计算机研究与发展
Journal of Computer Research and Development
DOI: 10.7544/issn1000-1239.202220736
60(5): 1073−1085, 2023

面向边缘计算的服务解耦与部署策略

李丽颖[1,2] 张润泽[1] 魏同权[1]
[1](华东师范大学计算机科学与技术学院 上海 200062)
[2](南京理工大学计算机科学与工程学院 南京 210094)
(liyingli@njust.edu.cn)

Service Decoupling and Deployment Strategy for Edge Computing

Li Liying[1,2], Zhang Runze[1], and Wei Tongquan[1]
[1](*School of Computer Science and Technology*, *East China Normal University*, *Shanghai* 200062)
[2](*School of Computer Science and Engineering*, *Nanjing University of Science and Technology*, *Nanjing* 210094)

Abstract In the era of the Internet of everything, the huge transmission delays between massive devices and data, clouds and devices have brought multiple challenges to developing applications, processing data, and ultimately improving the quality of service(QoS). As a result, edge computing architectures that significantly reduce the amount of data transferred to cloud servers and the response time of service requests by deploying computing power near end devices have emerged. However, load balancing, security, and mobility of edge devices are still the key points that affect the quality of service of edge computing architectures. In order to solve the above problems, we propose a two-stage QoS optimization scheme to solve the service deployment problem under the mobile edge computing architecture. In the first stage, we consider the decoupling of services and the load balancing of edge servers, model the problem of service deployment, propose a real-time decoupling scheme for centralized services, and design a static deployment strategy; In the second stage, we consider the mobility of edge devices, design a dynamic deployment strategy for device mobility-aware services, and optimize the quality of service under the mobile edge computing architecture. The experimental results on two datasets show that the static deployment strategy proposed in this paper can reduce the response time of service requests by 36%, and the dynamic deployment strategy can further reduce the service response time by 13%.

Key words edge computing; quality of service(QoS); response time; service decoupling; service deployment

摘 要 在这个万物互联的时代,海量的设备和数据、云和设备之间存在巨大的传输延迟,给开发应用、处理数据、最终提升系统的服务质量带来了多重挑战.因此,通过在终端设备附近部署计算能力以显著减少传输到云服务器的数据量和服务请求的响应时间的边缘计算结构应运而生.然而,负载均衡、边缘设备的安全性和移动性依旧是影响边缘计算架构的服务质量的关键点.为了解决上述问题,提出了一种2阶段的服务质量优化方案以解决移动边缘计算架构下的服务部署问题.在第1阶段,考虑了服务的解耦和边缘服务器的负载均衡问题,对服务部署问题进行建模,提出了一种集中式服务的实时解耦方案,并且设计了一种静态部署策略;在第2阶段,考虑了边缘设备的移动性,设计了一种设备移动感知的服务动态部署策略,优化了移动边缘计算架构下的服务质量.在2个数据集上的实验结果表明,所提出的静态部署策略

收稿日期:2022-08-22;**修回日期**:2023-03-30
基金项目:国家自然科学基金面上项目(62272169);上海市市级科技重大专项(2021SHZDZX);上海市可信工业互联网软件协同创新中心项目
This work was supported by the General Program of the National Natural Science Foundation of China (62272169), the Shanghai Municipal Science and Technology Major Project (2021SHZDZX), and the Project of Shanghai Trusted Industry Internet Software Collaborative Innovation Center.
通信作者:魏同权(tqwei@cs.ecnu.edu.cn)

能够使服务请求的响应时间降低36%,所提出的动态部署策略能进一步降低13%的服务响应时间.

关键词 边缘计算;服务质量;响应时间;服务解耦;服务部署

中图法分类号 TP301

随着物联网和5G无线网络的快速发展,万物互联已经成为一种趋势,促进了各个领域新兴应用的发展[1]. 与此相对应的是边缘设备及其产生的数据量迅速增长. 麦肯锡全球研究报告称,预计到2025年,物联网连接的设备将超过300亿个,相关的数据量也从2016年的96 ZB/月增长到了2021年的278 ZB/月[2]. 这些海量而多样的设备和数据连接物与物、物与人,最终实现“万物互联”[3].

关键技术的蓬勃发展也使得分布式的用户请求数量迅速增加. 然而,当前的服务模式大多是将应用程序的功能封装为独立的单元并将该单元部署在云服务器上. 分布式的用户向云端服务器发送服务请求,云端服务器运行相应的应用程序并向用户反馈结果. 然而,这种集中式的服务往往会带来巨大的服务响应延迟和带宽开销,降低服务质量(quality of service, QoS)[4].

解决上述问题的一种有效的方法是将这种集中式服务解耦为子服务. 例如,Auer等人[5]提出了一个基于问卷调查的评估框架,他们通过问卷调查归纳出将集中式服务解耦并部署到子服务架构上的重要指标,并根据问卷调查的结果来推断具体的集中式服务具体如何解耦并部署到子服务架构中. 然而,这种基于问卷调查的解耦方案受限于问卷设计,通常都比较主观,其效果和普适性较差,并且会耗费大量的人力和时间成本. 为了解决上述问题,Arisholm等人[6]提出了动态耦合度的概念用于衡量应用程序在实际运行时的相关程度. Poshyvanyk等人[7]提出了一种用于衡量程序的类之间的标识符和相关性的耦合度. 然而,这些现有的工作忽略了服务自身的特性(如服务的字段等),因而效果不佳.

除了对集中式服务进行解耦,将解耦后的子服务部署到边缘计算架构中也是一个能够有效降低时延的解决方案. 边缘计算利用终端设备的计算能力,可以显著减少传输到云服务器的数据量和服务请求的响应时间[8-9]. 然而,由于边缘设备的存储空间和计算能力有限,如何将解耦后的子服务部署在边缘服务器上是亟待解决的问题. 现有关于边缘计算架构下的服务部署的相关工作如文献[10−12],并没有同时考虑边缘服务器本身的异构性和服务器之间的负载均衡,并且脆弱的边缘设备也带来了严重的安全性问题. 首先,对于大多数边缘计算相关技术(如无线网络和分布式系统等),不仅需要保护其中的组件免受网络攻击,还需要协调不同等级的安全机制. 其次,对边缘计算而言,在异构的系统下提供稳定的连接和高可访问性是提供高质量服务的基础[13]. 因此,边缘服务器在保证网络安全的条件下及时响应用户的服务请求就显得尤为重要.

移动设备数量的爆炸式增长推动了传统的边缘计算向移动边缘计算(mobile edge computing, MEC)的方向发展. 移动边缘计算[14]将云端的计算能力推向了边缘端,即在移动设备和无线接入网的边缘部署计算平台,使得移动设备可以就近使用所需服务和云端计算功能[15]. 尽管移动边缘计算可以显著地降低端到端的延迟,但是设备移动的不可预测性给部署服务带来了新的挑战[16]. Wang等人[17]设计了一种服务迁移策略来解决顺序决策服务的部署问题. Taleb等人[18]基于马尔可夫链模型提出了一种面向移动设备的边缘计算部署策略. 然而,文献[14−18]所述工作都没有考虑到移动边缘计算服务部署的成本问题.

本文提出了一种2阶段的服务质量优化方案以解决移动边缘计算架构下的服务部署问题. 在第1阶段,本文考虑了服务的解耦和边缘服务器的负载均衡问题,对服务部署问题进行建模,提出了一种集中式服务的实时解耦方案,并且设计了一种静态部署策略;在第2阶段,本文考虑了边缘设备的移动性,设计了一种设备移动感知的服务部署策略,优化了移动边缘计算架构下的服务质量. 具体来说,本文的主要贡献包括3个方面:

1)在考虑边缘服务器负载均衡的前提下提出了一种面向集中式服务的解耦策略,并设计了一种基于分解的多目标进化算法的服务部署策略以优化服务质量. 该策略能够在保证服务器负载均衡和安全性的前提下,优化服务请求的响应时间.

2)进一步考虑了边缘设备的移动性,提出了一种动态服务部署策略. 首先,为了适应移动的服务请求,建模动态用户的服务部署问题. 然后,提出了一种设备移动感知的实时服务放置策略以进一步优化

服务请求的响应时间.

3）为了验证所提方法的有效性，本文进行了一系列的仿真实验. 实验结果表明，与多种基准方法相比，在保证系统安全以及服务器负载均衡的前提下，本文所提出的面向集中式服务静态部署策略能够服务请求的响应时间降低 36%，所提出的动态部署策略能进一步降低 13% 的服务响应时间.

1 系统模型

本节主要介绍本文主要用到的相关模型，包括系统架构模型、服务模型、耦合度模型和传输成本模型.

