计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.201100009

基于改进多目标进化算法的微服务用户请求分配策略

朱汉卿　马武彬　周浩浩　吴亚辉　黄宏斌
国防科技大学信息系统工程重点实验室　长沙 410073
(zhuhanqing258@foxmail.com)

摘　要　如何对基于微服务架构的系统进行并发用户请求的分配以使得时间、成本和均衡性等目标得到优化,是面向微服务的应用系统需关注的重要问题之一。现有的基于固定规则的用户请求分配策略仅着重于负载均衡性的解决,难以处理多目标需求间的平衡。为此,文中提出以请求处理总时间、负载均衡率和通信传输总距离为多个目标的微服务用户请求分配模型,研究并发用户请求在部署于不同资源中心的多个微服务实例间的分配策略,并使用基于改进初始解生成策略、交叉算子和变异算子的多目标进化算法对该问题进行求解。在不同规模的数据集上进行多次实验,结果表明,提出的方法与常用的多目标进化算法和传统的基于固定规则的方法相比,能够更好地处理多个目标间的平衡,具有更好的求解性能。

关键词:微服务;请求分配;多目标优化;进化算法;并发请求
中图法分类号　TP311.1

Microservices User Requests Allocation Strategy Based on Improved Multi-objective Evolutionary Algorithms

ZHU Han-qing,MA Wu-bin,ZHOU Hao-hao,WU Ya-hui and HUANG Hong-bin
Key Laboratory of Information System Engineering,National University of Defense Technology,Changsha 410073,China

Abstract　How to allocate concurrent user requests to a system based on a microservices architecture to optimize objectives such as time,cost,and load balance,is one of the important issues that microservices-based application systems need to pay attention to. The existing user requests allocation strategy based on fixed rules only focuses on the solving of load balance,and it is difficult to deal with the balance between multi-objective requirements. A microservices user requests allocation model with multiple objectives of total requests processing time,load balancing rate,and total communication transmission distance is proposed to study the allocation of user requests among multiple microservices instances deployed in different resource centers. The multi-objective evolutionary algorithms with improved initial solutions generation strategy,crossover operator and mutation operator are used to solve this problem. Through many experiments on data sets of different scales,it is shown that the proposed method can better handle the balance between multiple objectives and has better solving performance,compared with the commonly used multi-objective evolutionary algorithms and traditional methods based on fixed rules.

Keywords　Microservices,Requests allocation,Multi-objective optimization,Evolutionary algorithm,Concurrent requests

1　引言

随着互联网、云计算和大数据等技术的发展,传统的单体架构的应用日渐式微。为此,微服务架构应用而生,它解决了单体应用随着功能增多而逐渐臃肿,从而造成扩展和部署等过程困难的问题[1]。微服务就是将传统的单体应用按照一定原则拆分成一个个逻辑独立的服务,并通过服务间的相互调用来实现业务功能,每个服务都是一个或者一组紧耦合的功能单元的集合。但与传统的 SOA 架构不同,微服务架构下的每个服务都可以独立进行开发、测试和部署等过程,彼此之间完全独立,通常是通过基于 HTTP 协议的 RESTful 形式的接口进行通信[2]。微服务具有自治性、易扩展性、技术异质性、专业性和容错性等特点[3]。

面对移动互联时代带来的具有不确定性的海量并发用户请求,传统的单体架构应用若以峰值满足为标准,势必会在错峰期造成资源的严重浪费。微服务以其自治、易扩展的特点,可以通过灵活地启动或停止部署在不同资源中心的多个微服务实例,来满足不同时段处理用户请求的需求。

对于多目标微服务用户请求分配问题,首先需要进行多个目标的选择和确立,这主要涉及到微服务应用的性能评价。Leitner 等[4]建立了一个微服务应用的开销模型(CostHat);Barna 等[5]提出了针对多层数据密集型应用的性能评价模

到稿日期:2020-11-02　返修日期:2021-03-01　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:中国博士后科学基金(2019M664033)
This work was supported by the Chinese Postdoctoral Science Foundation (2019M664033).
通信作者:黄宏斌(hbhuang@nudt.edu.cn)

型，并构建了一个自主管理系统，根据上述模型得到的评价指标对系统行为进行调节；Ren 等[6]建立了基于微服务架构的应用的性能评价模型；Ma 等[7]以存储和计算利用率、负载均衡率等多个目标，对微服务在不同资源中心的部署和组合策略进行了研究，并使用多目标进化算法对问题进行求解。其次，微服务用户请求的形式和构成也是需要关注的问题之一。Ma 等[3]提出了一个 GMAT 模型，通过服务依赖图(Service Dependency Graph，SDG)来描述应用中众多微服务间的依赖关系，并对此进行可视化展示，以此分析可能存在潜在错误的服务调用链(Service Invocation Chain，SIC)，从而帮助开发者进行错误的快速定位；Zhou 等[8]在分析已有的开源微服务系统的基础上，提出并开发了一个基于微服务架构的 Benchmark 系统。

而对于微服务背景下的分配问题，现有研究主要集中在 3 个方面：1)微服务在集群环境下的组合部署问题；2)微服务任务和工作流的调度问题；3)针对不同目标的用户请求分配策略问题。

对于微服务在集群环境下的组合部署问题，Fu 等[9]介绍了微服务的不同部署方式，并详细阐述了使用容器部署微服务的一般流程；Xu 等[10]提出了一种基于 MAPE 环路的自适应部署优化方法，根据微服务调用链对集群资源固定情况下的微服务实例进行调节，并实现了一个面向微服务系统的运行时部署优化工具；Xia 等[11]考虑了以 docker 为载体的微服务应用在不同的主机上进行部署的策略问题，力求实现资源占用和通信时延的最优化。

