边缘环境下面向移动性用户的微服务选择方法

赵庶旭，蒋恺俊，王小龙
（兰州交通大学电子与信息工程学院，甘肃 兰州 730071）

摘　要：为解决移动边缘计算中终端用户频繁移动导致服务难以连续的问题，提出结合云边加速比和边缘效益指数（EGI）的微服务选择与部署策略。首先，建立了基于容器与工作流的微服务选择架构，提出了融合服务质量评价指标的迁移负载优化模型。其次，设计了基于EGI的微服务部署算法，实现低时延、高命中率的微服务部署方案。最后，提出了基于云边加速比模型和动态云边迁移策略的微服务选择算法，解决用户移动中的跨域迁移问题。仿真结果表明，所提方法相比于传统方法在服务质量上提升约23.2%，在能耗和时延上分别降低约25.4%和25.1%。

关键词：边缘计算；微服务选择；容器调度；服务部署；工作流
中图分类号：TN92
文献标志码：A
DOI：10.11959/j.issn.1000−436x.2025080

Microservice selection approach for mobile users in edge computing environment

ZHAO Shuxu, JIANG Kaijun, WANG Xiaolong
School for Electronic and Information Engineering, Lanzhou Jiaotong University, Lanzhou 730071, China

Abstract: To address the problem of service discontinuity caused by frequent user mobility in mobile edge computing, a microservice selection and deployment strategy was developed by integrating the cloud-edge speedup ratio and the edge gain index (EGI). Firstly, a microservice selection architecture was established based on containers and workflows, and a migration load optimization model incorporating service quality evaluation metrics was formulated. Then, a microservice deployment algorithm based on EGI was designed to achieve low-latency and high-hit-rate deployment. Furthermore, a microservice selection algorithm was proposed based on the cloud-edge speedup model and a dynamic cloud-edge migration strategy to handle cross-domain migration under user mobility. Simulation results demonstrate that the proposed approach improves service quality by approximately 23.2%, while reducing energy consumption and latency by 25.4% and 25.1%, respectively, compared to traditional methods.

Keywords: edge computing, Microservice selection, container scheduling, service deployment, workflow

0　引言

5G、移动物联网等在智慧城市中的大规模布置与应用，催生城市计算向边缘延伸，在公共交通、智能安防、智能环境等领域带来了深刻变革。作为一种分布式的数据处理和存储架构，移动边缘计算（MEC, mobile edge computing）[1]能够在更接近用户和数据源的位置进行数据分析与处理，从而为用户提供更低的时延和带宽成本。然而，边缘服务器在处理、存储和通信等方面的资源有限，显著制约了其处理具有高度并发性且存在复杂依赖关系

收稿日期：2025−01−17；**修回日期：**2025−04−21
基金项目：甘肃省教育厅高校教师创新基金资助项目（No.2024B-059）；兰州交通大学青年科学基金资助项目（No.2024007）
Foundation Items：Gansu Provincial Department of Education University Teacher Innovation Fund Project (No.2024B-059), Youth Science Foundation of Lanzhou Jiaotong University (No.2024007)

的物联网（IoT, Internet of things）应用的能力，难以充分满足高效、可靠运行的需求[2]。

为了解决这一问题，微服务架构应运而生。微服务架构将传统单体应用分解为更细粒度且松耦合的微服务集合，每个微服务能够专注于实现某一特定功能，并通过组合形成工作流以满足复杂的业务需求[3-4]。考虑到应用程序的多样性和单个边缘服务器资源的有限性，结合微服务的轻量化、易部署、易迁移和相对独立等特点，在资源受限的边缘服务器上部署基于微服务架构划分的应用程序，将有助于每个服务独立地部署、更新和拓展，从而避免了传统单体应用架构中“牵一发而动全身”的问题。这有助于解决边缘计算中资源分配不均、服务响应时延长以及系统维护复杂等挑战。通过在边缘服务器和云服务器之间合理部署、调度微服务，可以实现云边、边边多个异构服务器之间的协同工作，确保应用程序的高性能和高可靠性，同时提高资源利用率。

在移动边缘计算环境中，用户的移动性是一个不可忽视的关键因素，显著限制了服务质量的提升[5]。用户在移动过程中，信道状态的不稳定性可能会对原有任务卸载方案的性能产生负面影响。为了确保服务质量（QoS, quality of service）和服务的连续性，跟随用户进行任务迁移成为一种有效的解决方案。然而，大量用户的动态加入或离开所引发的任务迁移，不仅导致服务器负载的剧烈波动，还打破了原有负载分布的均衡状态。这种非均衡的负载分布直接导致特定服务器上的任务处理压力增大，任务排队时延显著增加，进而限制了系统资源的有效利用，并降低了整体吞吐率[6]。因此，如何在保证用户服务连续性的同时做出有效的微服务选择与迁移决策，实现系统负载均衡，成为移动边缘计算任务迁移中亟待解决的重要问题。

基于上述问题，本文主要研究工作及贡献如下。

1) 针对传统微服务部署方式中因静态部署无法适应边缘环境而导致的边缘资源浪费、时延开销大等问题，设计了基于云边加速比的微服务部署策略，通过量化边缘节点与云端之间的性能差异，结合实时负载和用户位置信息选择最优部署位置，从而降低任务时延与能耗，提升用户体验。

2) 针对移动边缘计算中终端用户频繁移动导致的服务中断和负载不均问题，设计了基于边缘效益指数（EGI, edge gain index）的微服务选择算法，通过动态云边迁移策略，实时调整微服务部署，确保用户无缝获取低时延服务。同时，针对高并发场景，设计了基于优先级的分层调度机制，结合微服务依赖关系构建调度队列，从而最大化系统并行处理能力，减少响应时延。

3) 根据边缘计算场景的特点，结合微服务之间的依赖关系与云边混合场景，构建了云边协同的微服务选择架构，并基于此建立了时延模型、能耗模型与服务质量评价模型，旨在提升服务质量的同时最大化利用边缘资源。在真实轨迹数据集上的仿真实验表明，与现有算法相比，本文微服务选择与调度算法不仅能够有效减少任务迁移次数，也能提高用户服务的稳定性和连续性。

1　相关工作

1) 微服务架构的优势与挑战

随着应用规模和业务数量的持续增长，单体架构因整体扩展局限导致资源利用率低下，难以实现精准扩容[7]。为解决分布式系统的扩展困境，微服务架构通过业务解耦和轻量化通信重构系统拓扑[8]，其模块化特征不仅支持独立部署与扩展，也通过服务粒度的精准控制显著提升资源利用率。容器化技术的引入进一步提升了服务编排的灵活性，使复杂业务流的动态组合成为可能。针对微服务架构的模块化、灵活性和可扩展性等核心特征，学术界围绕其在不同场景下的技术适配性展开深入探索。相关研究在验证架构优势的同时，也揭示了服务依赖管理、资源调度优化等新型挑战。文献[9]发现，在集群规模增加时，微服务请求的工作流中“短板效应”将会被放大，对应用程序的尾时延影响较大。文献[10]聚焦在重构单体应用时面临的微服务边界定义问题，并提出根据开发和部署单元来定义微服务边界。文献[11]设计了基于微服务架构的边缘计算资源管理方案，通过微服务的模块化优势和灵活性，优化了边缘计算中的资源分配与调度效率。文献[12]系统研究了单体系统向微服务架构的迁移路径，提出基于多维相似度度量的模块化拆解框架，重点解决功能重构挑战与架构迁移可行性问题。文献[13]构建了面向多实例微服务拓扑的部署优化模型，通过任务–资源协同调度算法实现服务依赖解析与部署效率的联合优化。

2) 移动边缘计算中的服务迁移

服务迁移策略的提出，旨在通过精确判断用户移动时的迁移时机、方式和目标位置，以保障服务的无缝连接和低时延[14]。针对用户移动中的迁移策略问题，文献[15]提出了一种基于Lyapunov优化的在线服务迁移算法，研究了移动边缘计算网络中服务部署和请求路由的联合优化问题，以平衡服务迁移成本和系统性能，最大化长期网络效用。但并未考虑复杂工作流中微服务之间的依赖关系，以及服务器集群中的负载均衡。文献[16]以边缘场景下的终端用户在服务范围内的停留时间服从均匀分布为假设，设计了基于区块链技术的移动感知卸载策略，该策略能够在确保用户隐私安全的同时最小化成本。然而，该策略并未考虑任务之间的依赖关系，且在并发量较大时会产生大量的数据库开销。文献[17]考虑了云边混合环境中用户移动时的卸载问题，将迁移成本作为重要的决策依据，提出了基于概率性能感知和演化博弈算法的卸载策略。但该方案在工作中并未考虑用户的实际运动轨迹以及服务器资源的实时使用情况，存在一定的局限性。文献[18]以用户覆盖率最大化和重分配率最低为目标，将资源分配视为一个可迭代的在线进化过程，设计了一种移动感知的分配算法。文献[19]提出了一种面向移动性用户的服务迁移策略，该方法通过马尔可夫近似和分布式更新技术平衡服务性能与迁移成本，在大规模应用场景中有良好的表现。然而，该方法未充分考虑微服务间的依赖关系和服务器的实时状态。尽管上述方法在求解微服务迁移问题时都取得了一定成效，但是随着用户的跨域移动，节点状态会随着并发访问程度的增加而动态改变，因此，需要实时考虑多方面因素进行调整以适应动态性。

3) 边缘环境下的微服务选择策略

随着用户需求的日益复杂，单一功能的服务已经很难满足用户的需求，因此采用服务组合技术将多个分布式微服务组合成一个复杂的应用程序，并在边缘集群中部署多个容器实例处理用户的并发请求成为一种有效的方案。由于边缘节点通常分布在不同的地理位置[20]，容器实例之间在通信时可能会产生较长的时延和较大的网络消耗，因此在调度时为并发请求的每个子任务选择最佳的微服务实例对于满足请求的QoS至关重要[21]。目前，国内外对微服务选择问题主要聚焦于资源受限的边缘环境下如何从每个子任务的候选服务集合中选择最佳的微服务。文献[22]针对边缘环境中服务器资源受限、用户位置变化导致微服务需求复杂的问题，以响应时间为优化目标，提出了一种基于交叉变异算子的布谷鸟优化算法以解决MEC环境中的微服务选择问题。但是该方法收敛速度较慢，不适用于高并发组合优化场景。文献[23]构建了物联网环境下基于QoS的云边缘服务发现与选择模型，并提出一种基于服务质量感知的服务选择算法GWO-GA，以提高经济和能耗效益为目标获取最佳服务组合方案。但该方案并未考虑动态用户需求以及服务器集群的实时负载。文献[24]从减少平均时延和网络消耗出发，提出了一种基于微服务优先级和改进蚁群的微服务选择策略。作者结合工作流机制和优先级调度，基于时延和网络优化蚁群算法的搜索过程，加速收敛至最优解，但是忽略了可以通过云边协同的方式利用云服务器资源，使得资源利用最大化。文献[25]研究了基于服务质量感知和微服务负载均衡的调度与选择问题，利用图来描述微服务之间的负载依赖关系，为QoS感知问题提出了一种完全多项式时间的近似方案。

