计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
[doi: 10.15888/j.cnki.csa.009711] [CSTR: 32024.14.csa.009711]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

多类型任务负载预测的负载均衡任务卸载①

胡　辉, 沈　艳

(成都信息工程大学 计算机学院, 成都 610225)
通信作者: 沈　艳, E-mail: sheny@cuit.edu.cn

摘　要: 在移动边缘计算 (mobile edge computing, MEC) 背景下, 不合理的任务卸载策略和资源分配以及多类型任务数量急剧增加导致边缘服务器间的负载不均衡. 针对上述问题, 本文基于多用户多 MEC 的边缘环境, 提出一种面向多类型任务的负载预测以及均衡分配方案 (load prediction and balanced assignment scheme for multi-type tasks, LBMT). 该方案包括划分任务类型, 任务负载预测, 任务自适应映射 3 个部分. 首先, 考虑任务类型的多样性设计了任务类型模型, 利用该模型划分任务类型. 其次, 考虑不同任务对服务器造成的负载具有差异性提出了任务负载预测模型, 并在此基础上采用改进 KNN (K-nearest neighbor) 算法用于预测任务负载. 然后, 综合考虑 MEC 服务器异构性、资源有限等因素, 结合 MEC 服务器负载均衡模型设计了任务分配模型, 并提出基于自适应任务映射算法用于任务分配. 最后, LBMT 针对 MEC 服务器资源利用率和任务处理率进行优化, 得到最优负载均衡任务卸载策略. LBMT 与基于改进的 min-min 卸载方案、基于中间节点的卸载方案、基于加权二分图的卸载等方案进行仿真实验对比, 实验结果表明 LBMT 在资源利用率上提高了 12.5% 以上, 任务处理率提高了 20.3% 以上, 并显著降低了负载均衡标准差值, 更有效的实现了服务器之间的负载均衡.

关键词: 移动边缘计算; 任务卸载; 负载均衡; 负载预测; 多类型任务

引用格式: 胡辉,沈艳.多类型任务负载预测的负载均衡任务卸载.计算机系统应用. http://www.c-s-a.org.cn/1003-3254/9711.html

Load Balancing Task Offloading for Multi-type Task Load Prediction

HU Hui, SHEN Yan

(School of Computer Science, Chengdu University of Information Technology, Chengdu 610225, China)

Abstract: In mobile edge computing (MEC), load imbalance among edge servers occurs due to irrational task offloading strategies and resource allocation, as well as a sharp increase in the number of multi-type tasks. To address the above-mentioned issues, this study proposes a load prediction and balanced assignment scheme for multi-type tasks (LBMT) in a multi-user, multi-MEC edge environment. The LBMT scheme includes three components: task type classification, task load prediction, and task adaptive mapping. Firstly, considering the diversity of task types, a task type model is designed to classify tasks. Secondly, a task load prediction model is developed, considering the varying loads imposed by different tasks on servers, and employs an improved K-nearest neighbor (KNN) algorithm for load prediction. Thirdly, taking into account the heterogeneity of MEC servers and the limitation of resources, a task allocation model is designed in conjunction with a server load balancing model. Additionally, a task allocation method based on an adaptive task mapping algorithm is proposed. Finally, the LBMT scheme optimizes resource utilization and task processing rates for MEC servers to achieve the optimal load-balanced task offloading strategy. Simulation experiments compare LBMT with improved min-min offloading, intermediate node-based offloading, and weighted bipartite graph-based offloading schemes. The results show that LBMT improves the resource utilization rate by more than 12.5% and the task processing

① 基金项目: 国家自然科学基金 (62172061); 四川省揭榜挂帅项目 (2023YFG0374)
收稿时间: 2024-04-23; 修改时间: 2024-06-17; 采用时间: 2024-07-11; csa 在线出版时间: 2024-10-31

rate by more than 20.3%. Additionally, LBMT significantly reduces the standard deviation of load balancing, more effectively achieving load balance among servers.