对于微服务任务和工作流的调度问题，Fazio 等[12]主要提出了如何针对微服务进行弹性调度和实时调度这一问题，认为可以从用户请求的预测和性能指标的评价等方向进一步考虑；Ion-Dorinel 等[13]设计和实现了一种混合云调度模型，该模型主要用于将任务(Job)分配到相应的处理实体(PE)上；Bao 等[14]提出了基于微服务的应用程序工作流调度问题，优化目标为公共云中用户指定的预算约束下的最小端到端延迟，即考虑如何将微服务应用的不同功能映射到不同主机的不同容器上；Tang 等[15]通过消息中间件机制，设计了一种基于微服务架构的任务调度系统，该系统主要用于处理定时任务和具有依赖关系(通常是顺序关系)的任务间的调度。

对于针对不同目标的用户请求分配策略问题，除通过一些负载均衡(如客户端负载均衡和服务端负载均衡)的工具，以随机、轮询、加权轮询、地址哈希、最小链接数等负载均衡算法，将用户请求均衡分配到不同服务器的多个微服务实例上外，Yang 等[16]讨论了在移动云系统中，为方便对用户提供服务，服务在云端的部署问题和用户请求的时延问题；Zheng 等[17]从云基础设施提供商的角度，以优化微云提供商的成本为目标，提出了请求分发和容器部署在基于容器的微云环境下的解决方案；Xu 等[18]为解决微服务架构下用户请求的响应时延问题(包括网络时延、处理时延、排队时延)，提出了一种基于微服务架构的动态优先级分级调度算法(DPHS)，用于分配用户请求；Tao 等[19]则主要分析了数据中心内部、多数据中心之间和在边缘的小型数据中心等 3 种不同数据中心场景下的流量调度与请求分配问题。

微服务用户请求大多包含了对多个微服务的调用，用户请求的分配即将用户请求分配到不同资源中心的多个微服务实例中。从某种程度上，微服务用户请求分配问题可认为是柔性作业车间调度问题的扩展。柔性作业车间调度问题[20]可以简单描述为多个具有固定加工工序的工件在多台加工机器上的分配问题，以使得最大完工时间最短，其中每道工序可以在多台机器上进行加工，每台机器同一时间只可加工一个工件。用户请求就相当于工件，微服务实例就相当于加工机器。与之不同的是，微服务实例可以并发处理多个用户请求，且请求到达不同资源中心的微服务实例的时间存在差异。

因此，本文提出了一种多目标微服务用户请求分配模型，用于将用户请求分配到不同资源中心的多个微服务实例上，以使得所有用户请求的总处理时间、负载均衡率和通信传输总距离等 3 个目标达到一种最优的平衡。此外，本文提出了一种新的初始解生成策略以及改进的交叉算子和变异算子来对多目标进化算法进行改进，并用其对多目标微服务用户请求分配问题模型进行了求解，得到了该问题经过优化后的满意解。实验结果证明，本文提出的改进算法具有良好的求解性能。

2 问题模型

在对多目标微服务用户请求分配策略进行建模之前，需要明确以下定义。

定义 1(微服务集合 MS) 某单体应用经过逻辑拆分后形成的微服务组成的集合，$MS=\{MS_1, MS_2, \cdots, MS_{MSN}\}$，$MSN$ 为集合中微服务的数量。其中，每一个微服务 $MS_i(i=1,\cdots,MSN)$ 都有其特定的微服务处理时间 BPT_i(business processing time)，即每个微服务完成自身业务流程处理所需的时间；同时，每个微服务 MS_i 可以并发处理的用户请求数量 urq_i(user requests quantity)为整数。

定义 2(微服务依赖图 SDG(Service Dependency Graph)) 对于一个复杂的微服务架构的应用，某些功能的实现可能需要多个微服务之间的相互调用，这就使得微服务之间形成了依赖关系。微服务依赖图正以有向图的形式描述了上述依赖关系，图中的顶点集合即微服务集合，每个顶点对应一个微服务，图中的弧集合即依赖关系的集合，每一条弧对应着微服务间的一个调用依赖关系。图 1 给出了一个微服务依赖图的实例。

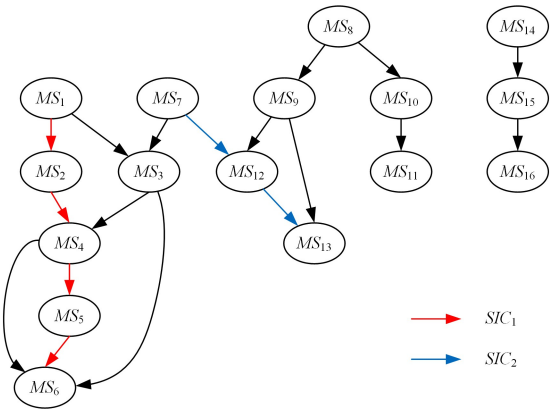

图 1 微服务依赖图示例(电子版为彩色)

Fig. 1 Example of SDG

定义 3(微服务调用链 SIC(Service Invocation Chain))

对于微服务架构应用提供给用户的一个外部 api 接口,该 api 接口可能需要多个微服务间的相互调用才能最终提供给用户反馈结果。微服务调用链正描述了每个用户请求对应的微服务间的调用依赖关系。如图 1 所示,$SIC_1(MS_1,MS_2,MS_4,MS_5,MS_6)$ 和 $SIC_2(MS_7,MS_{12},MS_{13})$ 即对应于两个不同用户请求的两条微服务调用链。

定义 4(用户请求集合 U) $U=\{U_1,U_2,\cdots,U_{UN}\}$,$UN$ 为用户请求的数量。其中,每个用户请求 $U_i(i=1,\cdots,UN)$ 均包含两类信息,分别是位置信息和请求信息。位置信息即发送请求的用户的地理坐标,请求信息即一条微服务调用链。即 $U_i=(LOC_i(x_i,y_i),SIC_i(MS_k,\cdots,MS_q))$,其中,$LOC_i(x_i,y_i)$即用户请求的地理位置信息,$x_i$ 和 y_i 分别代表经度和纬度。