1.1 系统架构模型

不失一般性，本文采用常见的移动边缘计算系统架构. 假设该架构硬件上包含 1 个云服务器、N_{es} 个边缘服务器、N_{bs} 个基站和 N_{td} 个终端设备. 基站之间通过有线连接，终端设备与基站之间通过无线连接，终端具有可移动性. 边缘服务器的集合用 $ES=\{ES_1, ES_2, \cdots, ES_{N_{es}}\}$ 表示，其中 ES_i $(1\leqslant i\leqslant N_{es})$ 为第 i 个边缘服务器. 基站的集合用 V_{bs} 表示，终端设备的集合用 V_t 表示. 系统架构图如图 1 所示，其拓扑关系可以用无向图 $G=\{V, E\}$ 表示，其中 $V=V_{bs}\cup V_{td}$，$E=E_{wire}\cup E_{wireless}$，其中 E_{wire} 为 2 基站之间所有的有线连接组成的集合，$E_{wireless}$ 为所有基站和终端设备之间的无线连接所组成的集合.

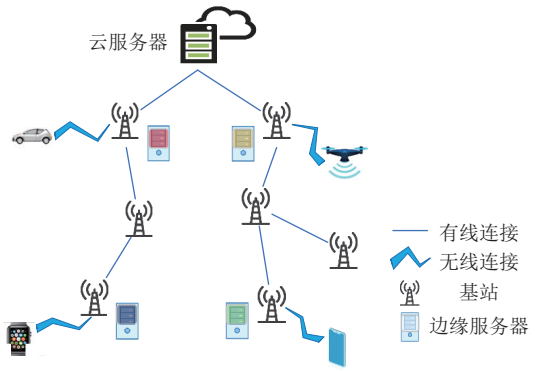

Fig. 1 System architecture

图 1 系统架构

在云端服务器上维护完整的集中式服务，并且负责将集中式服务解耦为 N_{ser} 个子服务并且将这些子服务部署/卸载到 N_{es} 个边缘服务器上. 每个边缘服务器通过使用子服务来快速响应终端设备的服务请求，这些边缘服务器在计算能力和存储空间上是异构的. 由于基站数量的不断增加，在所有的基站上部署边缘服务器会带来巨大的成本，因此并不是所有的基站附近都会部署边缘服务器，即 $N_{es}\leqslant N_{bs}$. 在本文中，使用文献 [19] 所提出的方法将 N_{es} 个边缘服务器部署在 N_{bs} 个基站上. 基站会负责接收邻近终端设备的服务请求，这个基站被称为这些终端设备的本地基站，对应的边缘服务器被称为这些终端设备的本地边缘服务器. 如果终端设备的本地基站部署了边缘服务器，且终端设备的服务请求可以在本地边缘服务器上处理，则请求将直接在本地边缘服务器上处理，否则这个服务请求将被转发到可以处理该请求的边缘服务器上，结果会被返回给终端设备.

1.2 服务及耦合度模型

假设云服务器维护 N_{ser} 个服务，服务集合 $S=\{S_1, S_2, \cdots, S_{N_{ser}}\}$，每个服务 S_j $(1\leqslant j\leqslant N_{ser})$ 包含 M_j 个字段. 定义一个 $M_j\times N_{es}$ 大小的矩阵 $\boldsymbol{I}_j$ 来表示服务中各个字段的被订阅情况：

$$\boldsymbol{I}_j=\begin{pmatrix} I_{j,1}^1 & \cdots & I_{j,1}^{N_{es}} \\ \vdots & & \vdots \\ I_{j,M_j}^1 & \cdots & I_{j,M_j}^{N_{es}} \end{pmatrix}, \tag{1}$$

如果边缘服务器 ES_i $(1\leqslant i\leqslant N_{es})$ 订阅服务 S_j $(1\leqslant j\leqslant N_{ser})$ 上的第 m $(1\leqslant m\leqslant M_j)$ 个字段，则 $I_{j,m}^i=1$，否则 $I_{j,m}^i=0$. 易得，矩阵 $\boldsymbol{I}_j$ 中第 i 列为边缘服务器 ES_i 对服务 S_j 的订阅情况. 对服务中字段的操作包括 3 种：删除、增加和修改. 我们定义了 3 个指示变量，以分别表示对服务 S_j 中字段 m 的操作.

$$\mu_{j,m}^{\mathrm{del}}=\begin{cases}1, \text{服务}S_j\text{删除字段}m,\\ 0, \text{其他}.\end{cases} \tag{2}$$

$$\mu_{j,m}^{\mathrm{add}}=\begin{cases}1, \text{服务}S_j\text{增加字段}m,\\ 0, \text{其他}.\end{cases} \tag{3}$$

$$\mu_{j,m}^{\mathrm{upd}}=\begin{cases}1, \text{服务}S_j\text{更新字段}m,\\ 0, \text{其他}.\end{cases} \tag{4}$$

对于边缘服务器 ES_i $(1\leqslant i\leqslant N_{es})$，定义一个对服务 S_j 中字段 m 的影响因子：

$$f_{j,m}^i=I_{j,m}^i\left(\omega^{\mathrm{del}}\mu_{j,m}^{\mathrm{del}}+\omega^{\mathrm{add}}\mu_{j,m}^{\mathrm{add}}+\omega^{\mathrm{upd}}\mu_{j,m}^{\mathrm{upd}}\right), \tag{5}$$

其中 ω^{del}, ω^{add}, ω^{upd} 分别为删除、增加和更新字段 m 对服务 S_j 影响的权重因子. 需要注意的是，这 3 种对服务中字段的操作都是原子操作. 也就是说，在每次对字段的操作中，有且只有上述 3 种操作的其中一种. 服务的耦合度用于判定服务是否需要被解耦的依据. 将服务 S_j 在边缘服务器 ES_i 上的耦合度定义为

$$FS_j^i=\sum_{m=1}^{M_j} f_{j,m}^i. \tag{6}$$

因此，服务 S_j 的耦合度为

$$FS_j=\sum_{i=1}^{N_{es}}FS_j^i. \tag{7}$$

从式(6)(7)的定义可以看出，一个服务的耦合度越高，其中的字段操作越频繁. 如果不对这种高耦合度的服务进行解耦，产生的传输成本就会越高. 因此，本文的方案倾向于解耦对高耦合度的服务.

1.3 安全性模型

本文定义了一个包含了 K 种不同类型的安全服务的集合，该集合用 $A=\{A_1, A_2, \cdots, A_K\}$ 表示. 假设 sec_i ($1\leqslant i\leqslant K$)为服务 S_j 使用安全服务 A_i 后的安全等级. $T(A_i, S_j)$ 为服务 S_j 使用安全服务 A_i 时的开销：

$$T(A_i, S_j)=C_i\times sec_i, \tag{8}$$

其中 C_i 为常数系数，取决于具体的安全服务算法. 本文采用指数模型[20]计算服务 S_j 采用安全服务 A_i 失败的概率为

$$P_{\text{fail}}(S_j)=\begin{cases}1-\exp\{-\lambda\left(sec_j^{\min}-sec_i\right)\}, & sec_j^{\min}\geqslant sec_i,\\ 0, \text{其他},\end{cases} \tag{9}$$

其中 $sec_j^{\min}$ 是服务 S_j 最低要求的安全级别，λ 表示具体应用的安全风险的常数. 因此，服务被成功运行的概率为

$$QS(S_j)=1-P_{\text{fail}}(S_j). \tag{10}$$

整个系统的安全性可以被定义为各个服务的安全性的乘积：

$$QS_{\text{sys}}=\prod_{j=1}^{N_s}QS(S_j). \tag{11}$$

因此，为所有服务提供安全服务的总耗时为

$$T_{\text{ss}}=\sum_{j=1}^{N_s}T(A_i, S_j)Cost(S_j), \tag{12}$$

其中 $Cost(S_j)$ 为传输服务 S_j 所带来的开销，在 1.4 节详细定义了该值.

1.4 传输成本模型

为了将服务从云服务器卸载到边缘服务器，一般会将服务封装到 Internet 协议数据包中. 因此，除了要考虑服务本身的大小，还需要考虑一些额外信息(如头部信息等)所产生的开销. 假设这类开销用 $Cost_{\text{head}}$ 表示，则传输服务 S_j 所带来的开销被计算为

$$Cost(S_j)=Cost_{\text{head}}+\sum_{m=1}^{M_j}Cost(m), \tag{13}$$

其中，$Cost(m)$ 为服务 S_j 中第 m 个字段的大小.

假设服务 S_j 被解耦为 N_{sub} 个子服务，第 n 个子服务用 $S_n^{(j)}$ 表示，则 $S_j=S_1^{(j)}\cup S_2^{(j)}\cup\cdots\cup S_{N_{\text{sub}}}^{(j)}$. 卸载子服务的开销为

$$Cost\left(S_n^{(j)}\right)=Cost_{\text{head}}+\sum_{m=1}^{\left|S_n^{(j)}\right|}Cost(m), \tag{14}$$

其中 $\left|S_n^{(j)}\right|$ 为服务 S_j 解耦产生的子服务的个数.