Key words: mobile edge computing (MEC); task offloading; load balancing; load prediction; multi-type task

1 引言

随着 5G 技术和物联网 (Internet of Things, IoT) 的快速发展, 移动设备数量呈指数级增长, 物联网的应用场景越来越多样化[1]. 例如网络游戏、视频直播、虚拟现实[2]/增强现实[3]等应用产生的数据量呈爆炸式增长, 对资源的需求也各不相同, 给移动设备带来了极高的要求. 由于用户设备的条件有限, 无法完全满足这些需求[4], 云计算成为了解决这一问题的重要手段[5]. 云计算通过数据中心能够存储和处理大规模数据, 但是大量数据传输到较远的云中心处理会导致响应时间延长和能耗增加等问题. 为克服以上问题, 2014 年欧洲电信标准化协议 (ETSI) 提出了移动边缘计算 (MEC)[6]. 边缘计算作为云计算的一种演变, 将应用托管从集中的数据中心带到网络边缘, 更接近应用程序生成的数据[7]. 用户设备将任务卸载到这些边缘服务器上进行处理, 相比将任务传输至云端处理, 能够更有效地节省成本. 然而, 由于边缘服务器资源有限, 不合理的卸载策略以及资源分配可能导致出现部分服务器过载. 例如. 当某一个边缘服务器附近聚集了大量用户设备, 而这些用户设备依据就近原则同时将任务卸载到该边缘服务器时, 该服务器的资源消耗会迅速增加, 超出其承载能力, 从而导致其服务性能下降甚至中断. Fastly CDN 宕机事件、AWS edge network 宕机事件是因边缘服务器负载不均衡导致的典型事件. 因此, 边缘计算负载均衡成为边缘计算任务卸载的重要的方向之一[8].

Zhang 等人[9]研究了 MEC 系统中的并行卸载和负载均衡策略. 通过设计基于 LYP-CCMA 的集中成本管理算法和两种基于 ADMM 的分布式资源分配算法, 以优化通信和计算资源分配, 最大化协作 MEC 服务器的计算能力利用率. Bisht 等人[10]提出了一种用于工作流调度的 min-min 算法的改进版本, 该算法倾向于选择较小的任务分配到资源较丰富的节点上, 易导致资源浪费. Chang 等人[11]提出一种基于 Lyapunov 优化的卸载决策生成算法, 有效解决边缘环境中静态控制器部署方案带来的负载不均衡问题. Park 等人[12]提出了一种分布式任务重定向方法, 通过将拥塞的 MECS 任务分配给一组 MEC, 实现了 MEC 之间的负载均衡, 有效减小负载差异, 提高多访问边缘计算系统资源效率. 相较于传统方法, 该方法在高任务卸载率条件下降低了平均任务阻塞率, 且在任务数量方面表现更为优越, 但未满足延迟要求.

然而, 随着计算环境的不断演进, 不同边缘设备的计算和存储资源异质性逐渐凸显. 以上静态负载均衡算法在这种异质性环境下可能变得无法满足需求, 因此动态负载均衡在边缘服务器环境中显得愈发关键[13]. 这引发了广泛的研究热潮, 相关人员纷纷投入动态负载均衡的深入研究中, 以更灵活地适应不断变化的工作负载和异质设备的资源差异. 这一研究趋势旨在提高系统性能, 确保最佳资源利用, 以适应快速变化的边缘计算环境.