边缘计算位于用户侧，网络环境较为复杂。在边缘环境中进行微服务选择以保障用户请求的QoS是当前研究的热点问题。上述文献都试图在资源受限的边缘节点上寻找部署和选择微服务的最佳方式，但是都存在着算法收敛速度慢、资源利用率低下等不足，显著影响了系统的响应速度和整体性能。此外，上述工作主要关注云数据中心或大型边缘服务器集群环境中基于微服务的物联网应用，但面对5G多层边缘计算这一新兴架构中云边协同的复杂应用场景，微服务的选择问题无疑变得更加错综复杂。这一变化不仅增加了技术实施的难度，也要求在设计与优化调度策略时必须充分考虑云资源与边缘节点之间的紧密协同与高效互动。

2 问题描述

应用程序被分解为由相互依赖的微服务组成的工作流，并依据微服务之间的依赖关系构建分层调度队列。边缘环境下基于工作流的微服务选择架构如图1所示，展示了如何将工作流中的微服务根据其依赖关系分成不同的层次，每一层的微服务在同一调度队列中并行执行，以最大化系统的并行能力，

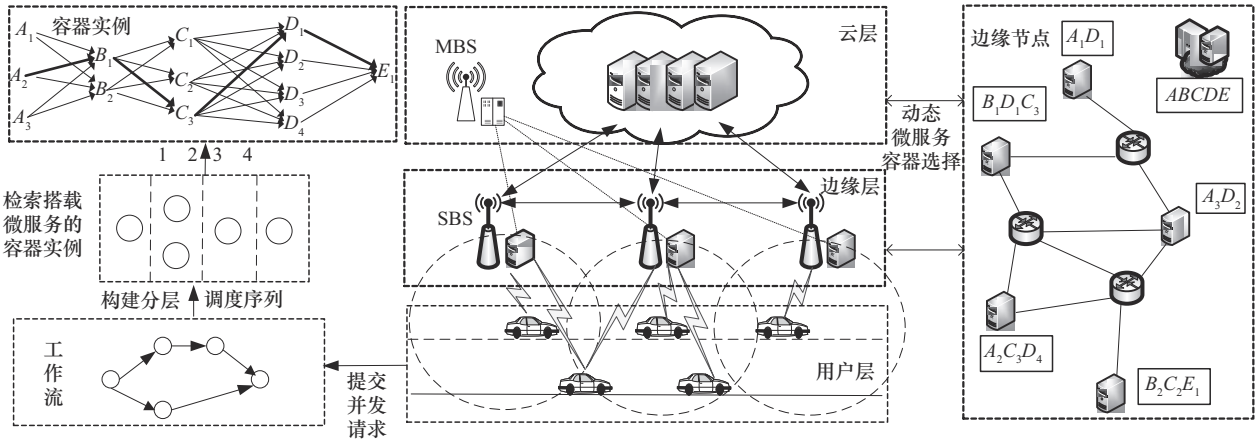


图1　边缘环境下基于工作流的微服务选择架构

从而提升整体的响应速度。为确保单个边缘节点故障不会导致服务中断，对于同一微服务的并发请求可以通过争用容器实例的方式处理，其中每个容器实例在同一时间仅响应一个用户的请求，而其余指向该实例的请求则会在该节点的请求队列中顺序等待。以图1为例，工作流由[A,B,C,D,E]5个微服务组成，这5个微服务分别在不同的边缘节点创建了多个实例，当用户对应用发起请求时，根据边缘服务器集群当前的负载与时延情况，通过决策机制为该请求包含的微服务选择[A_2,B_1,C_3,D_1,E_1]形成完整的执行路径。

为了保证所有用户至少得到一台云服务器所提供的计算服务，本文考虑典型的云边混合场景。假设中心云服务器和边缘云服务器只能为其覆盖范围内的用户提供服务，用户只能在满足距离约束的条件下，才能与服务器建立连接。在云边混合场景中，异构网络由一个宏基站（MBS, macro base station）和若干微基站（SBS, small base station）组成。中心云服务器部署在MBS上，利用其较大的覆盖范围为终端用户提供服务，以保证区域内的用户至少可以将任务通过一台云服务器进行处理。MEC服务器部署在各个SBS上并通过蜂窝基站实现分布式部署，为邻近的终端用户提供服务。尽管单个MEC的覆盖范围较小，计算能力较弱，但是由于离用户近、能耗小、租金低廉，可以为服务范围内的用户提供更低成本、更短时延的服务。然而，用户跨域移动可能会对服务的连续性带来挑战，如图1用户层所示，当用户穿越不同的覆盖区域时，原本提供服务的边缘节点或云节点可能不再有效，需要在新的区域重新选择部署所需微服务的边缘节点。这种跨域切换可能会导致额外的时延和计算资源重新分配，甚至在某些情况下，任务处理可能会暂时中断。因此，需要通过优先级部署和微服务迁移等策略，结合用户的实时位置，灵活调整任务调度，确保在跨域移动过程中能够顺利切换至最优的服务节点，从而避免服务中断和不必要的时延。

2.1　问题建模

2.1.1　系统模型

本文将由微服务工作流组成的应用 $A=\left(V^a,E^a\right)$ 描述为一个有向无环图[26]，其中 $V=\left\{V_1^a,V_2^a,\cdots,V_m^a\right\}$ 表示组成应用 A 的微服务集合，$V_i^a=<\text{data}_{i,a}^{\text{in}},\text{data}_{i,a}^{\text{out}}>$，$\text{data}_{i,a}^{\text{in}}$ 和 $\text{data}_{i,a}^{\text{out}}$ 表示微服务 V_i^a 的输入数据量和输出数据量；$E^a=\left\{\left(V_i^a,V_j^a\right)|V_i^a,V_j^a\in V^a\right\}$ 表示应用 A 中的依赖关系。为了简化，利用矩阵 $\left[\mathbf{Dep}_j^i\right]_{m\times m}$ 表示微服务 V_i^a 和 V_j^a 是否有依赖关系，若 $\mathbf{Dep}_j^i=1$，表示 V_i^a 是 V_j^a 的直接前驱，否则 $\mathbf{Dep}_j^i=0$。微服务 V_i^a 的第 b 个容器实例定义为 $I\left(a,i,b\right)$。

考虑一个由 N 个边缘服务器和一个云服务器组成的云边混合场景，其中异构的服务器可以抽象为具有不同资源容量的服务器节点 $\text{EN}=\left\{\text{EN}_0,\text{EN}_1,\text{EN}_2,\cdots,\text{EN}_N\right\}$，其中 EN_0 代表云服务器。$\mathcal{K}=\{1,2,\cdots,K\}$ 表示用户集合。假设用户的时间轴被离散为时间片 $t\in\mathcal{T}=\{1,2,\cdots,T\}$。在每个离散时隙内的微服务调度问题可以抽象为用户在准静止状态下做出的调度决策。在时隙 t 内，用户 k 与各服务器节点之间的距离为 $l_t^k=\left[l_{(t,\theta)}^k,l_{(t,1)}^k,l_{(t,2)}^k,\cdots,l_{(t,N)}^k\right]$，服务器 n

的覆盖范围用c_n表示，用二元变量$\rho_{k,t}^n$表示在时隙t内，用户k是否在边缘服务器n的覆盖范围内，即

$$\rho_{k,t}^n=\begin{cases}1, l_{k,t}^n \leqslant c_n\\ 0, \text{其他}\end{cases} \tag{1}$$

假设用户至少能得到一台云服务器提供的计算服务，同时用户的任务将被发送到最近的服务器进行处理。

2.1.2 时延模型

1) 传输时延

对于边缘环境中的服务请求，传输时延主要考虑用户上传数据的时间。假设用户与边缘节点之间信道的平均带宽为W，传输功率为P，噪声功率为σ^2；用$h_{k,n}^t=g_{k,n}d_{k,n}^{-\alpha}(t)$表示时隙$t$内用户$k$与服务器$n$之间的信道增益，其中$d_{k,n}(t)$表示用户$k$与服务器$n$之间的距离，$\alpha$为路径损耗指数，$g_{k,n}$为有效信道增益系数[27-28]。则用户$k$与服务器$n$之间最大上行链路传输速率可以表示为

$$r_{k,n}^t=W\text{lb}\left(1+\frac{Ph_{k,n}^t}{\sigma^2}\right) \tag{2}$$

因此，用户将输入数据上传到边缘节点上的时延可表示为

$$t_{k,n}^{\text{up}}=\frac{\text{data}_{i,a}^{\text{in}}}{r_{k,n}^t} \tag{3}$$

在微服务执行完成后，任务的输出结果$\text{data}_{i,a}^{\text{out}}$需要通过下行链路回传至用户设备。其中下行链路的传输速率为$r_{n,k}^t$，则数据回传的下行传输时延可以表示为

$$t_{n,k}^{\text{down}}=\frac{\text{data}_{i,a}^{\text{out}}}{r_{n,k}^t} \tag{4}$$

网络中的每对边缘节点彼此通过路由连通，因此不同节点上具有依赖关系的容器实例之间的数据传输时延由数据量和传输速率决定。用二元变量$y_{i,b,n}$表示微服务实例$I(a,i,b)$是否部署在节点EN_n上。当$I(a,i,b)$部署在EN_n上时$y_{i,b,n}=1$，反之$y_{i,b,n}=0$。则从部署在EN_{n_1}上的容器实例$I(a,i,b_1)$到部署在EN_{n_2}上的容器实例$I(a,j,b_2)$之间的传输时延为

$$T_{\text{com}}^{i,j}=\sum_{n_1=1}^{N}\sum_{n_2=1}^{N}\frac{\text{data}_{j,a}^{\text{in}}}{r_{n_1,n_2}^t}y_{i,b_1,n_1}y_{i,b_2,n_2} \tag{5}$$

2) 计算时延

在本文中，微服务需要处理的数据是其所有前驱微服务传输给它的数据量之和。由于边缘服务器存在异构性，相同微服务的不同实例在不同服务器上的计算时延通常不同，因此假设实例$I(a,i,b)$所在EN_n上CPU的处理能力为$C_{i,b}^a$，则该微服务实例的计算时延为

$$T_{\text{exe}}^{i,n,q}=\frac{\sum_{i=1}^{m}\text{data}_{i,a}^{\text{out}}\mathbf{Dep}_i^j}{C_{i,b}^a} \tag{6}$$

3) 排队时延

由于容器实例在同一时刻只能执行一个请求，因此，本文为每台边缘服务器设置了一个等待队列，旨在缓存那些已选择该服务器上的实例但却因资源竞争未能立即获得执行的任务。等待队列中的任务将根据优先级进行智能调度，确保高优先级任务能优先获得服务器资源。鉴于一个任务由多个微服务按特定拓扑结构组合构成一种工作流模式，因此采用广度优先遍历对任务进行编号。其中，编号较小的任务在工作流中的位置更为前置，其执行时延的累积效应将对后续子任务产生更广泛的影响。因此，在等待队列的调度策略中，优先级高的任务，即编号较低的任务，将被赋予更高的调度优先级，以确保工作流的顺畅执行。队列中任务的排队时间可以迭代计算为

$$Q_{i,b}^{a,\text{task}}=Q_{i,b}^{a,\text{task}'}+T_{\text{exe}}^{i,\text{task}'} \tag{7}$$