在传输成本方面，假设设备通过无线连接与服务器进行通信，服务器之间通过电缆或者光纤的有线连接方式传输信息. 2 个边缘服务器 ES_i 和 ES_j ($1\leqslant i\leqslant N_{\text{es}}$, $1\leqslant j\leqslant N_{\text{es}}$)之间的传输时延为

$$T_{i,j}^{\text{trans}}=W_{i,j}/B_{i,j}, \tag{15}$$

其中 $W_{i,j}$ 和 $B_{i,j}$ 分别为边缘服务器 ES_i 和 ES_j 之间需要传输的数据量和带宽大小. 而这 2 个服务器之间的传播时延为

$$T_{i,j}^{\text{prop}}=D_{i,j}/\theta, \tag{16}$$

其中，$D_{i,j}$ 为 ES_i 和 ES_j 之间根据路由的物理链路距离，θ 为电缆/光缆中电信号的传播速率. 因此，可以定义 ES_i 和 ES_j 之间的有线总时延：

$$T_{i,j}^{\text{wire}}=T_{i,j}^{\text{trans}}+T_{i,j}^{\text{prop}}. \tag{17}$$

设备 i 与服务器 ES_j 之间的无线传输时延为

$$T_{i,ES_j}^{\text{wireless}}=W_{i,ES_j}\Big/TR_{i,ES_j}, \tag{18}$$

其中 W_{i,ES_j} 和 TR_{i,ES_j} 分别为设备 i 和 ES_j 之间需要传输的数据大小和最大传输速率. TR_{i,ES_j} 计算为[21]

$$TR_{i,ES_j}=B_{i,ES_j}\ln\left(1+P_{\text{sig}}/P_{\text{noi}}\right), \tag{19}$$

其中，B_{i,ES_j} 为设备 i 和 ES_j 之间的带宽，P_{sig}, P_{noi} 分别为设备 i 和 ES_j 之间的信号功率与噪声功率.

2 集中式服务的实时解耦方案以及部署策略

本文所提方案用于解决移动边缘计算架构下的服务安全部署问题. 本节主要介绍所提方法的第 1 阶段. 具体来说，在这个阶段，我们考虑了服务的安全性和边缘服务器的负载均衡问题，对服务部署问题进行建模，提出了一种集中式服务的实时解耦方案以及部署策略.

2.1 基于 *K*-Means 算法的服务实时解耦

根据 1.2 节对服务耦合度的定义可知，如果一个服务的耦合度越高，其中的字段被进行增加、修改和删除的操作越频繁. 如果不对这种高耦合度的服务进行解耦，会产生较高的传输成本. 因此，本文所提方案仅对高耦合度的服务进行解耦. 具体来说，定义一个服务的耦合度阈值 F_{th}. 如果服务 S_j 的耦合度 $FS_j>F_{\text{th}}$，则 S_j 被认为是高耦合度服务，需要被解耦；反之，则认为服务 S_j 为低耦合度服务，不需要被解耦.

在对服务进行耦合度分类后，采用文献 [22] 中所提出的算法 ISODATA（iterative self-organizing data analysis techniques algorithm），对服务进行初始解耦. 这种解耦方法原理简单且易于实现，然而，该解耦方式本质上是一种离线的解耦，不适合增加、删除和修改操作频繁的移动感知应用场景，其对应的响应时延也较长. 因此，本文提出了一种基于加权 *K*-Means 算法的服务解耦方案，该方案可以实现对服务的实时解耦.

K-Means 是一种迭代求解的聚类分析算法，是基于欧氏距离的聚类算法. *K*-Means 算法认为 2 个目标的距离越近，相似度越大. 具体来说，该算法首先会随机选择初始化后的 K 个样本作为初始聚类中心，这些聚类中心被表示为$\{C_1, C_2, \cdots, C_K\}$. 对于每个待聚类的样本，首先计算该样本和每一个聚类中心的欧氏距离，然后将样本分类到与其距离最小的聚类中心对应的类别，最后算出每个类别的平均值以更新聚类中心.

在本文的应用场景下，文献 [22] 所提出的 ISODATA 算法会产生一个对服务的初始解耦结果，即对服务集合中每一个服务的解耦方案，解耦出的子服务即为一个类别，这些子服务的中心即为聚类中心. 以服务S_j为例，假设 ISODATA 算法将服务S_j解耦为 N_{sub} 个子服务，其中，第 n 个子服务用$S_n^{(j)}$表示，则$S_j = S_1^{(j)} \cup S_2^{(j)} \cup \cdots \cup S_{N_{\text{sub}}}^{(j)}$. 为每个子服务定义了一个中心，在时隙 t 的第k个中心的定义为

$$C_k^t = \frac{C_k^{t-1} n_k^{t-1} \pm x_k^{t-1} m_k^{t-1}}{n_k^{t-1} \pm m_k^{t-1}}. \tag{20}$$

其中，x_k^t为时刻 t 第 k 个子服务中所增加（或删除）字段的中心；m_k^t为属于第 k 个聚类中心的新增加（或删除）字段的数量；n_k^t为时刻 t 第 k 个子服务中所包含的字段的数目，该值的计算式为

$$n_k^t = n_k^{t-1} \pm m_k^{t-1}. \tag{21}$$

时刻 t 中心的集合被表示为$\left\{C_1^t, C_2^t, \cdots, C_{N_{\text{sub}}}^t\right\}$. 需要注意的是，如果是增加字段，那么式（21）中的±取“+”；如果是删除字段，式（21）中的±取“−”；如果是更新字段，则m_k^t=0.

我们只对高耦合度的服务进行解耦. 具体来说，定义了一个耦合度阈值F_{th}，对于每一个服务S_j，采用式（7）计算其对应的耦合度FS_j. 若$FS_j > F_{\text{th}}$，则服务S_j为高耦合度服务，需要采用解耦算法进行解耦；反之，服务S_j为低耦合度服务，不需要被解耦. 提出的基于加权 *K*-Means 算法的实时服务解耦方案将 ISODATA 产生的结果作为初始解，实时地根据对服务中字段的操作（增加、删除和更新），不断地实时更新每一个服务的解耦方案，在考虑边缘服务器节点负载均衡的条件下优化服务请求的响应时间.

所提出的基于 *K*-Means 的服务解耦算法对应的伪代码如算法 1 所示.

算法 1. 基于 *K*-Means 的服务解耦算法.

输入：阈值F_{th}，服务集合 $S=\{S_1, S_2, \cdots, S_{N_{\text{ser}}}\}$；

输出：对每一个服务的解耦方案.

① 初始化：$\Omega=\varnothing$，$K=N_{\text{sub}}$；
② for each 服务S_j
③ $FS_j = 0$；
④ 采用式（7）计算服务S_j的耦合度；
⑤ if $FS_j > F_{\text{th}}$
⑥ 将服务S_j加入集合 Ω；
⑦ 采用 ISODATA 算法[22] 得到初始解耦方案和 K 个聚类中心的初始取值；
⑧ else
⑨ 不对服务S_j进行解耦操作；
⑩ end if
⑪ end for
⑫ for each $S_j \in \Omega$
⑬ for each 子服务中心C_k^t /*往服务S_j中增加字段*/
⑭ 计算增加字段与C_k^t之间的欧氏距离；
⑮ 将增加的字段加入与之最近的C_k^t对应的子服务；
⑯ 根据式（20）更新聚类中心C_k^t；
⑰ end for
⑱ for each 子服务中心C_k^t/*往服务S_j中删除或更新字段*/
⑲ 从对应的子服务中删除/更新字段；
⑳ 根据式（20）更新聚类中心C_k^t；
㉑ end for
㉒ end for
㉓ return 对每一个服务的解耦方案.

算法 1 以耦合度阈值F_{th}和服务集合 $S=\{S_1, S_2, \cdots, S_{N_{\text{ser}}}\}$作为输入，并输出对每一个服务的解耦方案. 算法 1 首先初始化需要被解耦的服务的集合为空集，该集合用 Ω 表示；然后将聚类中心的个数 K 设为预计解耦得到的子服务的个数 N_{sub}（算法 1 的行①）. 对于每一个服务S_j，首先将其耦合度FS_j初始化为 0，然后采用式（7）计算服务S_j的耦合度（算法 1 的行③④）. 如果服务S_j的耦合度大于耦合度阈值，则认为S_j为高

耦合度服务，此时需要对服务进行解耦，因此将S_j加入集合Ω，并采用ISODATA算法[22]得到初始解耦方案和K个聚类中心的初始取值；否则认为S_j为低耦合度服务，不需要对其进行解耦(算法1的行②~⑪)．对集合Ω中每一个服务S_j，如果此时需要增加其中的字段，则算法1首先计算增加的字段与此时每一个聚类中心C_k^t之间的欧氏距离，然后将增加的字段加入与之最近的C_k^t对应的子服务，并根据式(20)更新C_k^t(算法1的行⑬~⑰)．如果此时需要删除/更新服务S_j中的字段，则从对应的子服务中删除/更新相应字段，并根据式(20)更新C_k^t(算法1的行⑱~㉑)．需要注意的是，如果相应的操作是更新，则式(20)中$m_k^t=0$．也就是说，此时相应子服务中的字段数量不会改变．迭代结束后，算法1会返回对每一个服务的解耦方案．