定义 5(资源中心集合 R) $R=\{R_1,R_2,\cdots,R_{RN}\}$,$RN$ 为资源中心的数量。其中,每个 $R_i(i=1,\cdots,RN)$代表一个资源中心,资源中心可以为微服务实例的启动和运行提供所需的环境,资源中心通常包含微服务集合中的所有微服务,并可以根据需要灵活地启动每个微服务相应数量的微服务实例。每个资源中心 R_i 均包含两类信息,分别是位置信息和微服务实例启动信息。位置信息即资源中心所处的地理坐标,微服务实例启动信息即对于微服务集合中的每个微服务,启动微服务实例的数量。即 $R_i=(LOC_i(x_i,y_i),RMSN_i(RMSN_1,\cdots,RMSN_{MSN}))$,其中 $RMSN_j(j=1,\cdots,MSN)$为资源中心 R_i 启动的第 j 个微服务 MS_j 的实例数量。

定义 6(微服务调用时延 MIT(Microservice Invocation Time)) 分布在不同资源中心的微服务实例相互调用产生的网络传输时间。对于分布在同一资源中心的微服务实例间的相互调用,通常忽略该时延,且两个资源中心距离越远,认为该时延越大。可用矩阵的形式表示 RN 个资源中心间的微服务相互调用的时延,如图 2 所示。

$$\begin{array}{c} \\ R_1 \\ \vdots \\ R_{RN} \end{array}\begin{array}{c} \begin{array}{ccc} R_1 & \cdots & R_{RN} \end{array} \\ \begin{pmatrix} 0 & \cdots & MIT_{0,RN} \\ \vdots & 0 & \vdots \\ MIT_{RN,0} & \cdots & 0 \end{pmatrix} \end{array}$$

图 2 微服务调用时延矩阵

Fig. 2 MIT matrix

定义 7(用户请求传输时间 RTT(Request Transmission Time)) 用户请求首次到达资源中心的时间,用户距离资源中心越近,该时间越短。第 i 个用户请求的请求传输时间可用式(1)来表示:

$$RTT_i(x)=distance(U_i.LOC_i,R_x.LOC_x)*T_0(i=1,\cdots,UN,x=1,\cdots,RN) \tag{1}$$

其中,$distance(U_i.LOC_i,R_x.LOC_x)$表示用户请求 U_i 与到达的第 1 个资源中心 R_x 的直线距离;T_0 为一个固定的常数,以正比例函数的理想模式表示传输时间随距离的变化关系。

定义 8(用户请求完成时间 RFT(Request Finish Time)) 用户请求从发送、被处理完成到得到反馈结果的时间,包括微服务处理时间 BPT、微服务调用时延 MIT 和用户请求传输时间 RTT。第 i 个用户请求的完成时间可用式(2)表示:

$$RFT_i=\sum_{j=1}^{SIC_i.length}BPT_{SIC_i(j)}+RTT_i(x)+\sum_{j=2}^{SIC_i.length}MIT_{R(SIC_i(j-1)),R(SIC_i(j))}+\sum_{j=1}^{SIC_i.length}wait(SIC_i(j)) \tag{2}$$

其中,$SIC_i.length$ 表示用户请求 U_i 所包含的微服务调用链 SIC_i 中的微服务的数量,$SIC_i(j)$表示用户请求 U_i 所包含的微服务调用链 SIC_i 中的第 j 个微服务的编号,$R(SIC_i(j))$表示响应上述微服务调用的微服务实例所在的资源中心的编号($R(SIC_i(j-1))$同理),$\sum_{j=1}^{SIC_i.length}wait(SIC_i(j))$ 表示因应用处理能力有限而导致的上述微服务请求必须等待的时间。

定义 9(请求反馈总时间 $TRFT$(Total Requests Feedback Time)) 微服务应用处理完所有用户请求的总时间,因用户请求可以并发访问,且应用处理能力有限,所以最后一个被完成的用户请求的时间即为请求反馈总时间,如式(3)所示:

$$TRFT=\max_{i=1}^{UN}RFT_i \tag{3}$$

定义 10(用户请求分配策略 $Strategy(U,R)$) 对于给定的资源中心集合 R 和用户请求集合 U,用户请求在多个资源中心的微服务实例上的分配策略可以表示为:

$$\begin{aligned}Strategy(U,R)=\{&R_1.RMS_1(MS_1(U_{k1}.SIC_{k1}.MS_1,\cdots,U_{kn}.SIC_{kn}.MS_1),\cdots,MS_{MSN}(U_{p1}.SIC_{p1}.MS_{MSN},\cdots,U_{pm}.SIC_{pm}.MS_{MSN})),R_2.RMS_2(MS_1(U_{q1}.SIC_{q1}.MS_1,\cdots,U_{ql}.SIC_{ql}.MS_1),\cdots,MS_{MSN}(U_{r1}.SIC_{r1}.MS_{MSN},\cdots,U_{rx}.SIC_{rx}.MS_{MSN})),\cdots,R_{RN}.RMS_{RN}(MS_1(U_{s1}.SIC_{s1}.MS_1,\cdots,U_{sy}.SIC_{sy}.MS_1),\cdots,MS_{MSN}(U_{t1}.SIC_{t1}.MS_{MSN},\cdots,U_{tz}.SIC_{tz}.MS_{MSN}))\}\end{aligned}$$

其中,$k1,kn,p1,pm,q1,ql,r1,rx,s1,sy,t1,tz$ 均为用户请求集合中某一请求的编号;$R_1.RMS_1(MS_1(U_{k1}.SIC_{k1}.MS_1,\cdots,U_{kn}.SIC_{kn}.MS_1))$即表示分配到资源中心 R_1 的微服务 MS_1 实例上的用户请求 $U_{k1},\cdots,U_{kn}$的集合,其他同理。

定义 11(负载均衡率 LBR(Load Balancing Rate)) 分配到每个资源中心的总用户数量与该资源中心总处理能力(各微服务处理能力的和)的比值的方差,如式(4)所示:

$$LBR=variance\left(\frac{\sum_{j=1}^{MSN}R_i.RMS_i.MS_j.length}{\sum_{j=1}^{MSN}R_i.RMS_i.RMSN_j*urq_j}\right)\quad i=1,\cdots,RN \tag{4}$$