2.1.3 能耗模型

当微服务V_i^a在EN_n上执行时，产生的能耗取决于V_i^a在EN_n上的执行时间$T_{\text{exe}}^{i,q}$和处理器单位时间产生的能耗ρ[29]

$$\varepsilon_{\text{exe}}^{i,q}=\rho T_{\text{exe}}^{i,q} \tag{8}$$

当EN_{n_1}和EN_{n_2}发生通信时，产生的通信能耗取决于传输功率P和通信时间$T_{\text{com}}^{i,j}$[29]，即

$$\varepsilon_{\text{com}}^{i,j}=PT_{\text{com}}^{i,j} \tag{9}$$

2.1.4 迁移模型

用户在执行任务前会进行服务器的选择，因此本文使用二元变量$x_{i,n}(t)\in\{0,1\}$和$x_{i,\theta}(t)\in\{0,1\}$来表示是否需要迁移微服务。当发生迁移且将微服务$V_i^a$迁移至服务器$\text{EN}_n$时$x_{i,n}(t)=1$，否则$x_{i,\theta}(t)=1$且请求将被发送到云中心进行处理。由于用户的移动轨迹存在不可预测性，因此网络运营商需要根据用户的移动位置快速地迁移与调度微服务，从而无缝、实时地处理用户请求。然而，服务的动态迁移需要付出额外的能耗与时延，假设微服务V_i^a镜像大小为Image_i，μ^m为单位迁移时延系数，则V_i^a从

EN_{n_1}迁移到EN_{n_2}的时延为

$$\mathrm{Tm}^i_{n_1,n_2}=\frac{\mathrm{Image}_i}{r^t_{n_1,n_2}}+d_{n_1,n_2}(t)\mu^m \tag{10}$$

为了刻画微服务在迁移中对网络开销带来的影响，定义V^a_i从EN_{n_1}迁移到EN_{n_2}的能耗为$\varepsilon\mathrm{m}^i_{n_1,n_2}$，其中$\eta$为单位距离迁移的能耗因子，即

$$\varepsilon\mathrm{m}^i_{n_1,n_2}=\mathrm{Tm}^i_{n_1,n_2}\left(P_{n_1}+P_{n_2}\right)d_{n_1,n_2}(t)\eta \tag{11}$$

2.1.5　服务质量评价模型

为了描述用户执行任务的服务评价指标，联合考虑任务处理时延以及处理任务时通过服务迁移所节约的能量，将服务质量建模为关于任务处理总时延$T^{\mathrm{total}}_{i,j,k}$和能量节省量$E^{\mathrm{save}}_{i,j,k}$的函数，其中，$E^{\mathrm{msave}}_{i,j,k}$表示最大节省能耗，即云上执行能耗和在最大云边加速比MEC上执行能耗的差，$E^{\mathrm{save}}_{i,j,k}$表示节省能耗，即云上执行能耗和在该MEC上执行能耗之差。则服务质量$U_{i,j,k}$为

$$U_{i,j,k}=\begin{cases}\left(\lambda-\lambda^{\frac{\left[T^{\mathrm{total}}_{i,j,k}-T_j\right]^+}{\beta_jT_j}}\right), T^{\mathrm{total}}_{i,j,k}\leqslant\beta_jT_j, E^r_j\geqslant\gamma E^{\max}_j\\ \left(\lambda-\lambda^{\frac{\left[T^{\mathrm{total}}_{i,j,k}-T_j\right]^+}{\beta_jT_j}}\right)\left(\lambda-\lambda^{\frac{\left[E^{\mathrm{msave}}_{i,j,k}-E^{\mathrm{save}}_{i,j,k}\right]^+}{E^{\mathrm{msave}}_{i,j,k}}}\right), T^{\mathrm{total}}_{i,j,k}\leqslant\beta_jT_j, 0<E^r_j<\gamma E^{\max}_j\\ 0,\text{其他}\end{cases} \tag{12}$$

其中，$[x]^+=\max\{x,0\}$，$\lambda>1$表示服务质量对任务完成时延和能量节省量敏感度，γ表示衡量能量消耗阈值占据执行任务的最大能耗的比例，β表示最大可容忍时延的控制因子。

1) 当$E^r_j\geqslant\gamma E^{\max}_j$时，用户执行任务的能耗不低于能耗阈值，服务质量主要取决于处理时延$T^{\mathrm{total}}_{i,j,k}$。当用户设备的处理时延$T^{\mathrm{total}}_{i,j,k}$不超过任务的理想时延阈值时，即$T^{\mathrm{total}}_{i,j,k}\leqslant T_j$时，$\dfrac{\left[T^{\mathrm{total}}_{i,j,k}-T_j\right]^+}{\beta_jT_j}=0$，因此有

$$U_{i,j,k}=\lambda-\lambda^{\frac{\left[T^{\mathrm{total}}_{i,j,k}-T_j\right]^+}{\beta_jT_j}}=\lambda-1>0 \tag{13}$$

2) 当$T_j<T^{\mathrm{total}}_{i,j,k}\leqslant\beta_jT_j$时，用户设备的处理时延$T^{\mathrm{total}}_{i,j,k}$超过任务的理想时延阈值，但没有超过可忍受的最大时延，$\dfrac{\left[T^{\mathrm{total}}_{i,j,k}-T_j\right]^+}{\beta_jT_j}\leqslant\dfrac{\left(\beta_j-1\right)}{\beta_j}<1$，因此有

$$U_{i,j,k}=\lambda-\lambda^{\frac{\left[T^{\mathrm{total}}_{i,j,k}-T_j\right]^+}{\beta_jT_j}}\leqslant\lambda-\lambda^{\frac{\beta_j-1}{\beta_j}}<\lambda-1 \tag{14}$$

此时，服务质量会随着处理时延的增加而从$\lambda-1$下降到0。

3) 当$E^r_j<\gamma E^{\max}_j$时，用户执行任务的能耗低于能耗阈值，此时服务质量不仅取决于处理时延$T^{\mathrm{total}}_{i,j,k}$，也取决于通过迁移处理所节省的能量$E^{\mathrm{save}}_{i,j,k}$，当用户设备的处理时延$T^{\mathrm{total}}_{i,j,k}$不超过任务的理想时延阈值且能够节省最大能量时，有

$$\lambda-\lambda^{\frac{\left[T^{\mathrm{total}}_{i,j,k}-T_j\right]^+}{\beta_jT_j}}=\lambda-1 \tag{15}$$

$$\lambda-\lambda^{\frac{\left[E^{\mathrm{msave}}_{i,j,k}-E^{\mathrm{save}}_{i,j,k}\right]^+}{E^{\mathrm{msave}}_{i,j,k}}}=\lambda-1 \tag{16}$$

$$U_{i,j,k}=(\lambda-1)^2 \tag{17}$$

4) 当$T_j<T^{\mathrm{total}}_{i,j,k}\leqslant\beta_jT_j, 0<E^{\mathrm{save}}_{i,j,k}<E^{\mathrm{msave}}_{i,j,k}$时，任务执行的处理时延超过任务的理想时延阈值但不超过可以忍受的最大时延，并且节省的能量低于最大节省能量$E^{\mathrm{msave}}_{i,j,k}$，则

$$U_{i,j,k}<(\lambda-1)^2 \tag{18}$$

此时，服务质量会随着任务处理时延$T^{\mathrm{total}}_{i,j,k}$的增大和能量节省量$E^{\mathrm{save}}_{i,j,k}$的降低而从$(\lambda-1)^2$逐渐下降，直到任务要求无法得到满足，或者没有能源节省时，服务质量评价为0。

2.1.6　负载均衡模型

为了衡量用户移动过程中因微服务的调度和动态迁移导致的网络负载变化，本文同时考虑计算资源与存储资源的负载均衡模型，其中$f^{\max}_n$表示EN_n的最大计算资源量，$S^{\max}_n$表示EN_n的最大存储资源量，ϕ_1和ϕ_2分别表示对计算资源以及存储资源的权重，满足$\phi_1+\phi_2=1$。二元变量$z^t_{i,n,a}=1$表示t时隙V^a_i被调度到EN_n上执行，则t时隙EN_n的负载为

$$L^t_n=\phi_1\frac{\sum\limits_{i\in M}z^t_{i,n,a}C^a_{i,b}\mathrm{data}^{\mathrm{in}}_{i,a}}{f^{\max}_n}+\phi_2\frac{\sum\limits_{i\in M}\left(y_{i,b,n}\mathrm{Image}_i+z^t_{i,n,a}\mathrm{data}^{\mathrm{in}}_{i,a}\right)}{S^{\max}_n} \tag{19}$$

考虑到边缘集群中不同服务器之间存在异构性，则服务器之间的平均负载定义为

$$\bar{L}_t=\frac{1}{N}\sum_{n\in N}L^t_n \tag{20}$$

为了考察服务器之间的负载分布情况，定义平均负载偏差系数用于表示当前集群中各服务器之间的负载差异水平，即

$$\mathrm{LF}_n^t = \frac{1}{N}\sum_{n\in N}|L_n^t - \bar{L}_t| \tag{21}$$

2.1.7 云边加速比

在微服务部署场景中，若某一服务的响应时延在边缘端显著低于云端，则表明该服务能够从边缘部署中获益。为了优化部署策略，降低微服务在调度时的平均响应时延，应当优先将那些在边缘部署中受益较大的微服务放置在边缘服务器上。为此，本文引入了云边加速比[30]这一指标，用以量化微服务在边缘部署中所能获得的性能提升。由于边缘节点存在异构性，同一微服务在不同节点上的响应时间往往不同，因此微服务 V_i^a 部署在 EN_n 上时的云边加速比 S_n^i 可以表示为

$$S_n^i = \frac{T_{\mathrm{com}}^{i,\phi} + T_{\mathrm{exe}}^{i,\phi,q}}{T_{\mathrm{com}}^{i,j} + T_{\mathrm{exe}}^{i,n,q}} \tag{22}$$