2.2 基于MOEA/D的服务静态放置策略

在对服务进行解耦之后，我们设计了一种静态的服务部署策略将解耦后的子服务部署到边缘服务器上以优化服务质量．具体来说，首先将子服务的部署问题建模为一个多目标优化问题(multi-objective optimization problem, MOP)，然后采用基于分解的多目标进化算法(multi-objective evolutionary algorithm based on decomposition, MOEA/D)去解决该问题，以保证在服务器负载均衡和安全性的前提下，优化服务请求的响应时间．

结合第1节的建模，可以将本文所关注的问题建模为一个多目标优化问题，该多目标优化问题表示为

$$\begin{aligned}&\min\{F_M(x)=\{f_1(x),f_2(x),\cdots,f_M(x)\}\},\\&\text{s. t. } x\in Y,\end{aligned}\tag{22}$$

其中$F_M:Y\to\mathbb{R}^M$由M个实值函数组成，Y为决策空间，$\mathbb{R}^M$为目标空间．为了便于表述，给出2个定义．

定义1. 如果存在2个向量$\boldsymbol{u}$, $\boldsymbol{v}\in\mathbb{R}^M$，任意$i=1,2,\cdots,M$，都满足$u_i<v_i$，则称$\boldsymbol{u}$支配$\boldsymbol{v}$．

定义2. 如果不存在$x\in Y$，满足$F_M(x)$支配$F_M(\hat{x})$，则点$\hat{x}$为$F_M(x)$的一个帕累托最优解，帕累托集合$PS=\{\hat{x}|$不存在$x\in Y, F_M(x)$支配$F_M(\hat{x})\}$．假设由所有帕累托目标向量所组成的集合$PF=\{F_M(x)\in\mathbb{R}^M|x\in PS\}$为帕累托前沿．

MOEA/D算法将式(22)中的目标函数分解为M个实值函数的优化问题，然后通过解决这M个实值函数的优化问题得到式(22)中的帕累托集合．使用基于切比雪夫的分解算法来解决多目标优化问题，具体来说，首先定义目标优化问题：

$$\begin{aligned}&\min\left\{g_M(x)=\max_{1\leqslant i\leqslant M}\{\lambda_i|f_i(x)-z_i^*\}\right\},\\&\text{s. t. } x\in Y,\end{aligned}\tag{23}$$

其中$\lambda=\{\lambda_1,\lambda_2,\cdots,\lambda_M\}$是权重集合，$z^*=\{z_1^*,z_2^*,\cdots,z_M^*\}$是式(22)对应优化问题的参考解的集合．在生成1组均匀分布的权重集合$\lambda=\{\lambda_1,\lambda_2,\cdots,\lambda_M\}$并设置参考解$z^*$后，式(22)的帕累托前沿$PF$可以近似地分解成$M$个标量子问题，其中第$j$个子问题表示为

$$\begin{aligned}&\min\left\{g_M\left(x|\lambda^j,z^*\right)=\max_{1\leqslant i\leqslant M}\left\{\lambda_i^j|f_i(x)-z_i^*\right\}\right\},\\&\text{s. t. } x\in Y,\end{aligned}\tag{24}$$

其中$\lambda^j=\left\{\lambda_1^j,\lambda_2^j,\cdots,\lambda_M^j\right\}$．由于函数$g$关于$\lambda$是连续的，因此，如果$\lambda^i$无限接近于$\lambda^j$，则$g_M(x|\lambda^i,z^*)$也无限接近于$g_M(x|\lambda^j,z^*)$．所以邻近$\lambda^j$的权重向量可以帮助优化问题$g_M(x|\lambda^j,z^*)$．定义$\lambda^j$的邻居为邻近$\lambda^j$的权重向量，第$j$个子问题的邻居是$\lambda^j$的邻居对应的子问题．

基于切比雪夫的MOEA/D算法，在每一轮迭代中更新4个内容：

1) 分解成的子问题数量M以及子问题i的当前解$x^i\in Y$；

2) 定义$FV^i=F_M(x^i),\forall i=1,2,\cdots,M$；

3) $z=\{z_1,z_2,\cdots,z_M\}$，其中z_i是到目前函数$f_i(x)$的最优值；

4) 在搜索过程中存储的非支配解的集合，该集合被表示为EP．

本文提出了基于MOEA/D的集中式服务放置算法来探索集中式服务配置到边缘服务器的解决策略．具体来说，随机选择交叉算子和变异算子作为本文的遗传算子：

1) 交叉算子．本文所关注的多目标优化问题为，在保证服务器负载均衡和安全性的前提下，将云服务器维护N_{ser}个服务中每一个服务对应的N_{sub}个子服务部署至N_{es}个边缘服务器上，以优化服务请求的响应时间．因此，该问题的解空间为$[0,N_{\text{es}}-1]^{N_{\text{sub}}}$，并在这个解空间内随机生成一个整数作为交叉算子．为了便于理解，以子服务个数N_{sub}=7、边缘服务器个数N_{es}=3的情况为例．假设随机生成的交叉算子为3，即从位置3开始进行交叉(子服务编号从0开始)，则交叉情况如图2所示．

2) 变异算子．本文使用基于正态分布的随机数生成方式进行变异操作．具体来说，对于染色体i的每个位置，先以P_{mutation}的概率判断是否会发生变异，若发生变异，则在该位置加上一个正态分布的随机数τ，并将结果调整到限定范围内，即

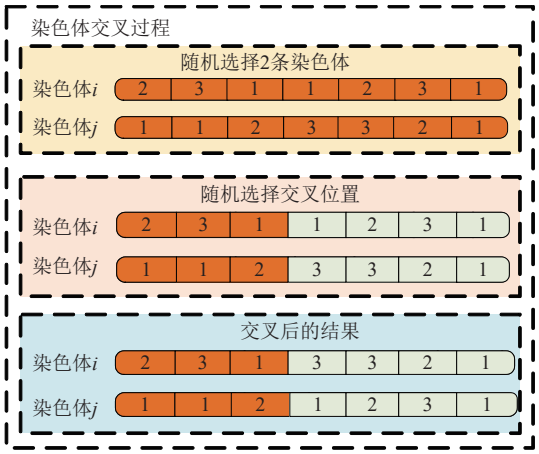

Fig. 2 An example of chromosome crossover process

图 2 染色体交叉过程示例

$$x_{i,j}^{t+1}=\begin{cases}x_{i,j}^{t}+Gauss(0,\sigma^2), & \tau<P_{\text{mutation}},\\ x_{i,j}^{t}, & \text{其他},\end{cases} \tag{25}$$

其中$x_{i,j}^{t}$表示第 t 轮迭代时第 i 个个体的第 j 个子服务部署在第$x_{i,j}^{t}$个边缘服务器上.

所提出的基于 MOEA/D 的子服务放置算法的伪代码如算法 2 所示.

算法 2. 基于 MOEA/D 的子服务放置算法.

输入: 最大迭代轮次$T_{\max}$, 子问题的数量N_{pop}, 权重向量集, 邻居数T_{nb};

输出: 对子服务的放置方案.

① $EP=\varnothing$, $PS=\varnothing$, $T=0$; /*初始化*/

② 随机生成的参考解$z=\{z_1,z_2,\cdots,z_{N_{\text{pop}}}\}$;

③ 计算每个权重向量的T_{nb}个邻居;

④ 随机生成种群$x^1,x^2,\cdots,x^{N_{\text{pop}}}$, 计算每个$FV^i$;

⑤ while $T\leqslant T_{\max}$

⑥ for each x^i in 种群

⑦ 生成基于个体 i 邻居的新解 y_i;

⑧ for each 邻居 j

⑨ if $g_{N_{\text{pop}}}(y|\lambda^j,z)\leqslant g_{N_{\text{pop}}}(x^j|\lambda^i,z)$

⑩ 更新$x^j=y_i, FV^j=z_j$;

⑪ end if

⑫ end for

⑬ end for

⑭ for PS 中的每一个 j

⑮ if z_j支配 y_i

⑯ 从EP中删除z_j;

⑰ 从PS中删除j;

⑱ end if

⑲ end for

⑳ if 在EP中没有向量支配 y_i

㉑ 向EP中添加z_j;

㉒ 从PS中删除j;

㉓ end if

㉔ $T=T+1$;

㉕ end while

㉖ return PS.

算法 2 以大迭代轮次$T_{\max}$、子问题的数量N_{pop}、权重向量集、邻居数T_{nb}为输入, 并输出对子服务的放置方案, 也就是本文提出的多目标优化问题对应的帕累托集合PS. 算法 2 首先将在搜索过程中存储的非支配解的集合EP和帕累托集合PS初始化为空集, 并且将迭代次数设置为 0(算法 2 中的行①). 然后随机生成参考解$z=\{z_1,z_2,\cdots,z_{N_{\text{pop}}}\}$, 并计算每一个权重向量的$T_{\text{nb}}$个邻居(算法 2 中的行②③). 接着, 随机生成种群$x^1,x^2,\ldots,x^{N_{\text{pop}}}$, 并计算每个$FV^i$(算法 2 中的行④). 在每一轮迭代中, 首先计算种群中每一个个体x^i的邻居的新解y_i, 然后对于每个邻居j, 如果$g_{N_{\text{pop}}}(y|\lambda^j,z)\leqslant g_{N_{\text{pop}}}(x^j|\lambda^i,z)$成立, 则更新$x^j=y_i, FV^j=z_j$(算法 2 中的行⑤~⑬). 对于每一个已经在帕累托集合PS中的邻居j, 首先判断z_j是否支配y_i, 如果是, 则分别从集合EP和PS中删除z_j和j(算法 2 中的行⑭~⑲). 如果在EP中没有向量支配y_i, 说明此时的邻居j为目前的最优解, 将对应的z_j和j加入集合EP和PS(算法 2 的行⑳~㉓). 迭代结束后, 算法 2 会返回对子服务的放置方案, 即迭代得到的帕累托集合PS.

