935**	**1.71408**
	RACPG	0.33335	3.95166
	RADR	0.34650	1.92794
	RADRS	0.33406	4.71358
区域二	RARCA	**0.24215**	**4.03052**
	RACPG	0.24531	5.74047
	RADR	0.24770	4.64063
	RADRS	0.25785	6.81164
区域三	RARCA	**0.21609**	4.03899
	RACPG	0.22263	7.50497
	RADR	0.22612	**3.67683**
	RADRS	0.23581	8.44932
区域四	RARCA	**0.18410**	5.61427
	RACPG	0.18427	9.27872
	RADR	0.19052	**5.48667**
	RADRS	0.20276	10.14248
区域五	RARCA	**0.24187**	5.06475
	RACPG	0.24515	10.88007
	RADR	0.25310	**4.67413**
	RADRS	0.26217	12.00758

为了验证 RARAC 在各场景下的整体性能，对不同资源分配方法在 5 个场景、3 种不同工作负载规模的 10 次独立重复实验结果取平均值。4 种资源分配方法的平均适应度值如图 9 所示，RARAC 相比其他 4 种资源分配方法得到了最优的适应度均值，其平均搜索性能相比 RACPG，RADR，RADRS 分别提升了约 7.5%，2.8%，14.5%。

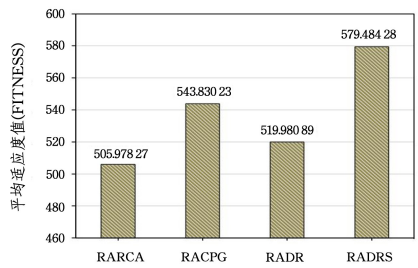


图 9 不同资源分配方法的平均适应度值

Fig. 9 Average fitness values for different resource allocation methods

此外，本节将 RARCA 与理想方案进行了对比。在理想方案的求解过程中，假设全局所有计算节点的信息可以共享，并且没有资源调整时间间隔对算法决策时间的限制，以此得到理论最优解。如图 10 所示，RARCA 在 5 个实验区域对 3 种不同工作负载规模的适应度值进行平均，其整体表现平均低于理想方案 1.87%。其中，在区域三内，RARCA 可以实现与理想方案相似的资源分配方案，二者之间的表现差异小于 1%，这表明所提算法可以获得接近最优的性能表现。

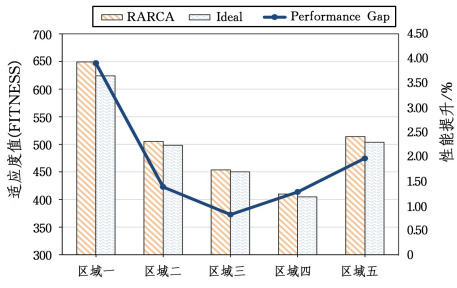


图 10 RARCA 与理想方案的适应度值比较

Fig. 10 Comparison of fitness values between RARCA and ideal

另外，在实际应用场景中，由于云边协同环境下面向负载时间窗口的无服务器应用资源分配问题的复杂度较高，解空间太大，因此很难达到理想资源分配方案的效果。RARCA 与理想方案的算法决策时间如表 5 所列，理想方案的决策时间随着场景复杂度的增加不断增长，对于有决策及时性要求的面向负载时间窗口的无服务器应用资源分配场景，这样长的决策时间是不可接受的。而 RARCA 得益于其分布协同式的特征，计算节点能够在分布式调整资源的同时，通过协同进化机制引导算法朝全局优化目标探索，其算法决策时间不受场景复杂度的影响，平均稳定在 3～4 s 左右，能够适用于具有及时性要求的资源分配场景。

表 5 RARCA 与理想方案的算法决策时间比较

Table 5 Comparison of decision time between RARCA and ideal

(s)

	区域一	区域二	区域三	区域四	区域五
RARCA	2.75	3.11	4.00	3.16	4.01
Ideal	401.95	954.46	1350.45	1482.40	1760.38

最后，为了验证 RARCA 求解最优资源分配方案所需的迭代次数，对 RARCA 在中等规模工作负载下 5 个实验区域内的迭代过程进行分析。如图 11 所示，在所有实验区域中，RARCA 都能在 100 轮迭代内快速收敛，并在 500 轮迭代左右获得最终的资源分配方案，这表明 RARCA 拥有良好的收敛性能。