由式(22)可知，当云边加速比越大时，将该微服务部署到边缘端所获得的性能提升越大，即边缘部署的受益程度越高。

由于微服务的部署必须满足硬件资源的约束，为了充分使用边缘端的异构资源，则应该将微服务部署在当前负载程度最小的节点上。为此，本文使用边缘效益指数（EGI）来表示选择服务器执行任务时相比于云上带来的效益，最终微服务 V_i^a 在对应 EN_n 上执行的 EGI_n^i 可以表示为

$$\mathrm{EGI}_n^i = S_n^i + \left(1 - L_n^t\right) \tag{23}$$

EGI 在部署时联合优化性能增益与资源利用率，避免节点过载，同时量化用户在选择该边缘服务器时对系统的增益水平。

2.2 问题描述

基于以上系统模型，本文考虑高并发用户在移动中的跨域问题，希望找到一种能够在节省资源的同时最大限度地保证用户 QoS，并提升任务执行中 EGI 的微服务选择策略。因此，微服务在调度和迁移时的最优选择问题可以定义为

$$\max\left(\sum_{i=1}^{M}\mathrm{EGI}_n^i\right)$$

$$\text{s.t.}\quad \mathrm{C1}: y_{i,b,n}\in\{0,1\},\forall V_i^a\in V^a,\mathrm{EN}_n\in\mathrm{EN}$$

$$\mathrm{C2}: \sum_{n=1}^{N} y_{i,b,n} = 1,\forall V_i^a\in V^a,\mathrm{EN}_n\in\mathrm{EN}$$

$$\mathrm{C3}: \sum_{a\in A}\sum_{i\in M} z_{i,n,a}^t C_{i,b}^a \mathrm{data}_{i,a}^{\mathrm{in}} < f_n^{\max},\forall\mathrm{EN}_n\in\mathrm{EN}$$

$$\mathrm{C4}: \sum_{a\in A}\sum_{i\in M}\left(y_{i,b,n}\mathrm{Image}_i + z_{i,n,a}^t\mathrm{data}_{i,a}^{\mathrm{in}}\right) < S_n^{\max} \tag{24}$$

其中，约束条件 C1 和 C2 表示容器实例是任务分配的最小单元，每个任务只能由一个实例执行；C3 和 C4 表示所选边缘节点必须有足够的计算资源和存储资源来部署和调度，如果任一资源不满足，则无法进行迁移或调度计算。本文涉及的主要系统参数如表 1 所示。

表 1 主要系统参数

参数	含义
A	应用程序
V	微服务集合与微服务
E	微服务之间的依赖关系
$I(a,i,b)$	微服务 V_i^a 的第 b 个实例
EN	服务器集合
$\mathcal{K}$	用户集合
$\mathcal{T}$	时间片集合
l_t^k	时隙 t 用户 k 与各服务器的距离
c_n	服务器 n 的覆盖范围
$\rho_{k,t}^n$	二元变量，表示时隙 t 内，用户 k 是否在服务器 n 的覆盖范围内
W	信道带宽
P	传输功率
σ^2	噪声功率
α	路径损耗指数
$g_{k,n}$	信道增益系数
$y_{i,b,n}$	表示微服务的部署位置的二元变量
$U_{i,j,k}$	用户服务质量
μ^m	单位迁移时延系数
γ	衡量能量消耗阈值占据执行任务的最大能耗的比例
β	最大可容忍时延的控制因子
$E_{i,j,k}^{\mathrm{save}}$	节省能耗
$E_{i,j,k}^{\mathrm{msave}}$	最大节省能耗
$f_n^{\max}, S_n^{\max}$	服务器节点 n 的最大计算和存储资源量
LF_n^t	集群的平均负载偏差
S_n^i	微服务 V_i^a 部署在 EN_n 上时的云边加速比
EGI_n^i	边缘效益指数
ω_i	EN_n 上微服务 V_i^a 的云边迁移权重
Ψ	服务实例的繁忙程度

3　微服务部署与迁移选择算法

3.1　基于边缘效益指数的微服务部署算法

为解决微服务实例在资源受限的边缘服务器上部署困难的问题，本文提出了将具有复杂依赖关系的工作流任务在边缘节点上进行部署的 Edge-Deployment 算法。Edge-Deployment 算法的输入是边缘服务器集合 EN，服务器可承载的容器实例数量 MaxSize 以及用有向无环图表示的微服务集合 V_i^a。Edge-Deployment 算法的描述如算法 1 所示。

算法 1　Edge-Deployment 微服务部署算法

输入　EN，MaxSize，V_i^a

输出　部署方案 $\mathcal{A}$

1)系统初始化

2)根据式(5)、式(6)、式(22)计算出各微服务在 EN 中各服务器的云边加速比

3)for $V_i^a \in V^a$

4)　对微服务 V_i^a 在 EN 中的云边加速比排序

5)　根据式(19)、式(22)和式(23)计算 EGI_n^i

6)　判断对于 EGI_n^i 最大的 EN_n，剩余的资源量是否满足部署要求

7)　　若满足，将 V_i^a 部署在 EN_n 上

8)　　更新 EN_n 的剩余资源

9)　若不满足，遍历排序中的下一个 EN

10) 没有找到合适的边缘节点，则目标为云

11)将当前容器实例部署到云上

12)end for

13)返回部署结果 $\mathcal{A}$

算法 1 首先根据容器的平均加速比进行降序排序，以确保高加速比的容器能够优先部署到边缘节点。在部署过程中，遍历排序后的边缘节点，为每个容器实例挑选最能提升整体性能的部署位置。这种排序策略采用了贪心思想，确保边缘端有限资源的高效利用。在选择部署节点时，首先剔除那些无法满足容器硬件资源需求以及那些无法从边缘计算中获益的节点（第 6 行~第 8 行）。随后，在剩余的节点中，会选择优先级最高的节点进行容器部署。在边缘集群负载极高的情况下，可能无法找到合适的边缘节点进行部署，此时容器将被部署到云端（第 9 行~第 10 行）。同时为了保证部署的准确性，每成功部署一个容器实例，都会同步更新对应节点的剩余资源信息（第 8 行）。最终，该部署算法确保了每个微服务实例都能找到一个合适的部署位置。算法 1 的时间复杂度主要由计算加速比和部署微服务两部分组成，在由 N 个边缘节点，M 个微服务组成的场景中，计算云边加速比并存储的复杂度为 $O(MN)$，对边缘服务器按云边加速比进行排序、检查资源并部署的复杂度为 $O\left(MN\log N\right)$，部署在云端的时间复杂度为 $O(1)$。因此算法 1 的总体时间复杂度为 $O\left(MN\log N\right)$。

3.2　基于动态云边迁移的微服务选择算法

用户移动中微服务调度示例如图 2 所示。考虑用户移动过程中发生的跨域情况，利用 MBS 覆盖范围广这一特点，保证区域内的用户至少可以将任务通过一台云服务器进行处理。本文选择将中心云服务器部署在 MBS 旁， MEC 服务器部署在各个 SBS 旁并通过蜂窝基站实现分布式部署，为临近的终端用户提供服务。在当前时隙内，每个用户有一个等待执行的微服务并访问最近的边缘节点。通过算法 1，每台边缘服务器上都部署了一定数量的微服务实例。

考虑共有 7 台服务器，包括 1 台云服务器和 6 台边缘云服务器的云边混合云场景（如图 1 所示）。假设云中心部署着所有种类的微服务实例，通过算法 1，每台边缘服务器上都部署了一定数量的微服务实例。一个移动用户沿着道路前进，具有明显的方向性，移动轨迹如图 2 所示。将其按时隙统计，不同时隙中可以调度的服务器集合如表 2 所示。表中的 1 表示当前用户在服务器覆盖范围内，可以选择其执行微服务，反之为 0。

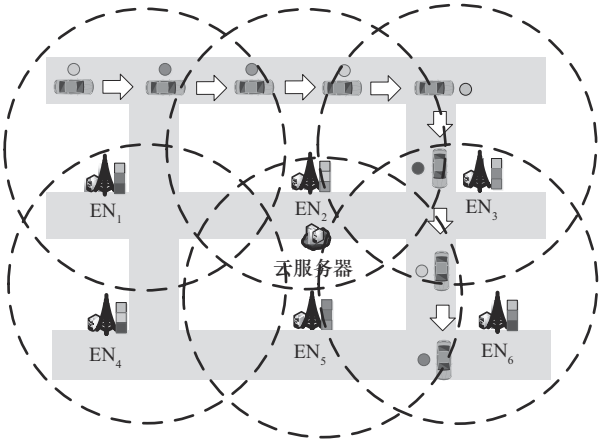


图 2　用户移动中微服务调度示例

表2 不同时隙中可以调度的服务器集合

t	EN_0	EN_1	EN_2	EN_3	EN_4	EN_5	EN_6
1	1	1	0	0	0	0	0
2	1	1	1	0	0	0	0
3	1	1	1	0	0	0	0
4	1	0	1	1	0	0	0
5	1	0	1	1	0	0	0
6	1	0	1	1	0	0	1
7	1	0	0	1	0	1	1
8	1	0	0	0	0	1	1

假设图2所示片区为一个边缘集群，当用户周围没有所需微服务实例或等待队列过长时，可以在集群内寻找部署了该实例的服务器进行迁移。若无服务器部署该实例，则该服务实例将会在云中心执行。在t_1时刻，服务器EN_1并未搭载所需的微服务实例V_1^a，此时比较V_1^a分别从EN_2和EN_4迁移至EN_1时的EGI的大小，进而确定迁移源。在t_7时刻，可以选择EN_3、EN_5和EN_6作为待迁移服务器，由图2可知这3台服务器上都未部署所需的微服务，因此此时选择去云中心执行。若跨域时任务尚未执行完，且离开了正在执行的服务器，将重新选择服务器执行该微服务。

为了避免在并发访问时因集群中缺少某一微服务而发生频繁访问云中心导致时延增加和资源浪费的问题，本文设计了动态云边迁移算法对容器实例进行从云端到边缘端的调整。在请求-应答的模式下，空闲状态的微服务不可避免地造成了服务器资源的暂时性闲置与浪费，而本应部署在边缘以提升效率的忙碌服务却受限于多种因素，依然部署于云端，这一现状无疑影响了系统整体响应时延，削弱了边缘计算本应带来的性能优势[30]。因此当用户找不到微服务实例而访问云中心时触发云边迁移，通过动态调整的方式把所需的微服务部署在集群中EGI最大的边缘服务器上，并将近期最不忙碌的微服务实例从服务器上移除。为了判断应该移除哪个微服务实例以部署新的微服务，本文引入了云边迁移权重以衡量服务实例在一定时间内的繁忙程度，并根据当前服务的繁忙程度预估服务实例未来的繁忙程度。EN_n上微服务V_i^a的云边迁移权重可以表示为

$$\omega_i = S_n^i \Psi_i \tag{25}$$

其中，Ψ_i表示该服务实例的繁忙程度，由每个时间片内对该实例的访问次数决定。然而，历史数据的有效性应该是逐渐降低的，因此可以通过对历史繁忙数据进行加权，距离当前时间越近的繁忙数据权重越大，反之权重越小，则一个服务实例的繁忙程度可以表示为

$$\Psi = \Lambda_0 + \Lambda_1 \xi^1 + \Lambda_2 \xi^2 + \cdots + \Lambda_n \xi^n \tag{26}$$

其中，Λ_t表示在时隙t里的访问次数，ξ表示衰减因子且$\xi^{32} = 0.5$，历史数据以等比递减的方式影响总的繁忙程度，并于32个时间帧后衰减至一半。每个服务器通过使用小顶堆来维护所有已部署微服务的云边迁移权重，以便选择迁移对象。