Laboni 等人[14]提出一种基于超启发式算法 (AWSH) 的高效资源分配框架, 优化 5G MEC 网络中的延迟、计算和网络负载, 解决了现有方法在应用延迟要求和计算负载均衡方面的局限性. Hui 等人[15]提出了一种工作负载迁移方案来解决负载不平衡问题, 并提高网络资源利用率. 该方案使用改进的伽马分布和二分搜索确定最佳 MEC 服务器部署密度, 最大化整个网络的资源利用率. 在动态车载网络中, Su 等人[16]提出了一种基于 Lyapunov 优化的在线动态方案, 以最大化公用事业能源效率并满足时间延迟约束, 平衡了服务器负载. 在文献 [17–19]中研究了结合改进的遗传算法解决不同场景的边缘负载均衡问题. 例如, 文献 [17]提出了一种基于整数线性规划的遗传算法和多层次控制方法, 以实现任务在虚拟机之间的负载均衡. 该方法通过优化任务分配以减少任务响应时间和提高资源利用率. Bonab 等人[20]提出了多层 NOMA HetNet 中的联合无线电资源分配和 MEC 优化, 以最大限度地提高系统的能源效率实现负载均衡. Lee 等人[21]开发了一种新颖的分布式策略, 用于联合管理任务分配和 VM 之间的卸载平衡, 旨在使用最小-最大标准最小化总体 MEC 任务的延迟, 但是仅考虑了计算密集型任务的卸载问题. Cui 等人[22]采用粒子群优化 (PSO) 和遗传模型进行编码, 提出了

一种自适应的 PSO 混合模型, 解决任务在时间序列和数据上的双重依赖关系以及计算负载不平衡而导致的依赖延迟问题. Chen 等人[23]提出一种新颖的两阶段多边缘协作负载平衡方法 (TDB-EC), 解决计算密集型任务在无线城域网中的均衡卸载问题. 班玉琦等人[24]提出了考虑设备移动轨迹的计算卸载方案, 利用凸优化和改进的 Kuhn-Munkres 算法解决复杂的任务分配问题, 以提升多设备和多 MEC 服务器场景下的用户体验质量. 彭世明等人[25]为解决车联网边缘计算中的任务卸载问题, 提出基于负载预测的多目标优化卸载策略算法, 降低任务时延并实现边缘服务器负载均衡, 该方法仅考虑计算密集型任务. Li 等人[26]提出了一种基于混合免疫鲸差分进化优化 (HIWDEO) 的多接入 MEC 计算卸载模型. 该模型结合鲸鱼优化算法和差分进化算法, 显著提高了计算资源利用效率, 并优化了服务器负载均衡. 该卸载模型没有考虑服务器异构性, 即没考虑服务器在计算能力、存储容量和读写能力等方面存在差异, 可能对卸载决策和优化结果产生重要影响.

上述方法综合考虑了边缘服务器资源的利用率和卸载成本, 制定了多种任务卸载策略与资源分配策略, 虽然实现了服务器的负载均衡, 但仍然存在以下几个问题.

(1) 策略缺少对多样化任务类型的支持. 仅考虑单一类型任务, 例如计算密集型任务, 但是在实际应用场景中任务应该分为多种类型. 不同类型任务对资源需求各不相同, 根据每种任务的具体资源需求设计合理且高效的资源分配策略, 有助于提升资源利用率和服务质量.

(2) 策略忽略任务对服务器造成的负载具有差异性. 任务对服务器造成的负载大小各异, 现今用户设备具有一定处理能力, 将负载较小的任务分配到用户设备执行, 能节约能源和成本、避免服务器资源竞争过大, 减轻服务器负载压力, 确保服务器负载均衡更优.

(3) 策略未考虑服务器的异构性. 在实际应用场景中应将服务器分为多种类型, 不同类型的服务器拥有的具体资源量不一样, 根据任务的具体资源需求选择最适合的服务器类型. 这不仅能确保资源的充分利用, 避免资源浪费, 还能增强系统对未来可能增加的多种类型任务的应对能力, 从而提高系统的灵活性与扩展性.

为解决上述问题, 本文提出了一种面向多类型任务的负载预测以及均衡分配方案 LBMT, 以最大化资源利用率和任务处理率为目标实现服务器之间的负载均衡. 本文的主要贡献如下.

(1) 考虑任务类型的多样性, 根据任务各类资源需求的差异, 建立了任务类型模型. 利用该模型精确计算不同任务类型, 从而确保能够有效处理和优化多种任务的卸载与资源分配工作.