定义 12(通信传输总距离 TD(Total Distance)) 用户请求在各个资源中心通信时传输的总距离,与现实中的通信成本相对应,距离越长,通信成本越高,反之亦然。具体计算式如下:

$$TD=\sum_{i=1}^{UN}\sum_{j=1}^{SIC_i.length}distance(R(SIC_i\cdot MS_j),R(SIC_i\cdot MS_{j-1})) \tag{5}$$

其中,$R(SIC_i.MS_j)$表示用户请求 U_i 的微服务调用链 SIC_i 中第 j 个微服务 MS_j 所分配到的资源中心,$distance(\cdot)$表示两个资源中心间的物理距离。

因此，多目标微服务用户请求分配策略问题可以描述为：在拥有存在依赖关系 SDG 的微服务集合 MS 的微服务架构应用系统中，对于给定的用户请求集合 U 和资源中心集合 R，寻找用户请求在不同资源中心的多个微服务实例上的分配策略 $Strategy_{best}(U,R)$，以使得请求反馈总时间 $TRFT(Strategy_{best}(U,R))$、负载均衡率 $LBR(Strategy_{best}(U,R))$ 和通信传输总距离 $TD(Strategy_{best}(U,R))$ 整体达到最优。

该问题需要明确以下 3 点假设：1）用户请求集合 U 中的所有请求均在 0 时刻发出；2）若请求某一资源中心的某微服务实例的请求数量超过了微服务实例所能并发处理的最大请求数，那么剩余的请求必须进行等待，直到某些用户请求被此微服务实例处理完成；3）用户请求的微服务调用链 SIC 中的每一个微服务，都只能被某一资源中心的相应的某一个微服务实例所处理。

该问题主要具备以下 3 点约束：1）在每个时刻 t，每个微服务实例可服务的用户请求数量有限；2）用户请求集合中所调用的全部微服务至少由一个资源中心对应的微服务的一个实例提供；3）每个用户请求对微服务的调用均严格按照微服务调用链中微服务的先后顺序。

综上所述，多目标微服务用户请求分配策略的数学模型可表示为：

对于解空间决策变量 $x\in\Omega$，$x=\{R_1.RMS_1(MS_1(U_{k1}.SIC_{k1}.MS_1,\cdots,U_{kn}.SIC_{kn}.MS_1),\cdots,MS_{MSN}(U_{p1}.SIC_{p1}.MS_{MSN},\cdots,U_{pm}.SIC_{pm}.MS_{MSN})),R_2.RMS_2(MS_1(U_{q1}.SIC_{q1}.MS_1,\cdots,U_{ql}.SIC_{ql}.MS_1),\cdots,MS_{MSN}(U_{r1}.SIC_{r1}.MS_{MSN},\cdots,U_{rx}.SIC_{rx}.MS_{MSN})),\cdots,R_{RN}.RMS_{RN}(MS_1(U_{s1}.SIC_{s1}.MS_1,\cdots,U_{sy}.SIC_{sy}.MS_1),\cdots,MS_{MSN}(U_{t1}.SIC_{t1}.MS_{MSN},\cdots,U_{tz}.SIC_{tz}.MS_{MSN}))\}$

$$\begin{cases}\min\limits_{x\in\Omega} F(x)=\min(TRFT(x),LBR(x),TD(x))\\ \text{s.t. } c_1(x):x.R_i.RMS_i.MS_j.length(t)\leqslant R_i.RMS_i.RMSN_j*urq_j\\ \quad i=1,\cdots,RN,j=1,\cdots,MSN\\ \quad c_2(x):\bigcup_{i=1}^{UN}U_i.SIC_i\in\bigcup_{j=1}^{RN}\bigcup_{k=1}^{MSN}R_j.RMS_j.MS_k\\ \quad (R_j.RMS_j.RMSN_k>0)\\ \quad c_3(x):SIC_i\in SDGi=1,\cdots,UN\end{cases}$$

其中，t 表示用户请求分配处理过程中的每一个时刻，$c_1(x)$，$c_2(x)$，$c_3(x)$ 分别对应了上述的 3 个约束。

由问题描述可知，这属于典型的 NP-Hard 问题，使用传统的多项式求解算法很难对此问题求得最优解，因此本文采用多目标进化算法对该问题进行求解。

3 求解算法

本文提出的多目标微服务用户请求分配策略问题是具有 3 个目标的多目标优化问题。因此，本文首先采用了二维编码方式对染色体进行设计，然后根据模型需要设计了新的初始解生成策略以及改进的交叉算子和变异算子，并将其应用于 NSGAIII，SPEA2 和 MOEA/D 这 3 种目前较为主流和经典的多目标进化算法中，从而对上述问题进行求解。其中，因微服务实例可以并发处理多个用户请求且每个微服务实例可同时处理的用户请求数量存在上限，这使得用户请求反馈总时间的计算较为困难。为此，本文提出了计算该时间的详细算法，亦可为相关研究提供一定程度的借鉴。

NSGAIII[21] 是非支配排序算法（Non-Dominated Sorting Genetic Algorithm，NSGA）的一种变体，通过在保留具有绝对优势的精英策略的基础上，引入参考点策略来进一步提升精英解的求解效率，从而提升算法性能。SPEA2 是增强帕累托进化算法（Strength Pareto Evolutionary Algorithm，SPEA）的一种变体，采用小生境法和外部存档精英保留机制[22] 来提升算法的收敛速度和搜索性能，是目前较为流行的多目标优化算法。MOEA/D[23] 是基于分解的多目标进化算法，通过将一种多目标进化算法分解为若干单目标问题进行求解，其中每个子问题优化都采用了其相邻子问题的信息，各子问题协同进化，从而降低了计算复杂度，提升了搜索效率。