在调度时为了充分利用边缘端资源，同时防止在处理子任务时出现集中处理导致的等待时延过长，采用基于广度优先搜索的算法构建任务调度队列，做到以下两点。

1)依赖型任务在满足其前置条件后能够准确执行，避免因等待依赖任务完成而导致的时延。

2)最大化系统中的并行处理能力，从而减少整体的执行时延。

在任务执行中把工作流按依赖关系进行分层，构建多个调度队列，处于同一调度队列中的微服务并行执行，以最大化系统的并行能力，分层调度模型如图3所示。

基于动态云边迁移的微服务选择算法（DMS, dynamic microservice selection）如算法2所示。

算法2 DMS微服务选择算法

输入 EN，MaxSize，V_i^a

输出 云边调整是否成功

1)系统初始化

2)调用Edge-Deployment()在集群中部署微服务

3)for $V_i^a \in V^a$

4) 判断可调度服务器中是否存在服务器部署了V_i^a：

5) 若存在，申请在该服务器上执行

6) 判断是否有空闲资源

7) 若无，进入等待队列

8) 若有，执行V_i^a

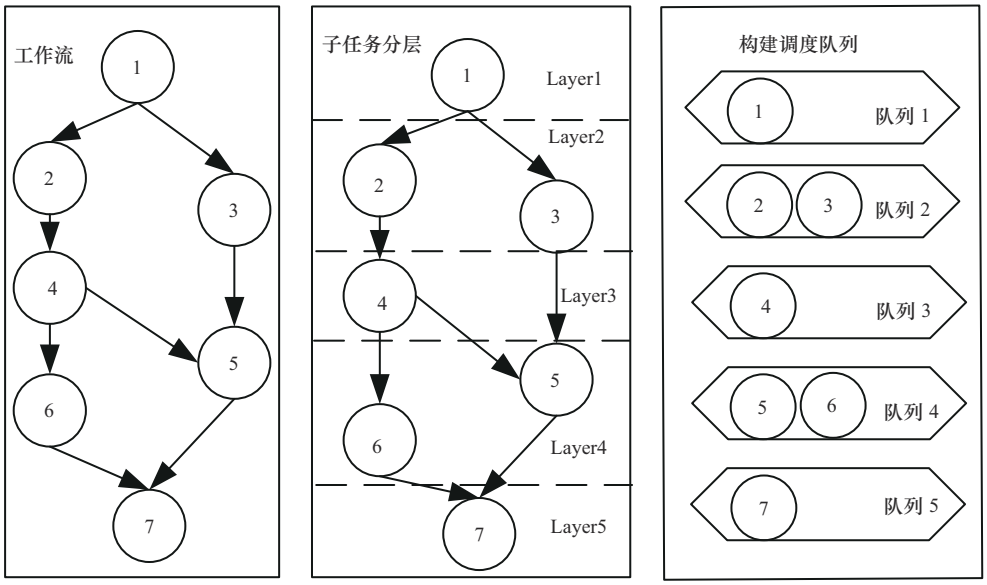


图3　分层调度模型

9)　　利用式(17)更新权重
10) 若不存在，考虑迁移或云上执行
11)　　判断当前集群是否有迁移源并生成可迁移服务器集合 $\mathcal{E}_i^t$
12)　　按 EGI 遍历 $\mathcal{E}_i^t$ 中的服务器，询问是否有空闲资源
13)　　若均无资源，在云上执行 V_i^a
14)　　选择 EGI 最大的可迁移服务器进行云边迁移并从堆中换出权重最小的微服务
15)　　利用式(25)更新权重
16)　　若存在 EN_n 有空闲资源
17)　　将 V_i^a 迁移至 EN_n 上执行
18)　　用式(25)更新权重
19) V_i^a 执行完时，判断是否跨域
20) 若未跨域，return true
21) 若跨域，重新执行第 4 行~第 18 行
22)end for
23)返回云边迁移结果

算法 2 描述了在用户执行任务中寻找和选择微服务的过程。首先利用算法 1 在集群中部署微服务，然后对于工作流任务中的每一个微服务遍历用户周围的服务器，判断是否存在一个有空闲资源的服务器部署所需的微服务；若没有，则遍历集群中的服务器以寻找迁移源，选择有空闲资源且 EGI 最大的服务器作为迁移目标（第 4 行~第 14 行）。若找不到空闲服务器或集群中不存在服务器部署了该微服务时，选择云中心执行并在集群中选择 EGI 最大的服务器进行云边迁移。每当通过边缘服务器执行完一个微服务时，都需要更新堆中该微服务的权重，并判断是否在执行过程中离开了执行服务器，若出现跨域，则需要重新执行该服务（第 15 行~第 21 行）。算法 2 在查找搭载所需微服务的服务器以及检查资源时的时间复杂度为 $O(N)$，在迁移选择与微服务部署更新时，需要在堆中进行排序与云边迁移置换，此时按 EGI 排序的复杂度为 $O\left(N\log N\right)$，而管理堆中微服务权重的复杂度为 $O\left(\log M\right)$。因此算法 2 的总体时间复杂度为 $O\left(N\log N+\log M\right)$。

4　仿真分析

为了验证本文 DMS 算法的有效性，本节将其与其他算法进行对比。不同微服务选择与迁移算法的综合对比如表 3 所示。其他算法的基本思路与 M 台边缘服务器上部署 N 类微服务实例时的复杂度特征如下。

1) AM（always migrate）[19]：部署策略采取 Zipf 分布与随机部署相结合的策略。初始状态时随机选择一定数量的微服务作为流行微服务，并将其部署在所有节点上。调度时服务实例始终跟随用户迁移至最近的边缘节点。

2) NM（never migrate）[19]：部署时采用均匀部署策略，调度时服务实例始终位于初始边缘节

表3 不同微服务选择与迁移算法的综合对比

算法	部署方式	迁移策略	调度策略	复杂度
AM	Zipf分布与随机部署相结合	始终就近迁移	就近执行	$O(MN)$
NM	均匀部署	不进行迁移	仅在初始服务器上执行	$O(M)$
SwarmSpread	均匀部署	负载均衡策略	基于 Docker Swarm 机制调度	$O(MN)$
SwarmRandom	随机部署	不进行迁移	随机选择服务器执行任务	$O(MN)$
LOSM	初始均匀部署	基于 Lyapunov drift 计算迁移决策	基于线性优化动态任务调度	$O\left(\tau\left(\max\left(mn,m\right)\right)^2\right)$
DMS	根据云边加速比部署	动态云边迁移	分层调度并根据EGI选择	$O\left(N\log N+\log M\right)$

点，不进行迁移。

3) SwarmSpread：部署时采取均匀部署策略，调度时采用Docker中集群的调度策略，如果节点配置相同，选择一个访问总量最少且距离近的节点进行部署或调度。

4) SwarmRandom：部署时采取随机部署策略，调度时采用Docker中集群的调度策略，该策略将对服务进行随机分配，不考虑集群中节点的状态。

5) LOSM（Lyapunov optimization-based online service migration）：该算法参考文献[15]的主要思想，采用Lyapunov优化理论进行微服务迁移。初始部署时采取均匀部署策略，在用户移动过程中每隔一段时间根据流行度对微服务重新部署。

本文采用了某平台申请的2016年11月位于成都的轨迹数据集，数据集中轨迹信息的格式为一个四元组（id,Tstamp,longitude,latitude）。其中，id表示订单ID，Tstamp表示时间戳，longitude和latitude分别表示经度、纬度，每隔约3 s记录一条轨迹定位数据。本文在预处理阶段通过连续轨迹点间的距离与时间间隔计算出车辆的移动速度，并据此划分出3类典型速度区间：低速（约1.5 m/s）、中速（约5 m/s）和高速（约10 m/s），从而模拟不同移动程度下的服务请求行为。轨迹区域划分示例场景如图4所示，本文参考文献[31]、文献[32]的思想，在目标区域中进行二维空间分割，该空间被分割为100个正方形小区，在每个小区中部署边缘节点，为该小区覆盖范围内的车辆提供服务，假设一个九宫格区域为一个边缘集群，可以在同一边缘集群中进行迁移和调度。

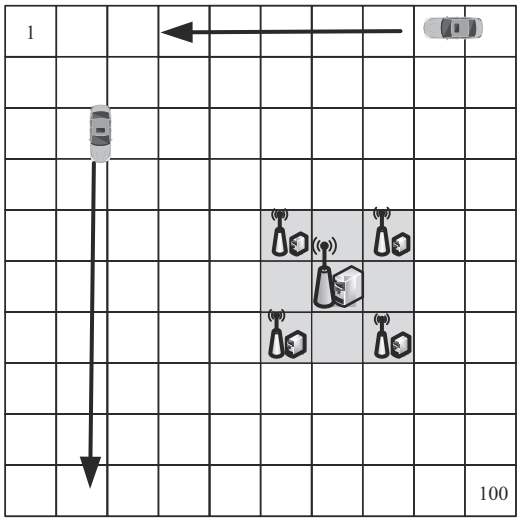


图4 轨迹区域划分示例场景

4.1 仿真参数设置

本文采用文献[24]提供的Montage、Cyber-Shake和LIGO这3种基准工作流进行仿真实验。Montage和Cyber Shake工作流由20个微服务组成，其中Montage工作流任务层数较多，Cyber Shake工作流较为扁平，但并行程度高。LIGO工作流由40个微服务组成，用于模拟求解空间大、决策变量多的场景。每个工作流代表一类业务，用有向无环图表示任务间的依赖关系。微服务之间的数据传输量定义为10~15 MB，镜像大小定义为30~50 MB，所需的处理器总周期数为1 200~1 500 Megacycle。边缘节点CPU的计算能力为5~10 GHz，在一个单位时间内的能耗ρ为5 mW，内存容量在[4,8,12] GB中随机选择，信道带宽W设为5 MHz，传输功率为50~100 mW，信道噪声功率σ^2为−100 dBm，路径损耗因子α为4。假设服务器位于划分后小区的中

心，边缘服务器的覆盖范围为200~300 m，云服务器的覆盖范围为2~3 km，云服务器上CPU的计算能力为30~50 GHz。在计算用户满意度时，设置单个微服务的理想时延阈值为0.5 s，时延忍受程度因子为2，能耗阈值比例为0.5。

4.2　实验结果及分析

1) 并发程度对算法性能的影响

首先通过改变并发程度来分析并发数对时延、能耗、负载均衡、任务完成率和边缘命中率的影响。在Montage工作流下，容器数量为6时实验结果并发程度对算法性能的影响如图5所示。

由图5(a)可知，无论在何种并发程度下，本文DMS算法的平均时延始终低于其他5种算法。这是因为DMS算法能够根据用户的实时位置和边缘节点的负载情况动态调整微服务的部署位置，从而选择时延最小的节点执行任务。其中AM性能随并发数增加时延上升最快。这是因为AM算法总是将服务实例迁移到最近的边缘节点，导致资源竞争激烈，时延增加。NM算法的时延也相对较高，因为它始终将服务实例部署在初始边缘节点，不考虑用户的实时位置和边缘节点的负载情况。

由图5(b)和图5(c)可知，在能耗和命中率方面，本文DMS算法和SwarmSpread算法表现良好，这是因为这2种算法在寻找微服务时边缘命中率高，能够充分利用边缘节点资源，从而降低能耗。同时DMS算法由于存在分层调度与云边迁移，使得在时延、能耗和命中率这3个指标上明显优于其他方法。