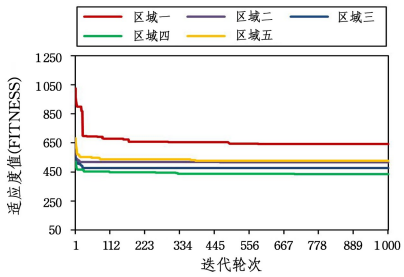


图 11 RARCA 迭代过程中适应度值的变化

Fig. 11 Changes of fitness values during RARCA iteration process

综上所述，本文提出的 RARCA 在云边协同环境下面向负载时间窗口的无服务器应用资源分配问题中，相比于其他基准方法，能够得到最佳的资源分配方案，有效平衡任务平均响应时间和资源成本两项评价指标。此外，RARCA 能够在秒级的决策时间内得到接近理想方案的资源分配结果。因此，本文提出的云边协同环境下面向负载时间窗口的无服务器应用资源分配方法在任务平均响应时间的缩短和资源成本的控制上有着良好的优化和平衡效果，同时能够保证资源分配决策的及时性。

结束语 本文提出了 RARCA 算法来探索云边协同环境下面向负载时间窗口的无服务器应用资源自适应分配问题。首先，设计了基于规则的计算节点分布式资源调整模型，能够在不同的系统状态下执行适当的管理操作。其次，引入协同进化机制，通过信息共享找到当前系统状态下全局最优的资源分配方案。基于上海电信基站的真实数据集，进行了大量的模拟仿真实验，验证了 RARCA 算法解决无服务器应用资源分配问题的有效性。结果表明，RARCA 算法相比其他经典算法，性能提高了 2.8%～14.5%，且 RARCA 可以获得接近理想方案的最优性能。在未来的工作中，将考虑更复杂的问题场景，如具有不同工作流的无服务器计算应用；考虑引入基于深度强化学习的方法[29]来进一步提高决策速度和准确性，并设计能够适用于不同场景的通用模型；在更多类型、更大规模的数据集上评估算法的性能。

参 考 文 献

[1] ALWARAFY A, AL-THELAYA K A, ABDALLAH M, et al. A survey on security and privacy issues in edge-computing-assisted internet of things[J]. IEEE Internet of Things Journal, 2020, 8(6): 4004-4022.

[2] RAZA M R, VAROL A, VAROL N. Cloud and fog computing: A survey to the concept and challenges[C]// 2020 8th International Symposium on Digital Forensics and Security (ISDFS). 2020: 1-6.

[3] HAO Y, JIANG Y, CHEN T, et al. iTaskOffloading: Intelligent task offloading for a cloud-edge collaborative system[J]. IEEE Network, 2019, 33(5): 82-88.

[4] CHADWICK D W, FAN W, COSTANTINO G, et al. A cloud-edge based data security architecture for sharing and analysing cyber threat information[J]. Future Generation Computer Systems, 2020, 102: 710-722.

[5] HABIBI P, FARHOUDI M, KAZEMIAN S, et al. Fog computing: a comprehensive architectural survey[J]. IEEE Access, 2020, 8: 69105-69133.

[6] CHEN X, ZHENG S. Resource allocation and task offloading strategy base on hybrid simulated annealing-binary particle swarm optimization in cloud-edge collaborative system[C]// 2022 IEEE 5th Advanced Information Management, Communicates, Electronic and Automation Control Conference. 2022: 379-383.

[7] VAN EYK E, GROHMANN J, EISMANN S, et al. The spec-rg reference architecture for faas: from microservices and containers to serverless platforms[J]. IEEE Internet Computing, 2019, 23(6): 7-18.

[8] EIVY A, WEINMAN J. Be wary of the economics of "serverless" cloud computing[J]. IEEE Cloud Computing, 2017, 4(2): 6-12.

[9] WU M, MI Z, XIA Y. A survey on serverless computing and its implications for jointcloud computing[C]// 2020 IEEE International Conference on Joint Cloud Computing. 2020: 94-101.

[10] CHEN Z,HU J,MIN G,et al. Towards accurate prediction for high-dimensional and highly-variable cloud workloads with deep learning[J]. IEEE Transactions on Parallel and Distributed Systems,2019,31(4):923-934.

[11] KIM I K,WANG W,QI Y,et al. Forecasting cloud application workloads with cloudinsight for predictive resource management [J]. IEEE Transactions on Cloud Computing,2020,10(3):1848-1863.

[12] YANG L J,CHEN X,HUANG Y H. A PSO-GA-based adaptive resource allocation method for cloud software services oriented to load-time window[J]. Journal of Chinese Computer Systems,2021,42(5):953-960.