3.1 染色体编码

本文采用二维编码方式，即将染色体设计为一张二维表。其中，二维表的每一行代表某个用户请求微服务调用链中的一个微服务，共 $\sum_{i=1}^{UN}\sum_{j=1}^{U_i.SIC_i.length}1$ 行；二维表中的每一列代表一个资源中心，共 RN 列。二维表中每个位置上的元素的取值只有 0 或 1 两种，其中，1 表示将此行对应的微服务分配到此列对应的资源中心相应的微服务实例上，0 则反之。考虑到假设 3），二维表中的每一行应有且仅有一个元素取值为 1，其他元素全为 0。染色体编码的具体形式如图 3 所示。

$$\begin{array}{c} \\ U_1.SIC_1(1)\\ \vdots\\ U_i.SIC_i(1)\\ \vdots\\ U_{UN}.SIC_{UN}(1)\\ \vdots\end{array}\begin{array}{c}\begin{array}{cccc}R_1 & R_2 & \cdots & R_{RN}\end{array}\\ \begin{pmatrix}0/1 & 0/1 & \cdots & 0/1\\ \vdots & \vdots & \ddots & \vdots\\ 0/1 & 0/1 & \cdots & 0/1\\ \vdots & \vdots & \ddots & \vdots\\ 0/1 & 0/1 & \cdots & 0/1\\ \vdots & \vdots & \ddots & \vdots\end{pmatrix}\end{array}$$

图 3 染色体二维编码图示

Fig. 3 Chromosome two-dimensional encoding diagram

3.2 初始解生成

在进化算法中，初始解的质量对算法求解的质量和收敛速度都有着很大程度的影响，因此设计合适的初始解生成策略是算法求解的重要部分，常用的初始解生成策略主要有贪婪寻优和随机赋值等。针对本文提出的多目标微服务用户请求分配策略问题，本文提出了一种约束满足的初始可行解生成策略，从而尽可能多地生成可行解以提高初始种群的质量。

初始解生成策略可描述为：1）针对二维表中的每一行，找出该行所对应的用户请求，及该用户请求微服务调用链中的微服务；2）寻找上述微服务对应的微服务启动实例数量大于 0 的全部资源中心；3）从上述得到的资源中心集合中随机选取一个资源中心，并将其在该行中对应的位置上的元素的取值设为 1，该行其他位置上的元素的取值设为 0。

容易分析得知，经过该策略生成的初始解一定是可行解。

3.3 交叉算子

交叉算子是进化算法中非常重要的一种操作，设计良好的交叉算子可在很大程度上影响种群进化的效率，常见的交叉算子主要有单点交叉、两点交叉、融合交叉和均匀交叉等。针对本文提出的多目标微服务用户请求分配策略问题，为了保证尽可能多地生成可行解，本文采用了一种行列交叉算

子[24]来进行交叉操作。

行列交叉算子可主要描述为：1）设置交叉概率，以此概率为是否进行交叉操作的判断；2）若执行交叉操作，则以 50% 的概率选择进行行交叉或列交叉。行交叉指，对于进行交叉操作的两个父代个体，逐行随机地从两个个体中选择一个来构成子代个体的每一行；列交叉指，对于进行交叉操作的两个父代个体，逐列随机地从两个个体中选择一个来构成子代个体的每一列。

若两个父代都是可行解，则每一行有且只有一个元素取值为 1，那么进行行交叉后，该约束依然满足，因此只通过行交叉获得的个体均为可行解；然而，当执行列交叉操作时，得到的子代的每一行中有一定概率存在两个或多个取值为 1 的元素，使得假设 3）不成立，从而得到一个不可行解。因此，通过该行列交叉算子，可以以较大概率获得新的可行解，在一定程度上提高了进化的效率。

3.4 变异算子

变异算子同样是进化算法中的重要一环，通过变异操作可以在一定程度上增加解的多样性，提高算法全局搜索的能力，常用的变异算子主要有基本位变异、均匀变异和高斯变异等。针对本文提出的多目标微服务用户请求分配策略问题，为了保证尽可能多地生成可行解，本文设计了一种新的变异算子来进行变异操作。

该变异算子可以描述为：1）设置变异概率，以此概率作为二维表中每一行是否执行变异操作的依据；2）若对某一行执行变异操作，则从该行取值为 0 的元素中随机选取一个元素，并将其取值设置为 1，将原先取值为 1 的元素设置为 0。

对于一个可行解，通过此变异操作，可使二维表的每一行始终有且仅有一个元素取值为 0，从而使得假设 3）成立，得到的是新的可行解。但对每一行含有两个及以上取值为 1 的元素的不可行解进行此变异操作时，生成的也一定是不可行解。

3.5 请求反馈总时间的计算

请求反馈总时间作为多目标微服务用户请求分配策略问题的一个很重要的评价指标，其计算影响到整个算法的求解过程。与传统的车间调度问题明显不同的是，车间调度问题假设一台机器在某个时刻只能加工一个工件，而对于多目标微服务用户请求分配策略问题，资源中心启动的每个微服务实例可以同时进行多个用户请求的处理。此外，若等待处理的微服务请求的数量超过了其分配的微服务实例的处理能力，则超出部分的请求需要等待其他请求处理完毕才能被处理。同时，每个用户请求的微服务调用链由多个不同的微服务组成，且微服务之间具有先后顺序关系，这就导致可能存在这样一种情况，前序微服务因超过处理能力而排队等待，后序微服务即使没有超过处理能力，也会因这种先后顺序约束的存在而强行等待。上述情况的存在使得请求反馈总时间的计算变得困难。

因此，本文提出了一种请求反馈总时间（Total Requests Feedback Time，TRFT）的计算方案，如算法 1 所示。

算法 1 TRFT 计算方案

输入：(R,U,MS,SDG)