由图5(d)可知，在负载偏差系数方面，本文DMS算法的负载偏差系数相对较低，并且随着并发数的增加，负载偏差系数变化不大。这说明DMS算法能够有效地平衡各个边缘节点的负载，避免资源集中在少数节点上，提高系统资源的利用率。而AM算法的负载偏差系数较高，并且随着并发数的增加，负载偏差系数显著波动。这说明AM算法无法有效地平衡各个边缘节点的负载，随着并发数的增加，资源竞争加剧，导致部分节点负载过高，而其他节点负载过低，资源利用率不均衡。

图5(e)为Montage工作流下，容器数量为6时并发数对服务质量评价的影响。可以看到，虽然本文DMS算法在总时延上通过分层调度取得了一定优势，但是在单一微服务的执行中，由于出现频繁迁移，尽管能够找到EGI最大的服务器执行以减少执行时延，但随着并发数的进一步增大，排队时延上升时，服务指标会呈现明显的下降趋势。

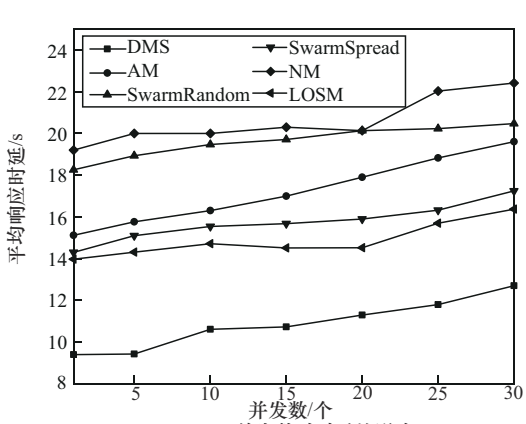
图5(f)为Montage工作流下，并发数对按时完成请求数量的影响。可以看到，随着并发数的增加，各算法的完成率都处于降低的趋势，但是本文DMS算法在并发数为40时仍能保持97.5%的完成率，相比其他算法提升了约16%。

图5(g)和图5(h)展现了在不同并发数下各方法在平均迁移次数和服务质量波动水平上的差异。本文DMS算法和LOSM算法在较高并发情况下表现出较为稳定的特性，迁移次数保持较低，并能够提供更加平稳的服务质量，确保了用户服务的连续性。DMS算法通过动态云边迁移优化微服务部署，显著减少了迁移频次，从而有效避免了因频繁迁移带来的服务质量波动。相比之下，AM算法在低并发情况下表现良好，迁移次数和服务质量波动都较低，但随着并发数增加，迁移次数逐渐增加，服务质量波动也显著增大。这是因为AM算法始终选择就近服务器进行迁移，在高并发情况下，系统需要频繁迁移以保证任务执行的及时性，从而导致了较大的服务质量波动。

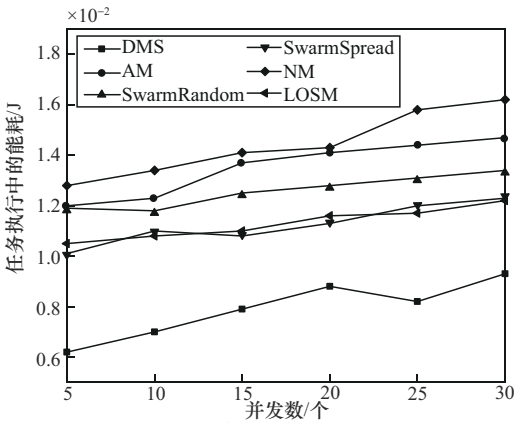
2) 容器数对算法性能的影响

在并发数为30、Montage工作流下，不同容器数量对平均时延和平均能耗的影响如图6(a)和图6(b)所示。可以看出，在请求数量较低的时候，容器数量越大越能分担服务器的压力，使得更多的任务不需要迁移或者到云服务器就能完成，并显著降低排队时延。尤其对于DMS算法，容器数量的增加进一步增强了其在动态迁移决策时的灵活性，使得高边缘效益的微服务更有机会优先部署在边缘，从而降低总体响应时延和能耗。

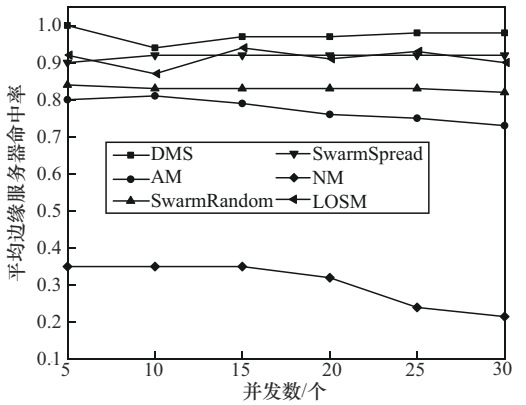
随着容器数量的增加，MEC上的命中率增加，在迁移时找到搭载微服务的源服务器的概率也会增大，因此在能耗上是一个下降的趋势。对于SwarmRandom和SwarmSpread而言，虽然在MEC上的命中率增加了，但是并不意味着MEC上部署着所需微服务就会在该服务器上执行，因为其在调度时的策略分别是随机选择和寻找访问量最小的服务器，因此容器数量对其影响并不明显。容器数量对用户服务质量评价指标的影响如图6(c)所示。可以看出，随着可用容器数的增加，本文DMS算法可以通过云边调整部署，更