[13] ZHOU H,WU T,CHEN X,et al. Reverse auction-based computation offloading and resource allocation in mobile cloud-edge computing[J]. IEEE Transactions on Mobile Computing,2022,22(10):6144-6159.

[14] YUAN H,ZHOU M C. Profit-maximized collaborative computation offloading and resource allocation in distributed cloud and edge computing systems[J]. IEEE Transactions on Automation Science and Engineering,2020,18(3):1277-1287.

[15] GAN D,GE X,LI Q. An optimal transport-based federated reinforcement learning approach for resource allocation in cloud-edge collaborative IoT[J]. IEEE Internet of Things Journal,2023,11(2):2407-2419.

[16] LI Y,LIN Y,WANG Y,et al. Serverless computing:state-of-the-art,challenges and opportunities[J]. IEEE Transactions on Services Computing,2022,16(2):1522-1539.

[17] JARACHANTHAN J,CHEN L,XU F,et al. Astrea:Auto-serverless analytics towards cost-efficiency and QoS-awareness [J]. IEEE Transactions on Parallel and Distributed Systems,2022,33(12):3833-3849.

[18] ASCIGIL O,TASIOPOULOS A G,PHAN T K,et al. Resource provisioning and allocation in function-as-a-service edge-clouds [J]. IEEE Transactions on Services Computing,2021,15(4):2410-2424.

[19] KIM Y K,HOSEINYFARAHABADY M R,LEE Y C,et al. Automated fine-grained cpu cap control in serverless computing platform[J]. IEEE Transactions on Parallel and Distributed Systems,2020,31(10):2289-2301.

[20] RAZA A,AKHTAR N,ISAHAGIAN V,et al. Configuration and placement of serverless applications using statistical learning[J]. IEEE Transactions on Network and Service Management,2023,20(2):1065-1077.

[21] 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.

[22] CHEN Z,YANG L,HUANG Y,et al. Pso-ga-based resource allocation strategy for cloud-based software services with workload-time windows[J]. IEEE Access,2020,8:151500-151510.

[23] MESTRE J. Greedy in approximation algorithms[C]// European Symposium on Algorithms. 2006:528-539.

[24] WANG S,ZHAO Y,HUANG L,et al. QoS prediction for service recommendations in mobile edge computing[J]. Journal of Parallel and Distributed Computing,2019,127:134-144.

[25] XIE R,GU D,TANG Q,et al. Workflow scheduling in serverless edge computing for the industrial internet of things:a learning approach[J]. IEEE Transactions on Industrial Informatics,2023,19(7):8242-8252.

[26] AKHTAR N,RAZA A,ISHAKIAN V,et al. COSE:Configuring serverless functions using statistical learning[C]// IEEE INFOCOM 2020-IEEE Conference on Computer Communications,2020:129-138.

[27] HU Q,CAI Y,YU G,et al. Joint offloading and trajectory design for UAV-enabled mobile edge computing systems[J]. IEEE Internet of Things Journal,2018,6(2):1879-1892.

[28] CHEN X,WANG H,MA Y,et al. Self-adaptive resource allocation for cloud-based software services based on iterative QoS prediction model[J]. Future Generation Computer Systems,2020,105:287-296.

[29] CHEN X,YANG L,CHEN Z,et al. Resource allocation with workload-time windows for cloud-based software services:a deep reinforcement learning approach[J]. IEEE Transactions on Cloud Computing,2023,11(2):1871-1885.

[30] LI H,WANG D,ZHOU M C,et al. Multi-swarm co-evolution based hybrid intelligent optimization for bi-objective multi-workflow scheduling in the cloud[J]. IEEE Transactions on Parallel and Distributed Systems,2021,33(9):2183-2197.

[31] SHI Y,EBERHART R. A modified particle swarm optimize [C]// 1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence(Cat. No. 98TH8360),1998:69-73.

[32] ZHOU B,XIE S,WANG F,et al. Multi-step predictive compensated intelligent control for aero-engine wireless networked system with random scheduling[J]. Journal of the Franklin Institute,2020,357(10):6154-6174.

ZHANG Minghao, born in 1998, postgraduate. His main research interests include cloud computing, resource allocation and serverless computing.

CHEN Xing, born in 1985, Ph.D, professor, Ph.D supervisor, is a distinguished member of CCF (No. 35725M). His main research interests include software engineering, systems software and cloud computing.

(责任编辑:何杨)