3 设备移动感知的动态服务部署策略

现实生活中, 越来越多的终端设备会随着用户一边移动一边进行服务请求. 为了进一步考虑边缘设备的移动性, 本文提出了一种设备移动感知的动态服务部署策略. 首先, 为了适应移动的服务请求, 建模了服务迁移成本队列解耦动态用户的服务部署问题; 然后, 提出了一种基于响应时间优先的实时服务放置策略.

图 3 展示了用户移动情况下, 在移动边缘计算架构下一个用户移动轨迹的示例[23]. 在该架构下, 考虑网络运营商设置了N_{mes}个 MEC 节点来服务N_{user}个移动用户. 每个 MEC 节点都通过高速局域网与邻近的基站或是无线接入点相连接. 每个用户携带一个移动设备, 在移动设备上运行的应用程序的服务配置文件将分配到给定的边缘服务器的专用虚拟机[24]或容器[25]. 将系统的运行均分成若干时隙, 其时间线被离散成若干时间帧$t\in\{0,1,\cdots,T\}$. 在每个离散时隙,

每个移动用户向本地 MEC 节点发送服务请求,由软件定义网络控制器(SDN 控制器)收集所有用户的服务请求信息并根据当前全局系统确定最佳 MEC 节点为相应用户提供服务.

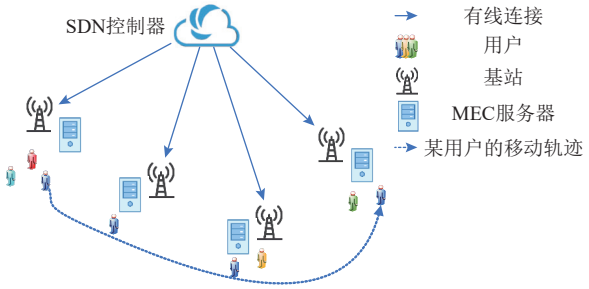

Fig. 3 An example of a user's movement trajectory under the mobile edge computing architecture

图 3 在移动边缘计算架构下用户的移动轨迹示例

用户的移动通常是不规律的,为了使用户可以获得高质量的 QoS 服务,每个用户的服务配置文件应该随着用户移动性动态迁移. 定义 $x_{i,j}(t)$ 表示在时隙 t 用户 j 的服务配置文件是否部署在 MEC 节点 i 上,如果是,则 $x_{i,j}(t)=1$,否则 $x_{i,j}(t)=0$.

在 MEC 架构下,安全关键型服务的 QoS 由通信时延、计算时延以及为服务提供安全服务的时延所决定. 假设记 $T_j^{\mathrm{QoS}}(t)$ 为时隙 t 时用户 j 的总 QoS 时延,$T_j^{\mathrm{QoS}}(t)$ 被定义为

$$T_j^{\mathrm{QoS}}(t)=\sum_{i=1}^{N_{\mathrm{mes}}}x_{i,j}(t)\left(T_{i,ES_j}^{\mathrm{wireless}}+T_{i,j}^{\mathrm{cal}}+T_{\mathrm{ss}}\right),\tag{26}$$

其中 $T_{i,ES_j}^{\mathrm{wireless}}$ 和 T_{ss} 分别由式(18)和式(12)给出,$T_{i,j}^{\mathrm{cal}}$ 被定义为

$$T_{i,j}^{\mathrm{cal}}=M_{i,j}/P_j,\tag{27}$$

其中,$M_{i,j}$ 为用户 i 在 MEC 节点 j 上的服务配置文件的大小,P_j 为节点 j 的计算能力.

使用 $Cost_{i_1,i_2}^{j}(t)$ 表示将用户 j 的服务配置文件从 MEC 节点 i_1 迁移到 MEC 节点 i_2 的开销,则在时隙 t 用户 j 的服务迁移成本 $Cost_j(t)$ 计算为

$$Cost_j(t)=\sum_{i_1}^{N_{\mathrm{mes}}}\sum_{i_2}^{N_{\mathrm{mes}}}x_{i_1,j}(t-1)\times x_{i_2,j}(t)\times Cost_{i_1,i_2}^{j}(t).\tag{28}$$

因此,在时隙 t 时所有用户的服务迁移成本 $Cost_{\mathrm{total}}(t)$ 计算为

$$Cost_{\mathrm{total}}(t)=\sum_{j=1}^{N_{\mathrm{user}}}Cost_j(t).\tag{29}$$

定义队列 $Q_{\mathrm{L}}(t)$ 表示从零时隙到时隙 t 时累计超出给定的长期平均期望预算的度量. 由式(26)(29)可得,$Q_{\mathrm{L}}(t+1)$ 可由时隙 t 时的 $Q_{\mathrm{L}}(t)$、时隙 t 时的服务迁移成本以及长期平均期望预算推导得出:

$$Q_{\mathrm{L}}(t+1)=\max\left\{Q_{\mathrm{L}}(t)+Cost_{\mathrm{total}}(t)-Cost_{\mathrm{avg}}(t),0\right\}.\tag{30}$$

为了保证迁移成本的平均值不超过长期平均期望预算,则 $Q_{\mathrm{L}}(t)$ 的长期平均值应趋向于 0. 因此,$Q_{\mathrm{L}}(t+1)\geqslant Q_{\mathrm{L}}(t)+Cost_{\mathrm{total}}(t)-Cost_{\mathrm{avg}}(t)$ 成立. 定义二次李雅普诺夫函数为

$$L(t)=\frac{1}{2}Q_{\mathrm{L}}(t)^2.\tag{31}$$

一般地,$L(t)$ 越小表示队列 $Q_{\mathrm{L}}(t)$ 的队列积压越小[26]. 因此,当二次李雅普诺夫函数保持有界时,则可推导得到队列 $Q_{\mathrm{L}}(t)$ 是稳定的. 此时假设服务部署策略为 d,并构建一条平稳分布的马尔可夫链. $p_d^{d'}(t)$ 是时隙 t 时状态 d 到状态 d' 的非负转移率,其中 $p_d^{d'}(t)$ 在满足式(32)条件时,马尔可夫链满足平稳分布[27].

$$p_d^{d'}(t)=\alpha\exp\left\{\frac{\beta}{2}\left(s(d',t)-s(d,t)\right)\right\},\tag{32}$$

其中 α 和 β 为常数.

本文所提出的设备移动感知的动态服务部署策略的伪代码如算法 3 所示.

算法 3. 设备移动感知的动态服务部署策略.

输入:服务放置策略的状态空间 $d(t)$,最大迭代轮次 $T_{\max}$;

输出:最优服务放置策略 d^*.

① 初始化服务部署策略 d,该部署策略将每个服务配置文件随机分配到 N_{mes} 个 MEC 节点上; /*初始化*/

② 迭代轮次 T=0;

③ while $T\leqslant T_{\max}$

④ 随机选择一个用户的服务配置文件;

⑤ for each $d'\in d(t), d'\neq d$

⑥ 使用式(26)计算成本函数;

⑦ end for

⑧ 根据式(29)计算得到的概率选择一种放置策略;

⑨ 通过将服务放置到新的 MEC 节点来更新服务放置策略;

⑩ 设使得 $Cost_{\mathrm{total}}(t)$ 最小的服务放置策略为 d^*;

⑪ T++;

⑫ end while

⑬ return d^*.

算法 3 以状态空间 $d(t)$ 和最大迭代轮次 $T_{\max}$ 作为输入,并输出最优服务放置策略 d^*. 算法 3 首先初始化迭代轮次为 0,并随机生成一个服务部署策略 d,该部署策略 d 将每个服务配置文件随机分配到 N_{mes} 个

MEC 节点上(算法 3 的行①). 对于每一次迭代, 首先随机选择一个用户的服务配置文件, 对于在此时服务放置策略的状态空间$d(t)$中除了d以外的每一个部署策略d', 使用式(26)计算成本函数(算法 3 的行⑤~⑦). 接着算法 3 根据式(29)计算得到的概率选择一种放置策略并通过将服务放置到新的 MEC 节点来更新现有的服务放置策略(算法 3 的行⑧~⑩). 设使得$Cost_{\text{total}}(t)$最小的服务放置策略为d^*, 该策略d^*即为最佳放置策略, d^*会在每一轮迭代中被更新(算法 3 的行⑩). 最后, 算法 3 会返回迭代结束后得到的最佳服务放置策略(算法 3 的行⑬).