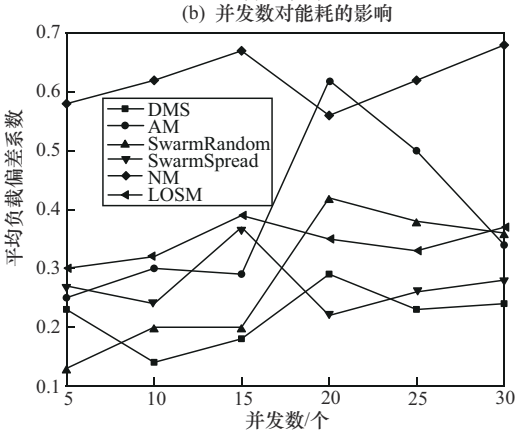
(a) 并发数对时延的影响



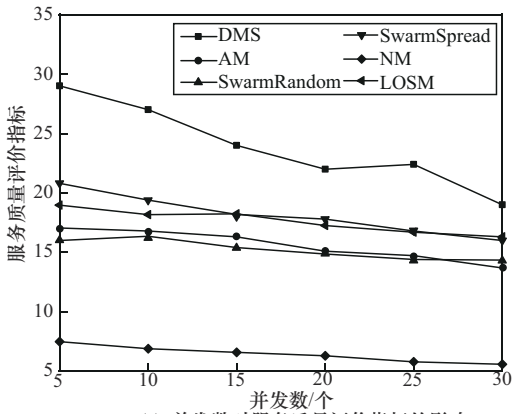
(b) 并发数对能耗的影响



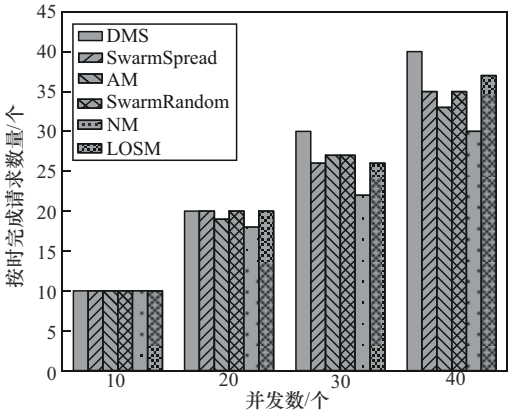
(c) 并发数对边缘服务器命中率的影响



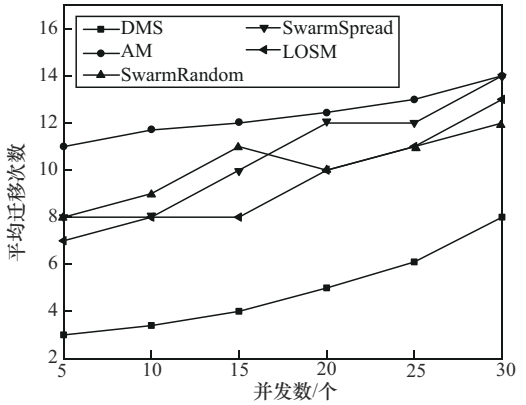
(d) 并发数对负载偏差系数的影响



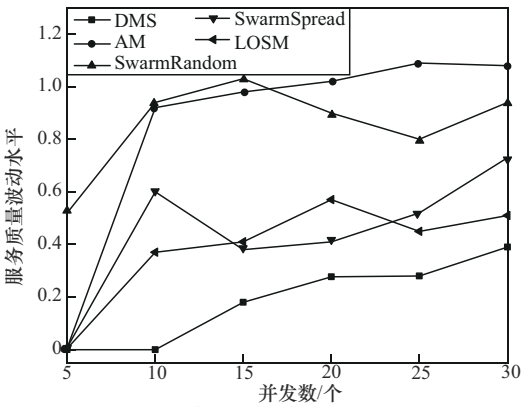
(e) 并发数对服务质量评价指标的影响



(f) 并发数对按时完成请求数量的影响



(g) 并发数对平均迁移次数的影响

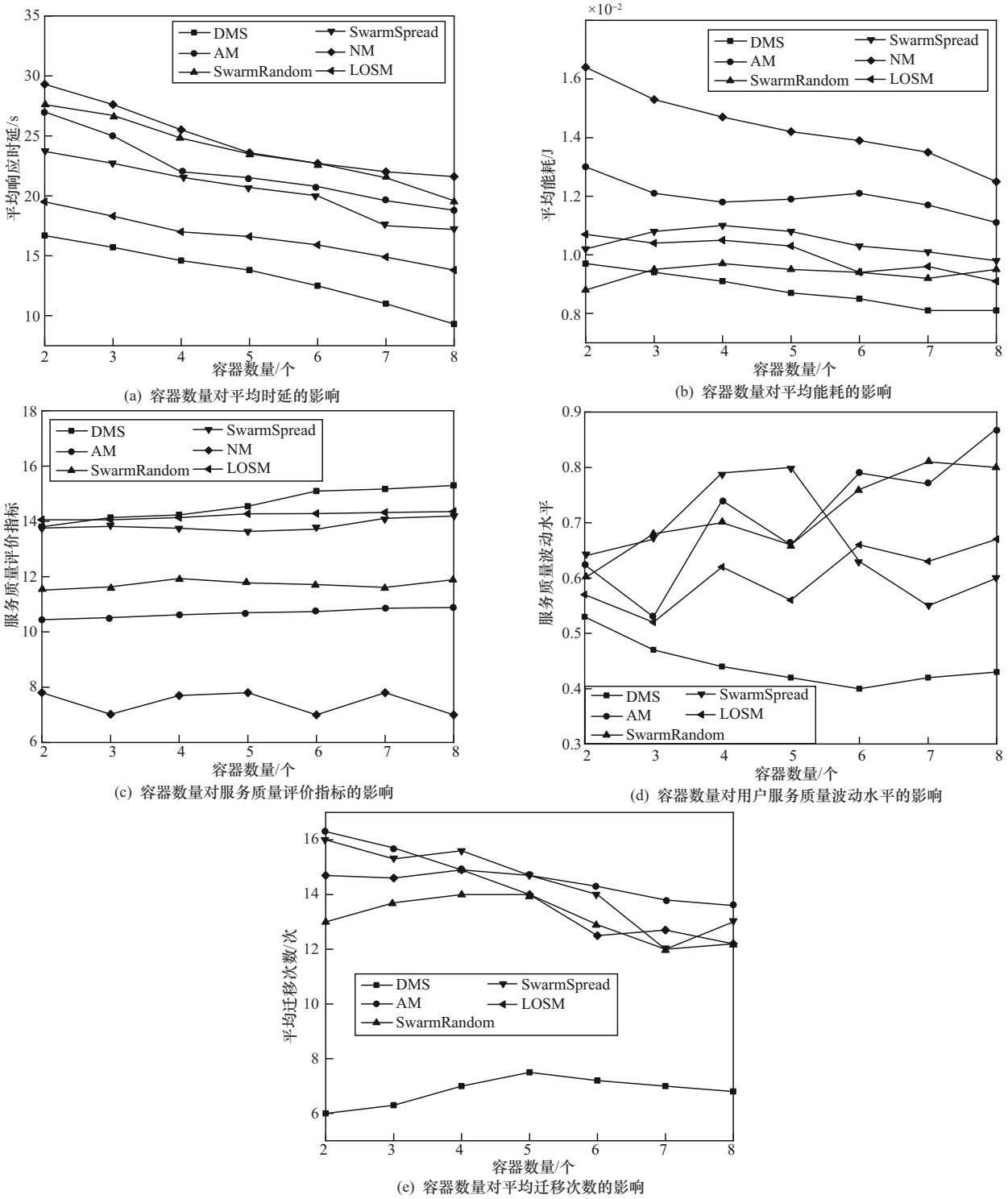


(h) 并发数对服务质量波动的影响

图5　并发数对算法性能的影响

好地利用集群中的资源，从而有效提升服务质量。LOSM 算法虽然能够通过 Lyapunov 优化理论进行微服务部署与迁移，但由于不能实时进行云边迁移的调整，其服务质量在容器数量增加时没有显著变化。

图 6(d)和图 6(e)展现了在不同容器数量下各方法分别在迁移次数和服务质量波动水平上的差异。可以看出，本文 DMS 算法表现相对稳定，迁移次数始终较低。在容器数量较少时有服务质量轻微波动，随着容器数量的增加，波动减小，能够在



(a) 容器数量对平均时延的影响

(b) 容器数量对平均能耗的影响

(c) 容器数量对服务质量评价指标的影响

(d) 容器数量对用户服务质量波动水平的影响

(e) 容器数量对平均迁移次数的影响

图 6　容器数对算法性能的影响

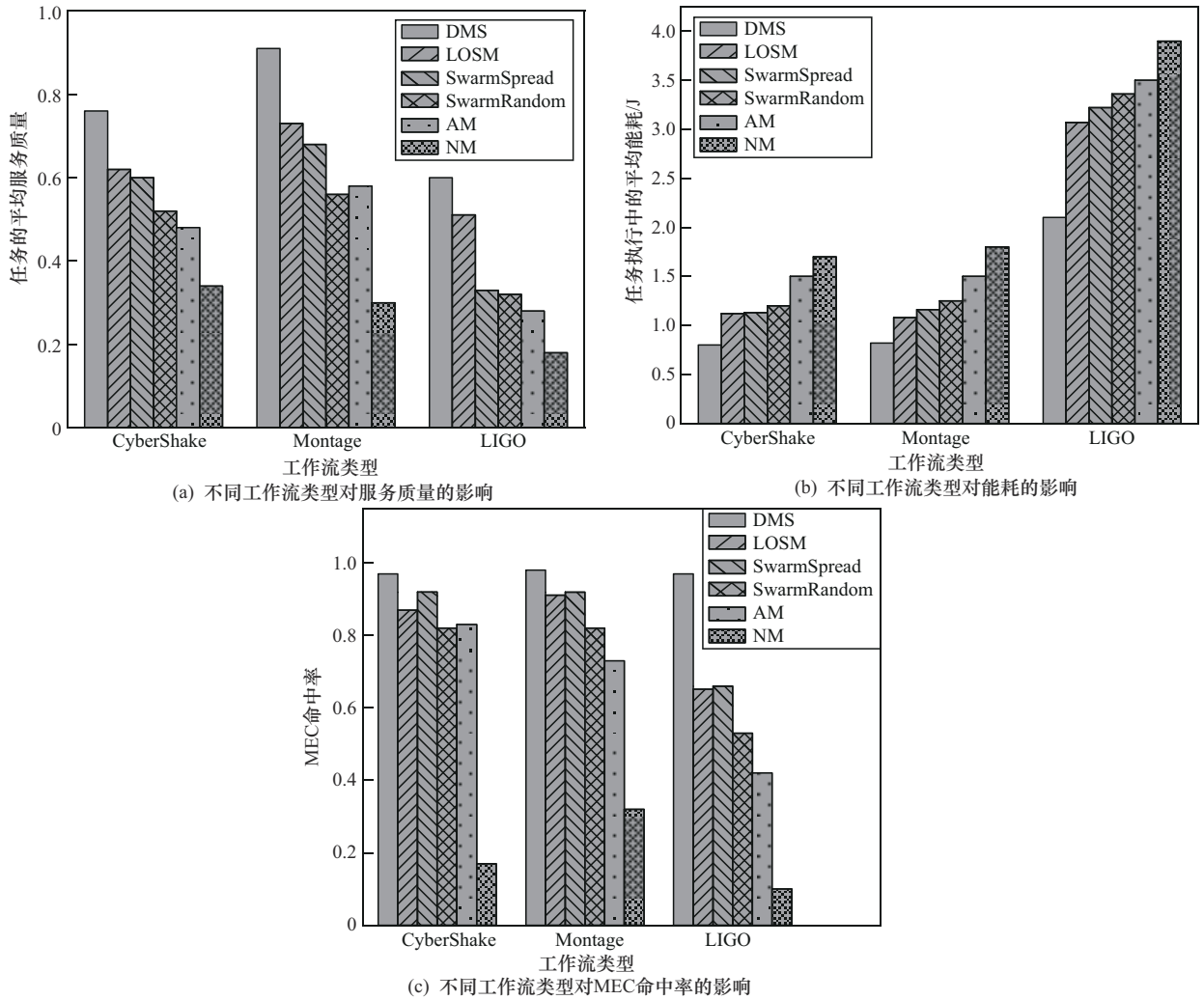
发生迁移时更好地平衡各个服务器的微服务部署情况，从而保障用户频繁跨域移动时服务质量的稳定性。相比之下，AM算法在容器数量较多时虽然迁移次数有所减少，但服务质量波动显著增大，表现不稳定。SwarmRandom算法负载分配不精准，导致服务质量波动较大，尽管增加容器数量有所缓解，但波动仍然较高。LOSM算法通过集群重新部署微服务的方式进行优化，能够在一定程度上减少迁移次数和波动，但由于其无法实时调整微服务部署决策，因此难以适应高动态性的用户请求，迁移次数仍然较高，服务质量波动依然存在。

3) 不同工作流类型对算法性能的影响

不同工作流类型对算法性能的影响如图7所示。可以看出，随着子任务数量的增加，所有算法的开销和平均时延都是增加的趋势。这是因为虽然子任务数在增加，但是每台服务器部署的微服务数量并没有增加，导致越来越多的微服务无法在边缘命中微服务实例，进而会频繁地去访问云中心，带来大量的时延能耗损失。本文DMS算法由于采用动态云边迁移机制，每当找不到所需微服务实例而访问云端时会动态调整部署结构，将所需的微服务实例部署在边缘集群中，故而保持了较高的MEC命中率。对比算法中AM增加的幅度最大，这是由于AM采用了Zipf分布与随机部署相结合的策略，更难在边缘找到所需的微服务，因此发生了大量的迁移与云端访问。

4) 不同移动速度对算法性能的影响

为验证本文DMS算法在移动场景下的服务连续性与性能表现，本文选取用户移动速度为1.5 m/s、5 m/s



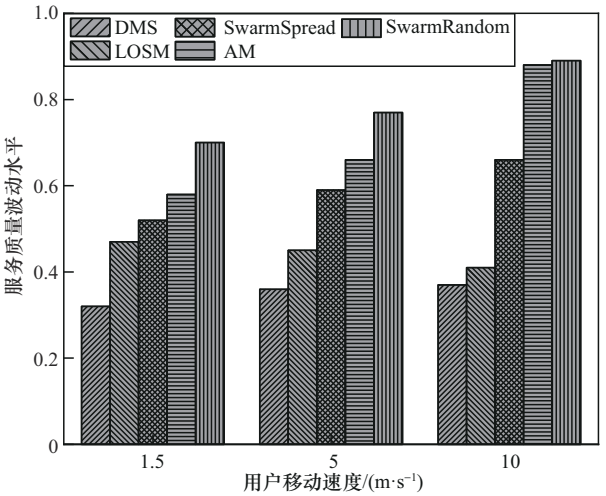
(a) 不同工作流类型对服务质量的影响

(b) 不同工作流类型对能耗的影响

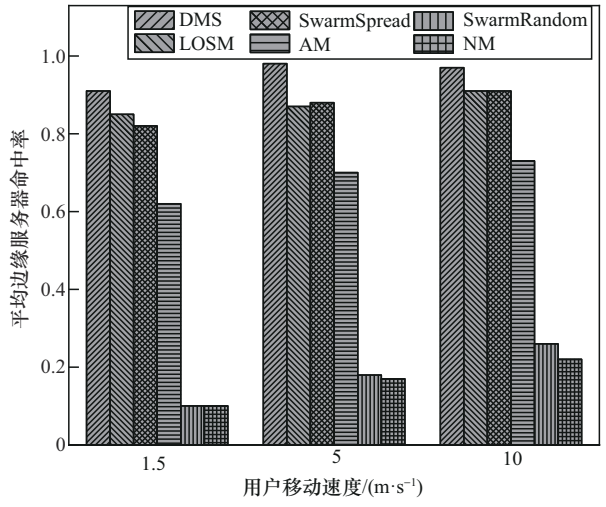
(c) 不同工作流类型对MEC命中率的影响

图7　工作流类型对算法性能的影响

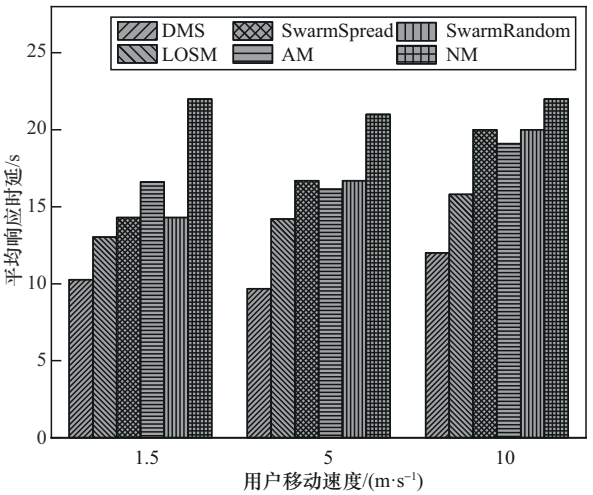
和 10 m/s 这 3 种典型条件下，对比分析各方法在服务质量波动水平、边缘服务器命中率及平均响应时延 3 个指标上的表现，实验结果如图 8 所示。