输出：TRFT

1. TRFT=0
2. In MS，获取当前可以准备执行的所有微服务请求 $MS_{readyToDo}$/ * 即没有前序微服务或前序微服务已被处理完成的微服务请求 * /
3. In $MS_{readyToDo}$，寻找当前无须排队的所有请求 $MS_{noWaiting}$/ * 根据各资源中心微服务实例的上限，按请求编号递增顺序选择 * /
4. In $MS_{noWaiting}$，寻找具有最小耗费时间 MET 的请求 MS_{MET}/ * 每个请求的初始耗费时间 ET 包含两部分：前序微服务（或用户位置）到该微服务的微服务调用时延（或请求传输时间），以及该微服务的处理时间 * /
5. For all ms in $MS_{noWaiting}$，do：
   ms. ET−=MET
   End for
   Remove MS_{MET} in MS
6. TRFT+=MET
7. Repeat 2−6，until MS. size()==0
8. Return TRFT

4 实验分析

4.1 实验数据

本实验采用模拟数据对算法进行测试。模拟数据主要包括微服务集合、微服务依赖图、资源中心集合和用户请求集合。

（1）微服务依赖图：采用图 1 所示的 *SDG* 进行模拟数据的生成，共包含 16 个微服务。其中，每个微服务的处理时间为 0～1 间的随机实数，每个微服务可同时处理的用户请求数量为 1～5 间的随机整数。

（2）资源中心集合：实验模拟了 3 个资源中心，每个资源中心的位置为(0.0,0.0)，(0.5,0.5)，(1.0,1.0)，每个资源中心启动的各微服务实例的数量可自定义设置，在本实验中均设置为 1，并随机挑选了 3 个某资源中心的某微服务实例，将其启动数量设置为 2。

（3）用户请求集合：本实验模拟了 120 条用户请求，其中每个用户请求的位置坐标（经度、纬度）均为 0～1 间的随机实数，每个用户请求包含的微服务调用链为从微服务依赖图中随机挑选一条。

4.2 参数设置

本实验采用了 NSGAIII，SPEA2 和 MOEA/D 算法对多目标微服务用户请求分配策略问题进行求解，其中采用自定义交叉和变异算子的两种算法（RCC_SM_NSGAIII 算法、RCC_SM_ SPEA2 算法）的实验参数取值如表 1 所列；采用自定义交叉和变异算子的 MOEA/D 算法（RCC_SM_MOEAD 算法）的种群大小（population size）设置为 100，最大评估次数（max evaluations）设置为 1 000。

表 1 RCC_SM_NSGAIII 和 RCC_SM_SPEA2 的参数

Table 1 Parameters of RCC_SM_NSGAIII and RCC_SM_ SPEA2

参数名称	方法或取值
交叉算子	见 3.3 节
交叉概率	0.9
变异算子	见 3.4 节
变异概率	1.0/二维表元素总数
种群数量	16
最大迭代次数	32

同时，为了验证自定义交叉算子和变异算子的有效性，本文采用了整数 SBX 交叉算子(Integer SBX Crossover)和整数多项式变异算子(Integer Polynomial Mutation)，分别将其应用于 NSGAIII 算法(SBX_IPM_NSGAIII)、SPEA2 算法(SBX_IPM_SPEA2)和 MOEA/D 算法(SBX_IPM_MOEAD)，前两种算法的参数设置如表 2 所列，SBX_IPM_MOEAD 算法的种群大小和最大评估次数同样为 100 和 1 000。

表 2 SBX_IPM_NSGAIII 和 SBX_IPM_SPEA2 的参数

Table 2 Parameters of SBX_IPM_NSGAIII and SBX_IPM_SPEA2

参数名称	方法或取值
交叉算子	整数 SBX 交叉算子
交叉概率	0.9
交叉分布索引值	20
变异算子	整数多项式变异算子
变异概率	0.01
变异分布索引值	20
种群数量	16
最大迭代次数	32

4.3 实验环境

本实验在基于 Intel(R) Core (TM) i5-6200U CPU @ 2.30GHz 2.40GHz 处理器的 Win10 64 位操作系统的计算机上运行，运行内存为 12GB。

4.4 结果分析

为了验证算法改进的有效性，本实验除了直接对各算法的求解结果进行比较外，还使用了国际通用的反世代距离(Inverted Generational Distance，IGD)，通过在数据集上进行 10 次重复实验，得到相应的 IGD 值进行比较。比较结果如图 4—图 7 所示，图中的每个点对应一种算法求得的可行解所对应的 3 个目标函数的取值。