4 实　　验

本文在 2 个数据集上进行实验以验证所提方案的有效性. 具体来说, 首先介绍了相关的实验设置; 然后以传输开销为评价标准来衡量所提的基于 *K*-Means 的服务解耦算法的有效性; 接着, 在多个方面将所提的基于 MOEA/D 的服务静态放置策略与 3 种基准算法相比较, 以验证所提的静态放置策略的有效性; 最后, 在多个方面将所提的设备移动感知的动态服务部署策略与 3 种基准方法相比较, 以验证所提的动态服务部署策略的有效性.

4.1 实验设置

本文的实验环境使用 Windows 10 操作系统, 8GB 内存, AMD Ryzen 5 4500U @ 2. 38GHz 处理器, 并使用 Python3. 7 的运行环境.

在数据集方面, 本文采用 2 个数据集进行实验. 第 1 个数据集是 1990 年美国人口普查公开数据集(以下简称数据集 1)[28], 该数据集包含了大约 20 万条 1990 年美国人口的普查结果, 其包含年龄等 68 种基本特征. 我们将该数据集分为 10 个服务, 并由 68 个边缘设备对这些服务中的字段进行订阅. 第 2 个数据集是由上海交易中心信息技术有限公司提供的真实数据集(以下简称数据集 2). 该数据集提供了上海交易中心信息技术有限公司真实的 20 个设备对 15 个服务对应的所有字段的订阅情况. 我们对每一个服务的耦合度进行计算, 并采用所提出的实时解耦策略对这 2 个数据集中的服务进行解耦. 删除、增加和更新这 3 个指示变量, 均设为 1. 为了比较离线算法与在线算法节省的带宽, 将头部信息的大小设置为 10b, 并假设每条数据为 200b.

在开销方面, 本文所提出的策略所带来的开销分为 2 个部分: 算法的运行时间和因传输解耦后的服务所带来的传输开销. 因此, 本文在 4.2~4.4 节对这 2 个方面的开销进行了详细的分析和对比.

4.2 基于 *K*-Means 的服务实时解耦策略验证

为了验证基于 *K*-Means 的服务实时解耦策略, 本文对比了解耦前后的传输开销. 数据集 1 和数据集 2 的解耦前后传输开销分别如图 4 和图 5 所示. 从图 4 可以看出, 编号 1 在解耦后的所有服务的通信开销平均减少了 29.04%; 对编号 3 的服务解耦效果最为明显, 其通信开销减少了 44.20%. 从图 5 可以看出, 与解耦前相比, 所有服务在解耦后的传输开销平均减少了 24.71%, 其中传输开销减少可高达 29.92%. 这是因为解耦后, 边缘服务器减少了大量无关子服务的订阅数, 因此减少了大量不必要的通信开销. 只有少数服务(如编号 8 的服务)在解耦前后通信开销变化不大. 这是因为这些服务解耦后边缘服务器仍然订阅了大量的子服务. 如果某服务订阅了解耦后服务集合的所有子服务, 会由于解耦增加的额外成本导致解耦后的字段开销大于解耦前的字段开销. 在实际应用中, 可以考虑不对该服务进行解耦.

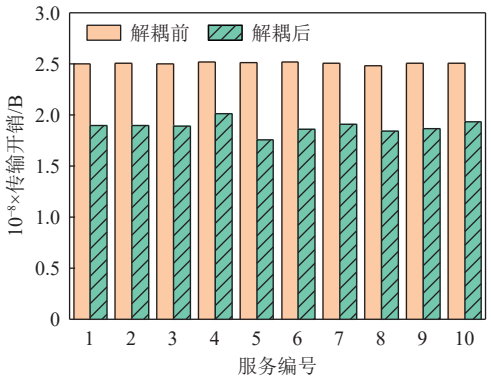

Fig. 4　Comparison of the transmission overhead for dataset 1 before and after decoupling

图 4　对比数据集 1 解耦前后的传输开销

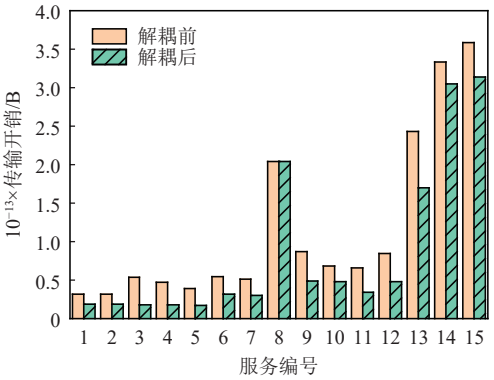

Fig. 5　Comparison of transmission overhead for dataset 2 before and after decoupling

图 5　对比数据集 2 解耦前后的传输开销

图 6 对比了在数据集 2 上增加字段时 ISODATA[22] 和所提出的基于 *K*-Means 的实时解耦策略的运行时间. 在不改变划分结果的条件下, 在线算法的运行时间远小于离线算法的运行时间. 在数据集 2 的 10 组数据中, 在线算法的算法运行时间平均减少了 78.14%, 最多减少高达 80.35%. 这是因为在线算法只需要遍历 1 遍数据即可完成对整个服务的划分, 而离线算法通常要迭代很多轮才能收敛得到最终的划分结果.

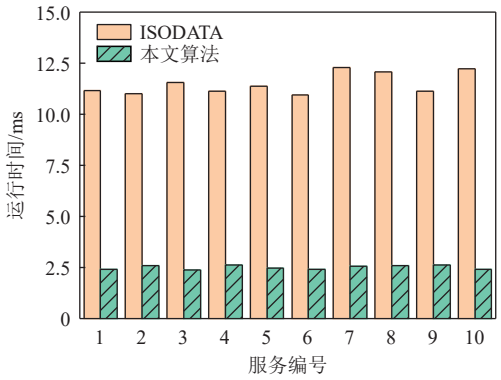

Fig. 6 Computation time of ISODATA and the proposed strategy when conducting add operation on dataset 2

图 6 对比在数据集 2 上增加字段时 ISODATA 和所提策略的运行时间

图 7 展示了字段个数分别为 2 万、4 万、8 万以及 12 万时 ISODATA[22] 和所提出的基于 *K*-Means 的实时解耦策略的运行时间. 相比于 ISODATA 算法, 4 种不同字段个数的在线算法的算法运行时间分别减少了 78.95%, 85.37%, 86.06%, 85.35%. 由于算法的时间复杂度与字段个数呈正相关, 字段个数越大, 在线算法节省的时间也就越多. 因此, 我们认为, 所提出的基于 *K*-Means 的实时解耦策略比较适合服务体量较大也就是字段个数较大的情况. 在这种情况下, 所提算法能够节约的处理时间较多, 效果更好. 但是当服务较小或子服务耦合度较低时, 所提方法反而可能会导致较大的开销. 这是因为所提方法需要传输解耦后的服务, 而这会带来较大的传输延迟.

图 8 展示了数据集 2 在删除字段的场景 ISODATA 算法和所提的基于 *K*-Means 的实时解耦策略的运行时间比较. 在解耦结果一致的条件下, 所提实时解耦算法的运行时间远小于 ISODATA 算法的运行时间. 具体来说, 在数据集 2 上所提出的实时解耦算法的运行时间平均减少了 73.02%, 最多减少高达 75.80%.

4.3 基于 MOEA/D 的服务静态放置策略验证

为了验证所提的基于 MOEA/D 的集中式服务静态放置算法的有效性, 本实验比较并分析了基于 MOEA/D 策略(本文算法)和 3 种基准算法的集中式服务静态放置结果, 其中 3 种基准算法分别是基于随机的算法(RAND)、遗传算法(GA)[29] 和粒子群算法(PSO)[30]. 这 3 种基准算法的原理为:

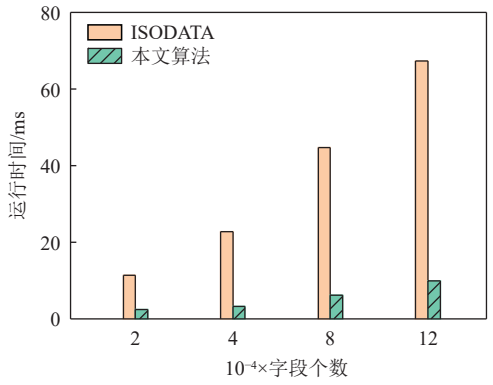

Fig. 7 Computation time of ISODATA and the proposed strategy under different numbers of adding segments

图 7 当增加字段量不同时 ISODATA 和所提策略的运行时间

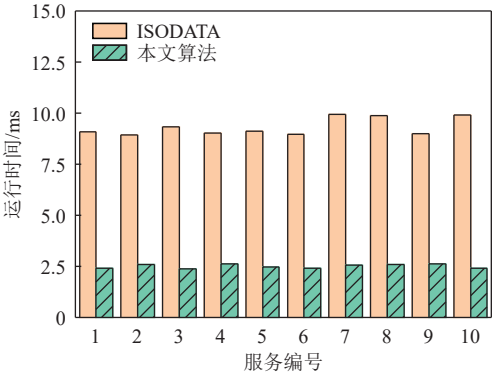

Fig. 8 Computation time of ISODATA and the proposed strategy when conducting delete operation on dataset 2

图 8 对比在数据集 2 上删除字段时, ISODATA 和所提策略的运行时间

1) RAND. 随机选取每个服务部署的边缘节点放置服务.