(2) 提出一种改进 KNN 任务负载预测算法. 首先, 通过该算法预测任务对服务器所造成的负载大小. 然后, 针对用户设备处理能力的有限性, 引入用户设备与边缘服务器处理能力权重因子, 从而得到划分任务负载类型的阈值, 并结合预测的结果得到任务负载的类型.

(3) 考虑 MEC 服务器的异构性, 将 MEC 服务器分为计算密集型、内存密集型、读写密集型. 首先, 根据任务类型与任务负载的类型构建任务缓存队列. 然后, 以提高资源利用率和任务处理率为目标, 结合 MEC 负载均衡模型设计了一个任务分配模型, 并提出基于自适应任务映射算法为各类服务器分配任务缓存队列, 得到最优负载均衡任务卸载策略.

2 系统模型

如图 1 所示的网络场景, 该系统是一个基于 SDN (software-defined network) 的多用户多 MEC 系统. 通过利用 SDN 对系统进行管理, 将网络资源的智能管理功能集中在 SDN 控制器. SDN 控制器由一个基站与中心控制器构成. 该系统 SDN 主要包含控制平面和数据平面. 控制平面由 SDN 控制器构成, 数据平面由基站和 MEC 服务器构成. 控制平面负责收集所有用户设备发送的任务资源需求和 MEC 服务器发送的可用资源信息, 然后通过负载均衡算法做出决策. 数据平面根据控制平面的决策决定用户的任务将被卸载到某个 MEC 上. SDN 控制器与 MEC 服务器之间以及相邻 MEC 服务器之间通过无线链路连接.

该系统模型包含多个用户, 用户集合表示为 $E = \{E_n | n = 1, 2, \cdots, N\}$. 每个用户设备产生一个任务, 任务为多种类型, 任务执行过程不存在间断执行且任务不可分. 任务集合表示为 $D = \{d_i | i = 1, 2, \cdots, N\}$. 每一个任务可以选择在本地设备处理, 也可以选择发送到 MEC 服务器处理, 任务之间互相独立没有依赖关系. 每一个任务 d_i 都包含以下属性:

$$d_i=\left\{d_i^{\text{cpu}},d_i^{\text{memo}},d_i^{\text{io}},d_i^{\text{class}},d_i^{\text{chl}}\right\} \tag{1}$$

其中, d_i^{cpu}、d_i^{memo}、d_i^{io}分别表示任务d_i的 CPU 资源需求、内存资源需求、IO 资源需求. d_i^{class}表示任务种类类型, d_i^{chl}表示该任务负载类型.

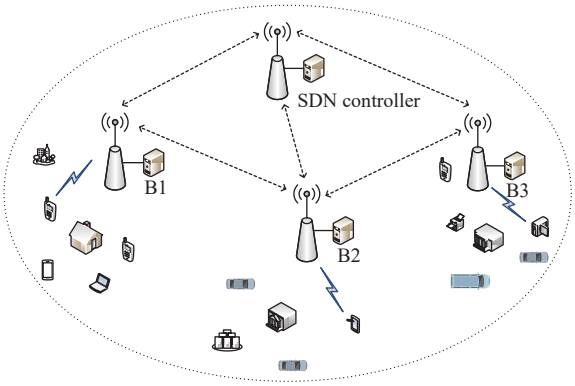

图 1 系统模型

系统模型包含多个异构的 MEC 服务器, MEC 服务器集合表示为$B=\{B_m|m=1,2,\cdots,M\}$. 每一个边缘服务器B_m都包含以下属性:

$$B_m=\left\{B_m^{\text{cpu}},B_m^{\text{memo}},B_m^{\text{io}},B_m^{\text{class}}\right\} \tag{2}$$

其中, B_m^{cpu}、B_m^{memo}、B_m^{io}分别表示服务器B_m的 CPU 可用资源、内存可用资源和 IO 可用资源, B_m^{class}表示该服务器的类型.