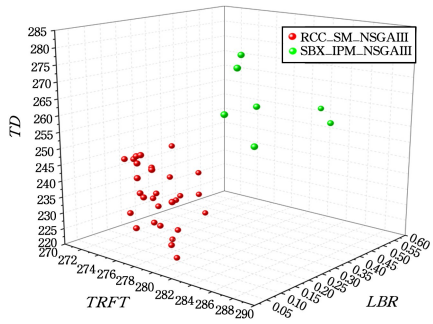

图 4 NSGAIII 不同算子的效果对比图

Fig. 4 Comparison of different operators of NSGAIII

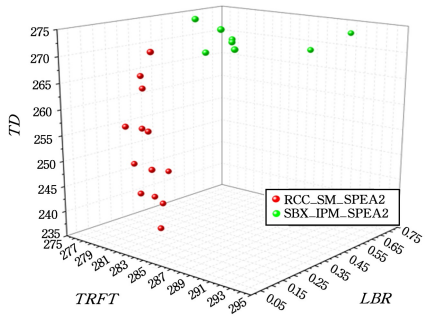

图 5 SPEA2 不同算子的效果对比图

Fig. 5 Comparison of different operators of SPEA2

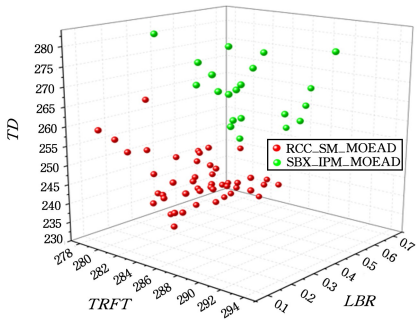

图 6 MOEA/D 不同算子的效果对比图

Fig. 6 Comparison of different operators of MOEA/D

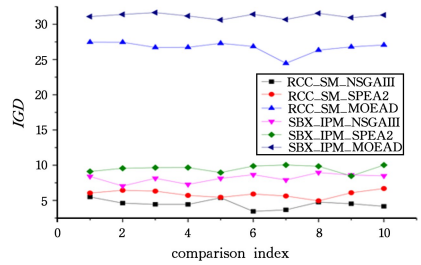

图 7 IGD 的比较结果图

Fig. 7 IGD comparison chart

由于多目标微服务用户请求分配策略模型的 3 个目标(请求反馈总时间、负载均衡率和通信传输总距离)均以最小化为优化方向，因此效果对比图中，越靠近左下角的点意味着解的质量越好。同时，算法求解的 IGD 值越小，意味着算法的求解性能越好。从图 4—图 6 可以看出，改进算法求解效果更好。分析造成此结果的原因可能有 3 点：1)是初始解生成策略的影响，原始算法中的初始解采用随机赋值的方式进行生成，这就使得染色体矩阵中每一行有且仅有一个元素为 1 的约束很难满足，因此只有极小的概率随机生成一个可行解，从而使得初始种群的质量极差，而改进的初始解生成策略从初始解的构造过程出发，初始解的每一个子块(每一行)都必须满足上述约束的限制，因此构造的初始解一定是可行解，从而极大地丰富了初始种群的多样性，提高了初始种群的质量；2)交叉算子的影响，为了算子的普适性，通用的交叉算子一般会尽量减少不必要的以问题为导向的启发信息，这就使得其对约束要求较为苛刻的问题缺乏适应性，从而影响子代解的质量，而改进的交叉算子在一定程度上弥补了这种适应性的缺乏，以问题的约束为导向，力求在子代构造过程中的每一步都满足约束，从而提高了子代个体为可行解的概率，进一步提升了种群的质量；3)变异算子的影响，同交叉算子的分析，改进的变异算子同样是用以问题约束为导向的方法来提高变异算子的适应性，从而提高生成子代可行解的概率，进一步提高种群的质量。因此，本文提出的初始解生成策略以及自定义交叉和变异算子可以显著改进算法的性能。同时，图 7 进一步证实了改进算法的有效性。此外，本实验还与常用于用户请求分发的轮询(Polling)与随机(Random)策略进行了对比。轮询策略即将用户请求按顺序逐个分配到每个资源中心相应的微服务实例上；随机策略即随机选择一个资源中心进行用户请求的分配。其比较结果如图 8 所示。

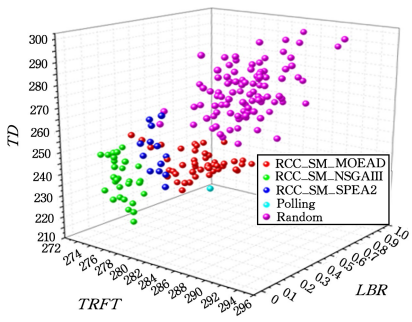

图 8　所有算法及策略的效果对比图

Fig. 8　Comparison chart of all algorithms and strategies

由图 8 可知，RCC_SM_NSGAIII 算法的求解效果显著好于 RCC_SM_SPEA2 算法、RCC_SM_MOEAD 算法、轮询策略和随机策略；RCC_SM_MOEAD 算法、RCC_SM_SPEA2 算法和轮询算法在各个目标上各有优劣；随机策略求解效果最差。这种实验结果在一定程度上是可以预见的。进化多目标算法及其改进算法以牺牲决策时间为代价，通过多次迭代不断进行解的优化，最终得到一个性能良好的满意解。而轮询策略则以牺牲一定的质量为代价，极大地缩短了算法的运行时间。随机策略具有不确定性，以概率获得不同质量的解，总体性能较差。

为了进一步验证 RCC_SM_NSGAIII 算法的求解性能，本实验比较了 3 种改进算法在用户请求数量分别为 140，160，180 和 200 时的 IGD 值，比较结果如图 9 所示。可以看出，RCC_SM_NSGAIII 算法在不同规模的数据集上具有良好的适应性和求解性能。

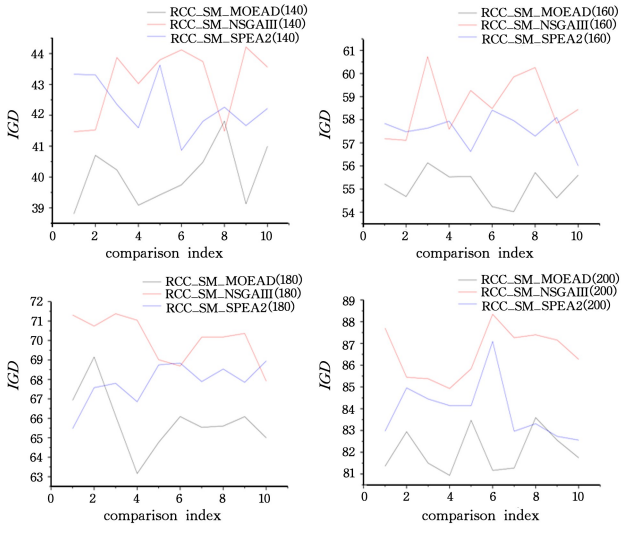

图 9　不同规模数据集下的 IGD 值比较图

Fig. 9　IGD comparison chart in different scale data sets

因此，对于多目标微服务用户请求分配策略问题的求解，使用本文提出的初始解生成策略和自定义交叉算子和变异算子的 RCC_SM_NSGAIII 算法是求解效率最高且效果最好的。

结束语　微服务架构作为复杂应用系统的新型架构形式，在灵活性、可扩展性、容错性等方面展示出了良好的发展态势。而微服务架构的形式，也给系统的用户请求分配策略带来了新的挑战。如何对用户请求进行多资源中心多微服务实例的分配，以使得用户请求反馈总时间、系统负载均衡率和用户请求通信传输总距离 3 个可能冲突的目标尽可能达到最优，成为了需要解决的问题之一。