2) GA. 遗传算法基于染色体自然进化, 对染色体编码后, 使用交叉和变异生成新的染色体, 并基于适应度函数评估染色体的质量, 最后根据最优染色体得到最优解以放置服务.

3) PSO. 粒子群优化是一种使用粒子模拟一群寻找食物的鸟的算法. 具有“速度”和“位置”的粒子是目标函数空间中的搜索个体, 算法会基于所有粒子的历史最优位置调整速度. 最后将所有粒子当前位置的最优解视为全局最优解.

图 9 对比了当设备数量分别为 3, 5, 10 时, 使用

本文算法、PSO、GA、RAND 得到的平均服务响应时间. 服务响应时间是由本文算法运行时间和终端设备请求服务的响应时间组成，终端设备的响应时间由服务请求的通信时延、服务加密消耗的时间以及服务在边缘服务器运行所需的时间组成. 由图 9 可以看出，与 PSO, GA, RAND 这 3 种基准算法相比，本文算法分别平均优化了 9.23%, 10.33%, 15.07%，最高优化分别为 24.89%, 25.94%, 35.55%. 这是因为本文算法的运行时间以及服务放置策略均优于这 3 种基准算法.

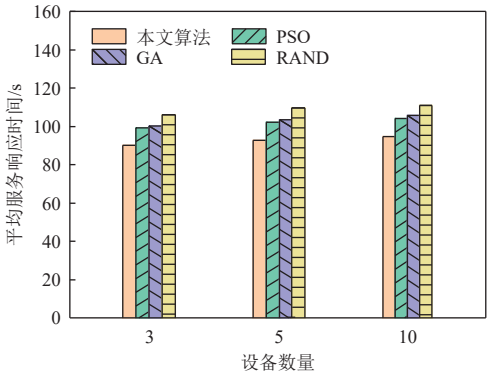

Fig. 9 Average service response time of the 4 algorithms under different numbers of devices

图 9 不同设备数量下对比 4 种算法的平均服务响应时间

图 10 展示了当设备数量分别为 3, 5, 10 时，使用本文算法、PSO、GA、RAND 算法得到的负载均衡平均优化率. 将本文算法的负载均衡平均优化率设置为 1，与 PSO, GA, RAND 这 3 种基准算法相比，本文算法分别平均优化了 3.28%, 29.15%, 66.97%，最高优化分别为 89.67%, 87.91%, 96.10%. 虽然当边缘服务器数量为 5 时，PSO 算法的负载均衡平均优化率小于 1，然而此时其响应时间比本文算法高 10.18%. 由

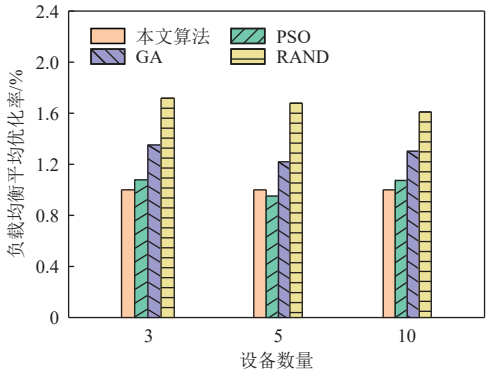

Fig. 10 Load balancing average optimization rate of the 4 algorithms under different numbers of devices

图 10 不同设备数量下对比 4 种算法的负载均衡平均优化率

于 PSO 和 GA 难以找到优化问题的全局最优解，因此使用 PSO 和 GA 算法得到的负载均衡平均优化率不如本文算法. 因此，由图 10 可以看出，本文所提的基于 MOEA/D 的服务静态放置策略适合被应用于设备数量较多、需要进行负载均衡的场景中. 此时，本文算法相较于基准算法，能够取得更好的优化效果.

4.4 设备移动感知的动态服务部署策略验证

为了验证所提设备移动感知的动态服务部署策略的有效性，本文所提方案与“总是迁移”(always migration, AM)、“从不迁移”(no migration, NM)以及“随机迁移 K 个服务”(random K-service migration, RKM)三种基准方法进行了对比. 这 3 种方法的原理简介如下：

1) AM. 在每个时隙中，用户的服务配置文件总是迁移到距离用户最近的 MEC 节点上.

2) NM. 在每个时隙中，用户的服务配置文件总保持在初始的 MEC 节点不迁移.

3) RKM. 在每个时隙中，随机选取 K 个服务配置文件迁移到当前最优的 MEC 节点上.

本节使用 2007 年由芬兰国家土地调查局提供的赫尔辛基市区的街道进行仿真[31]. 假设终端设备在该市区街道上进行移动，我们将该市区街道划分成 64 个正方形区域，每块大小 500 m×500 m，在每块区域的中心位置部署 MEC 服务器提供移动服务，其中每台 MEC 服务器具有多个 CPU 核，其计算频率为 25 GHz，存储空间为 100~160 GB，2 个 MEC 节点的距离使用曼哈顿距离. 假设街道上有 2 种类型的移动用户：一种是行人，速度在 0. 5~1.5 m/s；一种是车载用户，速度在 5.56~11.11 m/s，假设行人的数量占总人数的比率是 7/8. 本实验仿真 100 个时隙，每个时隙是 5 min，在每个时隙 t 内，假设用户的服务配置文件所在的 MEC 节点保持不变，响应用户所需要的服务大小均匀分布在 0.8~1 GB.

图 11 展示了 AM, NM, RKM 和响应时间优先的算法(本文算法)得到的总时延，该总时延为服务的通信时延、加密时延以及计算时延之和. RKM 算法中选取的配置文件个数 K 取总人数的 10%. 与 AM, NM, RKM 这 3 种基准算法相比，本文算法的总时延分别平均减少了 8.52%, 10.64%, 2.78%，最多减少可分别达 10.78%, 12.70%, 3.74%. 这是由于 NM 算法从不迁移服务配置文件，因此其服务总时延基本与初始化服务配置文件放置结果有关；而 AM 算法虽然一直将服务配置文件迁移到距离用户最近的位置，然而当一个局部地区的人数过多时，迁移到同一

MEC 服务器上会给服务器的计算带来较大的压力，每个服务分得的资源也会减少，甚至会出现资源不够的情况；RKM 算法每轮随机选取 K 个服务配置文件进行迁移，然而当 K 过小时，其性能会接近 NM 算法，且可能会导致无法优化部分服务配置文件的放置，而当 K 过大时会接近 AM 算法，同样可能出现资源分配不当的问题.

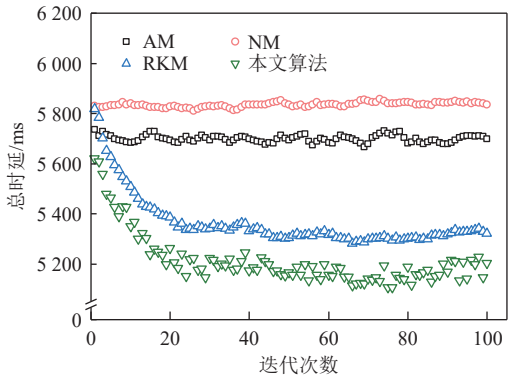

Fig. 11 Comparison of the total latency of four deployment methods

图 11 对比 4 种部署方案的总时延

图 12 展示了 AM, NM, RKM 和响应时间优先的算法(本文算法)的服务迁移成本，其中平均期望预算被设定为 200. 从图 12 中可以看出，AM 算法由于频繁地进行服务配置文件的迁移，其服务迁移成本远超过长期平均期望预算；虽然 NM 算法不进行服务配置文件的迁移，但其服务的总时延过高，且与初始化的结果有很大的关系，通常会导致服务质量不高；而 RKM 算法没有充分利用服务迁移预算，使其服务迁移成本一直保持在较低的水平；而本文算法其服务迁移成本保持在长期平均期望预算的迁移成本之内的条件下，能够取得最优的服务总时延.