系统模型中, 用户设备将其任务资源需求发送到 SDN 控制器, 同时 MEC 服务器也将其可用资源信息发送到 SDN 控制器. SDN 控制器利用本文提出的 LBMT 负载均衡算法得到卸载策略, 用户设备根据卸载策略将任务发送到目标 MEC 服务器, 该 MEC 服务器为任务分配资源并处理, 最后响应处理的结果到用户设备, 并实时将其可用资源信息发送到 SDN 控制器. 任务处理流程如下.

(1) SDN 控制器收集用户设备任务的资源需求和 MEC 服务器可用资源信息.

(2) SDN 控制器利用 LBMT 分析所收集的任务资源需求和服务器可用资源信息, 制定出负载均衡任务卸载策略.

(3) 用户设备根据卸载策略, 将任务发送至目标 MEC 服务器.

(4) MEC 服务器为请求的任务分配资源, 并执行该任务. 将结果响应到用户设备, 并实时将其可用资源信息反馈给 SDN 控制器.

3 任务类型模型

由于用户设备的异构性导致所产生的任务具有多样性, 其类型各异. 例如, 车载任务多数为计算密集型, 而一些数据库事务处理多数为读写密集型. 因此, 任务类型可以分为以下 3 类, 并以d_i^{class}作为标志.

(1) 计算密集型 (CPU 密集型)

这类任务主要依赖于 CPU 的计算能力, 涉及大量的计算操作, 令$d_i^{\text{class}}=0$.

(2) 内存密集型

这类任务主要涉及大量的内存使用, 令$d_i^{\text{class}}=1$.

(3) 读写密集型 (IO 密集型)

这类任务主要涉及对设备的读取和写入操作, 令$d_i^{\text{class}}=2$.

根据d_i^{cpu}、d_i^{memo}、d_i^{io}来计算任务的类型, 任务类型模型具体表示如下.

(1) 当$d_i^{\text{cpu}}\neq d_i^{\text{memo}}\neq d_i^{\text{io}}$时:

$$d_i^{\text{class}}=\text{argmax}\left\{d_i^{\text{cpu}},d_i^{\text{memo}},d_i^{\text{io}}\right\} \tag{3}$$

(2) 当$d_i^{\text{cpu}}=d_i^{\text{memo}}>d_i^{\text{io}}$时:

$$d_i^{\text{class}}=\begin{cases}0, & \dfrac{B_{\text{all}}^{\text{cpu}}}{N\_\text{cpu}}\geqslant\dfrac{B_{\text{all}}^{\text{memo}}}{N\_\text{memo}}\\[2ex] 1, & \dfrac{B_{\text{all}}^{\text{cpu}}}{N\_\text{cpu}}<\dfrac{B_{\text{all}}^{\text{memo}}}{N\_\text{memo}}\end{cases} \tag{4}$$

其中, $B_{\text{all}}^{\text{cpu}}$、$B_{\text{all}}^{\text{memo}}$分别表示所有 MEC 服务器的 CPU 资源量、内存资源量, N_cpu、N_memo分别表示已知的计算密集型任务数量、内存密集型任务数量. 式 (4) 整体含义表示当计算密集型任务得到的 CPU 资源量比内存密集型任务得到的内存资源量多时, 意味着 MEC 服务器能分配的 CPU 资源量比内存资源量多, 任务应为计算密集型任务, 否则为内存密集型任务.

(3) 当$d_i^{\text{cpu}}=d_i^{\text{io}}>d_i^{\text{memo}}$时:

$$d_i^{\text{class}}=\begin{cases}0, & \dfrac{B_{\text{all}}^{\text{cpu}}}{N\_\text{cpu}}\geqslant\dfrac{B_{\text{all}}^{\text{io}}}{N\_\text{io}}\\[2ex] 2, & \dfrac{B_{\text{all}}^{\text{cpu}}}{N\_\text{cpu}}<\dfrac{B_{\text{all}}^{\text{io}}}{N\_\text{io}}\end{cases} \tag{5}$$

其中, $B_{\text{all}}^{\text{io}}$表示所有 MEC 服务器的 IO 资源量, N_io表示已知的读写密集型任务数量. 式 (5) 整体含义表示 MEC 服务器能分配的 CPU 资源量比 IO 资源量多, 任务应为计算密集型任务, 否则为读写密集型任务.