针对上述问题，本文引入多目标进化算法（SPEA2，NSGAIII，MOEA/D）进行求解。首先构建了多目标微服务用户请求分配模型，明确了优化目标和约束条件，并做出了一定的假设；然后通过自定义初始解生成策略、交叉算子和变异算子，对 SPEA2 算法、NSGAIII 算法和 MOEA/D 算法进行了优化，使得算法更适用于本模型的求解；最后通过与常用的随机和轮询策略的对比，证明采用了自定义交叉和变异算子的 NSGAIII 算法在求解该问题时具有显著的优势，取得了不错的求解效果。

本文模型是在已知用户请求集合的前提下进行请求分配，但在实际生产环境下，当大量用户请求到达系统时，若采用本文算法可能导致用户的等待时间过长。因此，如何有效地预测在未来一段时间内的用户请求的组成，是需要继续深入研究的问题。

参 考 文 献

[1] THONE S, JOHANNE S. Microservices[J]. IEEE Software, 2015, 32(1): 116.

[2] LI Z H. Analysis of the development and impact of microservices architecture[J]. China CIO News, 2017(1): 154-155.

[3] MA S P, FAN C Y, CHUANG Y, et al. Using Service Dependency Graph to Analyze and Test Microservices (Conference Paper)[J]. Proceedings-International Computer Software and Applications Conference, 2018, 2: 81-86.

[4] LEITNER P, CITO J, STOCKLI E. Modelling and Managing Deployment Costs of Microservice-Based Cloud Applications [C]// IEEE/ACM 9TH International Conference on Utility and Cloud Computing (UCC). 2016: 165-174.

[5] CORNEL B, HAMZEH K, MARIOS F, et al. Delivering Elastic Containerized Cloud Applications to Enable DevOps [C] // IEEE/ACM 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS). 2017: 65-75.

[6] REN N N. Research and Implementation of Performance Optimization Technology of Microservice-based Applications in Clouds [D]. Xi'an: Xidian University, 2018.

[7] MA W B, WANG R, WANG W C, et al. Micro-service composition deployment and scheduling strategy based on evolutionary multi-objective optimization[J]. Systems Engineering and Electronics, 2020, 42(1): 90-100.

[8] ZHOU X, PENG X, XIE T, et al. Benchmarking Microservice Systems for Software Engineering Research[C] // 40th ACM/IEEE International Conference on Software Engineering (ICSE). 2018: 323-324.

[9] FU L L, ZOU S W. Research on Container Deployment of Microservices[J]. Computing Technology and Automation, 2019, 38(4): 151-155.

[10] XU C J, ZHOU X, PENG X, et al. Microservice System Oriented Runtime Deployment Optimization [J]. Computer Applications

and Software, 2018, 35(10): 85-93.
[11] XIA T Y, XU J Q, JIANG M. Research of Multi-Objective Optimization Based Algorithm for Docker-Microservices Placement [J]. Artificial Intelligence and Robotics Research, 2017, 6(2): 41-55.
[12] MARIA F, ANTONIO C, RAJIV R, et al. Open Issues in Scheduling Microservices in the Cloud[J]. IEEE Cloud Computing, 2016, 3(5): 81-88.
[13] ION-DORINEL F, FLORIN P, CRISTINA S, et al. Microservices Scheduling Model Over Heterogeneous Cloud-edge Environments As Support for IoT Applications[J]. IEEE Internet of Things Journal, 2018, 5(4): 2672-2681.
[14] BAO L, CHASE W, BU X X, et al. Performance modeling and workflow scheduling of microservice-based applications in clouds (Article)[J]. IEEE Transactions on Parallel and Distributed Systems, 2019, 30(9): 2101-2116.
[15] TANG Y. Design and Implementation of Job Scheduling System Based on Microservice Architecture[D]. Chengdu: Southwest Jiaotong University, 2019.
[16] YANG L, CAO J N, LIANG G Q, et al. Cost Aware Service Placement and Load Dispatching in Mobile Cloud Systems[J]. IEEE Transactions on Computers, 2016, 65(5): 1440-1452.
[17] ZHENG X J, LI J. Cost optimization of request dispatching and container deployment in cloudlets[J]. Journal of University of Science and Technology of China, 2019, 49(10): 820-827.
[18] XU Y. Research and implementation of related technology in web ar service platform based on micro service architecture[D]. Beijing: Beijing University Of Posts And Telecommunications, 2019.
[19] TAO X Y. Research on Techniques of Flow Scheduling and Request Allocation in Data Centers[D]. Dalian: Dalian University of Technology, 2019.
[20] ZHANG T F, MA Y, LI L, et al. Improved Genetic Algorithm for Flexible Job Shop Scheduling Problem[J]. Journal of Chinese Computer Systems, 2017, 38(1): 129-132.
[21] DEB K, JAIN H. An Evolutionary Many-Objective Optimization Algorithm Using Reference-Point-Based Nondominated Sorting Approach, Part I: Solving Problems With Box Constraints[J]. IEEE Transactions on Evolutionary Computation, 2014, 18(4): 577-601.
[22] WENG L G, WANG A, XIA H, et al. Improved SPEA2 based on local search[J]. Application Research of Computers, 2014(9): 2617-2619.
[23] GAO J L, XING Q H, FAN C L, et al. Double adaptive selection strategy for MOEA/D[J]. Journal of Systems Engineering and Electronics, 2019, 30(1): 132-143.
[24] LEANDRO L M, DIRK S, YAO X. Improved Evolutionary Algorithm Design for the Project Scheduling Problem Based on Runtime Analysis[J]. IEEE Transactions on Software Engineering, 2014, 40(1): 83-102.

ZHU Han-qing, born in 1997, postgraduate. His main research interests include microservices and scheduling theory.

HUANG Hong-bin, born in 1975, Ph.D, professor, Ph. D supervisor. His main research interests include CPS and data analysis.