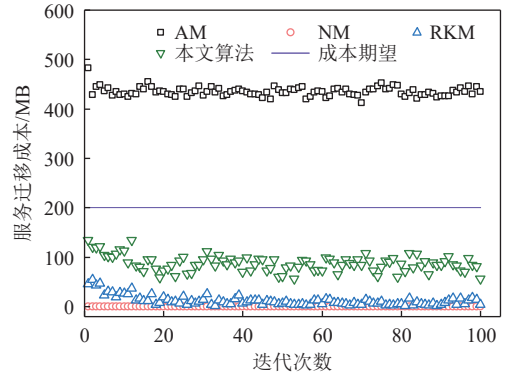

Fig. 12 Comparison of the migration costs of four deployment methods

图 12 对比 4 种部署方案的迁移成本

5 结 论

本文提出了一种 2 阶段的服务质量优化方案以解决移动边缘计算架构下的服务安全部署问题. 在第 1 阶段，本文考虑了服务的安全性和边缘服务器的负载均衡问题，对服务部署问题进行建模，提出了一种基于 K-Means 的服务实时解耦方案，并设计了一种基于 MOEA/D 的服务部署策略；在第 2 阶段，本文考虑了边缘设备的移动性，设计了一种设备移动感知的服务部署策略，优化了移动边缘计算架构下的服务质量. 实验结果表明，与多种基准方法相比，在保证系统安全以及服务器负载均衡的前提下，本文所提出的面向集中式服务部署策略能够使服务请求的响应时间降低 36%，所提出的动态部署策略能进一步降低 13% 的服务响应时间.

作者贡献声明：李丽颖提出了算法思路并撰写论文；张润泽负责完成实验；魏同权提出指导意见并修改论文.

参 考 文 献

[1] Shi Weisong, Sun Hui, Cao Jie, et al. Edge computing—An emerging computing model for the Internet of everything era[J]. Journal of Computer Research and Development, 2017, 54(5): 907−924 (in Chinese)
(施巍松, 孙辉, 曹杰, 等. 边缘计算：万物互联时代新型计算模型[J]. 计算机研究与发展, 2017, 54(5): 907−924)

[2] Cleber S, Leandro A, Flávia D, et al. Increasing the availability of IoT applications with reactive microservices[J]. Service Oriented Computing and Applications, 2021, 15(2): 109−126

[3] Qian Zhihong, Wang Yijun. IoT technology and application[J]. Acta Electronica Sinica, 2012, 40(5): 1023−1029 (in Chinese)
(钱志鸿, 王义君. 物联网技术与应用研究[J]. 电子学报, 2012, 40(5): 1023−1029)

[4] Xiao Yang, Li Haizhong, Li Bo. Bandwidth sharing schemes for multimedia traffic in the IEEE 802.11e contention-based WLANs[J]. IEEE Transactions on Mobile Computing, 2007, 6(7): 815−831

[5] Auer F, Lenarduzzi V, Felderer M, et al. From monolithic systems to Microservices: An assessment framework[J]. Information and Software Technology, 2021, 137: 106600

[6] Arisholm E, Briand L C, Foyen A. Dynamic coupling measurement for object-oriented software[J]. IEEE Transactions on Software Engineering, 2004, 30(8): 491−506

[7] Poshyvanyk D, Marcus A, Ferenc R, et al. Using information retrieval based coupling measures for impact analysis[J]. Empirical Software Engineering, 2009, 14(1): 5−32

[8] Wang Xiaofei, Han Yiwen, Leung V C M, et al. Convergence of edge computing and deep learning: A comprehensive survey[J]. IEEE Communications Surveys & Tutorials, 2020, 22(2): 869−904

[9] Shi Weisong, Cao Jie, Zhang Quan, et al. Edge computing: Vision and challenges[J]. IEEE Internet of Things Journal, 2016, 3(5): 637−646

[10] Yin Hao, Zhang Xu, Liu Hongqiang Harry, et al. Edge provisioning with flexible server placement[J]. IEEE Transactions on Parallel and Distributed Systems, 2016, 28(4): 1031−1045

[11] Farhadi V, Mehmeti F, He Ting, et al. Service placement and request scheduling for data-intensive applications in edge clouds[J]. IEEE/ACM Transactions on Networking, 2021, 29(2): 779−792

[12] Dong Yunmeng, Xu Gaochao, Ding Yan, et al. A 'joint-me' task deployment strategy for load balancing in edge computing[J]. IEEE Access, 2019, 7: 99658−99669

[13] Fan Qi, Li Zhuo, Chen Xin. Inference delay optimization of branchy neural network model based on edge computing[J]. Journal of Computer Applications, 2020, 40(2): 342−346 (in Chinese)
(樊琦, 李卓, 陈昕. 基于边缘计算的分支神经网络模型推断延迟优化[J]. 计算机应用, 2020, 40(2): 342−346)

[14] Ahmed E, Rehmani M H. Mobile edge computing: Opportunities, solutions, and challenges[J]. Future Generation Computer Systems, 2017, 70: 59−63

[15] Chen Xu, Shi Qian, Yang Lei, et al. ThriftyEdge: Resource-efficient edge computing for intelligent IoT applications[J]. IEEE Network, 2018, 32(1): 61−65

[16] Deng Tao, You Lei, Fan Pingzhi, et al. Device caching for network offloading: Delay minimization with presence of user mobility[J]. IEEE Wireless Communications Letters, 2018, 7(4): 558−561

[17] Wang Shiqiang, Urgaonkar R, Zafer M, et al. Dynamic service migration in mobile edge-clouds[C] //Proc of IFIP Networking Conf. Piscataway, NJ: IEEE, 2015: 1−9

[18] Taleb T, Ksentini A. An analytical model for follow me cloud[C]// Proc of the 2013 IEEE Global Communications Conf (GLOBECOM). Piscataway, NJ: IEEE, 2013: 1291−1296

[19] Cao Kun, Li Liying, Cui Yangguang, et al. Exploring placement of heterogeneous edge servers for response time minimization in mobile edge-cloud computing[J]. IEEE Transactions on Industrial Informatics, 2020, 17(1): 494−503

[20] Jiang Wei, Jiang Ke, Zhang Xia, et al. Energy aware real-time scheduling policy with guaranteed security protection[C]// Proc of the 19th Asia and South Pacific Design Automation Conf (ASP-DAC). Piscataway, NJ: IEEE, 2014: 317−322

[21] Wang Shangguang, Guo Yan, Zhang Ning, et al. Delay-aware microservice coordination in mobile edge computing: A reinforcement learning approach[J]. IEEE Transactions on Mobile Computing, 2019, 20(3): 939−951

[22] Wei Xiaohui, Li Zijian, Liu Yuanyuan, et al. SDLSC-TA: Subarea division learning based task allocation in sparse mobile crowdsensing[J]. IEEE Transactions on Emerging Topics in Computing, 2020, 9(3): 1344−1358

[23] Ouyang Tao, Zhou Zhi, Chen Xu. Follow me at the edge: Mobility-aware dynamic service placement for mobile edge computing[J]. IEEE Journal on Selected Areas in Communications, 2018, 36(10): 2333−2345

[24] Chaufournier L, Sharma P, Le F, et al. Fast transparent virtual machine migration in distributed edge clouds[C/OL]//Proc of the 2nd ACM/IEEE Symp on Edge Computing. Piscataway, NJ: IEEE, 2017 [2023-03-21].https://dl.acm.org/doi/10.1145/3132211.3134445

[25] Ma Lele, Yi Shanhe, Li Qun. Efficient service handoff across edge servers via docker container migration[C/OL]//Proc of the 2nd ACM/IEEE Symp on Edge Computing. Piscataway, NJ: IEEE, 2017 [2023-03-23].https://dl.acm.org/doi/10.1145/3132211.3134460

[26] Pierdomenico P. Converse Lyapunov theorems for discrete-time switching systems with given switches digraphs[J]. IEEE Transactions on Automatic Control, 2019, 64(6): 2502−2508

[27] Chen Minghua, Liew C S, Shao Ziyu, et al. Markov approximation for combinatorial network optimization[J]. IEEE Transactions on Information Theory, 2013, 59(10): 6301−6327

[28] Meek C, Thiesson B, Heckerman D. The learning curve method applied to clustering[C] // Proc of the 8th Int Workshop on Artificial Intelligence and Statistics. New York: PMLR, 2001: 196−202

[29] Zhao Chenhong, Zhang Shanhan, Liu Qingfeng, et al. Independent tasks scheduling based on genetic algorithm in cloud computing[C]//Proc of the 5th Int Conf on Wireless Communications, Networking and Mobile Computing. Piscataway, NJ: IEEE, 2009: 1−4

[30] Kennedy J, Eberhart R. Particle swarm optimization[C]//Proc of the Int Conf on Neural Networks. Piscataway, NJ: IEEE, 1995: 1942−1948

[31] Radenkovic M, Grundy A. Efficient and adaptive congestion control for heterogeneous delay-tolerant networks[J]. Ad Hoc Networks, 2012, 10(7): 1322−1345

Li Liying, born in 1995. PhD, lecturer. Her main research interests include Internet of things, mobile edge computing, and data analysis.

李丽颖, 1995 年生. 博士, 讲师. 主要研究方向为物联网、移动边缘计算和数据分析.

Zhang Runze, born in 1997. master. His main research interests include mobile edge computing and cloud computing

张润泽, 1997 年生. 硕士. 主要研究方向为移动边缘计算和云计算.

Wei Tongquan, born in 1973. PhD, associate professor. His main research interests include Internet of things, mobile edge computing, and cloud computing.

魏同权, 1973 年生. 博士, 副教授. 主要研究方向为物联网、移动边缘计算和云计算.