(a) 不同移动速度对服务质量波动水平的影响



(b) 不同移动速度对边缘服务器命中率的影响



(c) 不同移动速度对平均响应时延的影响

图 8　用户移动速度对算法性能的影响

由图 8(a)可知，本文 DMS 算法在所有速度条件下的服务质量波动范围最小，在移动场景下具备良好的连续服务保障能力。相比之下，AM 算法虽然在低速移动时表现尚可，但随着用户速度的提升，多次跨域导致的频繁迁移造成系统负载加重，进而导致服务质量波动显著增加；SwarmRandom 与 SwarmSpread 算法由于其调度策略缺乏针对性，在资源利用效率与时延控制方面表现一般。

图 8(b)和图 8(c)分别展示了不同算法在边缘服务器命中率与时延方面的差异。可以看出，本文 DMS 算法通过持续优化边缘节点的微服务部署策略，有效提升了命中率，降低了对云端的依赖，从而显著减少了系统整体能耗与迁移次数。同时，DMS 算法在各类速度条件下均保持较低的平均时延，尤其在中高速移动场景下优势尤为明显。这得益于 DMS 策略能够根据用户的实时位置与边缘节点的负载状况，利用云边迁移策略动态调整微服务部署位置，从而提高了任务执行效率。

5　结束语

针对用户移动中因频繁跨域和节点容量限制导致的难以选择适宜服务实例的问题，本文提出了一种结合优先级部署和动态云边调度的微服务选择算法 DMS。通过在基准工作流上的仿真实验，本文算法可以满足用户在中高速移动时的微服务迁移需求。仿真结果表明，本文 DMS 算法在并发场景下与其他算法相比，在服务质量上提升约 23.2%，在能耗方面降低约 25.4%，在时延上降低约 25.1%，在负载均衡程度上提升约 12%，在任务完成率方面提升约 7.7%。

在未来工作中，笔者将重点从 2 个方面进行深入研究：一方面，致力于降低算法的执行时间复杂度，通过优化资源分配策略和提高计算效率，增强实时性和高效性；另一方面，将更加关注车辆移动中的具体环境与实际应用场景，综合考虑网络条件、地理位置和动态负载变化等多方面因素，进一步提升算法在真实物联网环境下的适应能力和性能表现。

参考文献：

[1] DENG S G, ZHAO H L, FANG W J, et al. Edge intelligence: the confl-

uence of edge computing and artificial intelligence[J]. IEEE Internet of Things Journal, 2020, 7(8): 7457-7469.

[2] CHEN L L, XU Y C, LU Z H, et al. IoT microservice deployment in edge-cloud hybrid environment using reinforcement learning[J]. IEEE Internet of Things Journal, 2020, 8(16): 12610-12622.

[3] MAZLAMI G, CITO J, LEITNER P. Extraction of microservices from monolithic software architectures[C]//Proceedings of the 2017 IEEE International Conference on Web Services (ICWS). Piscataway: IEEE Press, 2017: 524-531.

[4] FAZIO M, CELESTI A, RANJAN R, et al. Open issues in scheduling microservices in the cloud[J]. IEEE Cloud Computing, 2016, 3(5): 81-88.

[5] WANG S Q, URGAONKAR R, HE T, et al. Dynamic service placement for mobile micro-clouds with predicted future costs[J]. IEEE Transactions on Parallel and Distributed Systems, 2017, 28(4): 1002-1016.

[6] WANG S G, XU J L, ZHANG N, et al. A survey on service migration in mobile edge computing[J]. IEEE Access, 2018, 6: 23511-23528.

[7] 彭许红. 面向微服务的服务供给与负载均衡关键技术研究[D]. 长沙: 中南大学, 2023.
PENG X H. Research on key technologies of service supply and load balancing for micro-service[D]. Changsha: Central South University, 2023.

[8] 约翰•卡内尔, 伊拉里•华卢波•桑切斯. Spring微服务实战[M]. 陈文辉, 译. 北京: 人民邮电出版社, 2022.
CARNELL J, SANCHEZ I V. Spring microservices in action[M]. CHEN W H, trans. Beijing: Posts & Telecom Press, 2022.

[9] VILLAMIZAR M, GARCÉS O, OCHOA L, et al. Infrastructure cost comparison of running web applications in the cloud using AWS lambda and monolithic and microservice architectures[C]//Proceedings of the 2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid). Piscataway: IEEE Press, 2016: 179-182.

[10] LIU C C, HUANG C C, TSENG C W, et al. Service resource management in edge computing based on microservices[C]//Proceedings of the 2019 IEEE International Conference on Smart Internet of Things (SmartIoT). Piscataway: IEEE Press, 2019: 388-392.

[11] SANTOS S, SILVA A R. Microservices identification in monolith systems: functionality redesign complexity and evaluation of similarity measures[J]. Journal of Web Engineering, 2022, 21(5): 1543-1582.

[12] HE X, TU Z Y, WAGNER M, et al. Online deployment algorithms for microservice systems with complex dependencies[J]. IEEE Transactions on Cloud Computing, 2023, 11(2): 1746-1763.

[13] ZHANG Y Q, GAN Y, DELIMITROU C. μqSim: enabling accurate and scalable simulation for interactive microservices[C]//Proceedings of the 2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS). Piscataway: IEEE Press, 2019: 212-222.

[14] NADEMBEGA A, HAFID A S, BRISEBOIS R. Mobility prediction model-based service migration procedure for follow me cloud to support QoS and QoE[C]//Proceedings of the 2016 IEEE International Conference on Communications (ICC). Piscataway: IEEE Press, 2016: 1-6.

[15] CHEN X Y, BI Y G, CHEN X P, et al. Dynamic service migration and request routing for microservice in multicell mobile-edge computing[J]. IEEE Internet of Things Journal, 2022, 9(15): 13126-13143.

[16] DOU W C, TANG W D, LIU B W, et al. Blockchain-based mobility-aware offloading mechanism for fog computing services[J]. Computer Communications, 2020, 164: 261-273.

[17] 雷鹰. 云边协同MEC环境下的任务卸载方法研究[D]. 重庆: 重庆大学, 2022.
LEI Y. Research on task unloading method in cloud edge collaborative MEC environment[D]. Chongqing: Chongqing University, 2022.

[18] PENG Q L, XIA Y N, FENG Z, et al. Mobility-aware and migration-enabled online edge user allocation in mobile edge computing[C]//Proceedings of the 2019 IEEE International Conference on Web Services (ICWS). Piscataway: IEEE Press, 2019: 91-98.

[19] OUYANG T, ZHOU Z, CHEN X. 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.

[20] PALLEWATTA S, KOSTAKOS V, BUYYA R. Microservices-based IoT application placement within heterogeneous and resource constrained fog computing environments[C]//Proceedings of the 2019 12th IEEE/ACM International Conference on Utility and Cloud Computing. New York: ACM Press, 2019: 71-81.

[21] 高思怡. 移动边缘环境下可靠性增强的服务部署方法研究[D]. 北京: 北京邮电大学, 2021.
GAO S Y. Research on service deployment method of reliability enhancement in mobile edge environment[D]. Beijing: Beijing University of Posts and Telecommunications, 2021.

[22] ZHU M, YU F L, YAN X K, et al. Scaling up mobile service selection in edge computing environment with cuckoo optimization algorithm[C]//Proceedings of the 2021 IEEE International Conference on Services Computing (SCC). Piscataway: IEEE Press, 2021: 394-400.

[23] WANG R H, LU J W. QoS-aware service discovery and selection management for cloud-edge computing using a hybrid meta-heuristic algorithm in IoT[J]. Wireless Personal Communications, 2022, 126(3): 2269-2282.

[24] 邵苏杰, 吴磊, 钟成, 等. 面向多工作流的基于容器的边缘微服务选择机制[J]. 电子与信息学报, 2022, 44(11): 3748-3756.
SHAO S J, WU L, ZHONG C, et al. Container based microservice selection for multi-workflow in edge computing paradigm[J]. Journal of Electronics & Information Technology, 2022, 44(11): 3748-3756.

[25] YU R Z, KILARI V T, XUE G L, et al. Load balancing for interdependent IoT microservices[C]//Proceedings of the IEEE INFOCOM 2019 - IEEE Conference on Computer Communications. Piscataway: IEEE Press, 2019: 298-306. [LinkOut]

[26] 张齐勋, 吴一凡, 杨勇, 等. 微服务系统服务依赖发现技术综述[J]. 软件学报, 2024, 35(1): 118-135.

ZHANG Q X, WU Y F, YANG Y, et al. Survey on service dependency technologies for microservice systems[J]. Journal of Software, 2024, 35(1): 118-135.

[27] YANG X L, FEI Z S, ZHENG J C, et al. Joint multi-user computation offloading and data caching for hybrid mobile cloud/edge computing[J]. IEEE Transactions on Vehicular Technology, 2019, 68(11): 11018-11030.

[28] JOŠILO S, DÁN G. Selfish decentralized computation offloading for mobile cloud computing in dense wireless networks[J]. IEEE Transactions on Mobile Computing, 2019, 18(1): 207-220.

[29] LIU H L, CAO L, PEI T R, et al. A fast algorithm for energy-saving offloading with reliability and latency requirements in multi-access edge computing[J]. IEEE Access, 2019, 8: 151-161.

[30] 洪陈杰. 云边协同环境下的服务部署问题研究[D]. 杭州: 杭州电子科技大学, 2023.
HONG C J. Research on service deployment in cloud-side collaborative environment[D]. Hangzhou: Hangzhou Dianzi University, 2023.

[31] TANG G M, GUO D K, WU K, et al. QoS guaranteed edge cloud resource provisioning for vehicle fleets[J]. IEEE Transactions on Vehicular Technology, 2020, 69(6): 5889-5900.

[32] ZHAO S X, CHEN X Y, WANG X L. Research on the edge resource allocation and load balancing algorithm based on vehicle trajectory[J]. Complexity, 2022(1): 1-17.

[作者简介]

赵庶旭（1976– ），男，山东青岛人，博士，兰州交通大学教授，主要研究方向为智能交通、边缘计算等。

蒋恺俊（2001– ），男，安徽合肥人，兰州交通大学硕士生，主要研究方向为边缘计算、微服务等。

王小龙（1989– ），男，甘肃定西人，博士，兰州交通大学讲师，主要研究方向为边缘计算、边缘联盟等。