(4) 当$d_i^{\text{memo}} = d_i^{\text{io}} > d_i^{\text{cpu}}$时:

$$d_i^{\text{class}} = \begin{cases} 1, & \dfrac{B_{\text{all}}^{\text{memo}}}{N\_\text{memo}} \geqslant \dfrac{B_{\text{all}}^{\text{io}}}{N\_\text{io}} \\ 2, & \dfrac{B_{\text{all}}^{\text{memo}}}{N\_\text{memo}} < \dfrac{B_{\text{all}}^{\text{io}}}{N\_\text{io}} \end{cases} \tag{6}$$

式 (6) 整体含义表示 MEC 服务器能分配的内存资源量比 IO 资源量多, 任务应为内存密集型任务, 否则为读写密集型任务.

(5) 当$d_i^{\text{cpu}} = d_i^{\text{memo}} = d_i^{\text{io}}$时:

在异构环境中, 用户设备通常趋向于某一类型, 各类资源需求量极少出现此类情况, 可以忽略此情况带来的影响. 因此, d_i^{class}在{0, 1, 2}中取随机值.

$$d_i^{\text{class}} \sim \{0, 1, 2\} \tag{7}$$

4 负载预测模型及算法

4.1 高负载任务与低负载任务定义

不同的任务对计算资源、内存资源以及读写资源的需求各异. 为了缓解 MEC 服务器的负载压力, 有效利用本地设备资源, 将任务负载分为高负载与低负载两种类型, 将高负载任务卸载到 MEC 服务器上执行, 低负载任务分配到本地用户设备执行. 任务的负载类型可以表示为:

$$d_i^{\text{chl}} = \begin{cases} 1, & \text{if } dl_i \in [\lambda, 1] \\ -1, & \text{if } dl_i \in [-1, \lambda) \end{cases} \tag{8}$$

其中, dl_i为任务负载, 即任务对服务器、用户设备造成的负载大小. λ表示划分任务负载类型的阈值, 可表示为:

$$\lambda = -1 \cdot a + 1 \cdot b \tag{9}$$

其中, a 和 b 分别表示 MEC 服务器处理任务能力和用户设备处理任务能力的权重因子. 引入λ的原因是需要考虑用户设备处理能力有限, 当用户设备无法处理低负载任务时, 可以通过设置 a、b 权重因子的大小更新阈值λ, 确保分配给用户设备的低负载任务在用户设备处理能力范围之内, 用户设备能处理该任务.

根据式 (8), dl_i在区间$[-1, \lambda]$内的任务定义为低负载任务, 在区间$[\lambda, 1]$内的任务定义为高负载任务.

4.2 任务负载预测模型

4.2.1 任务负载预测

在系统模型中, SDN 控制器作用之一是负责收集各个任务的资源需求, 由于任务尚未处理, SDN 控制器对任务负载未知. 因此, 利用历史任务集建立任务负载预测模型. 历史任务集存储在 SDN 控制器, 可表示为$H = \{h_j | j = 1, 2, \cdots, J\}$, h_j有以下属性:

$$h_j = \{h_j^{\text{cpu}}, h_j^{\text{memo}}, h_j^{\text{io}}, h_j^{\text{cp}}\} \tag{10}$$

其中, h_j^{cpu}、h_j^{memo}、h_j^{io}分别表示历史任务的 CPU 资源需求量、内存资源需求量和 IO 资源需求量, h_j^{cp}表示历史任务的负载大小, 可表示为:

$$h_j^{\text{cp}} = \begin{cases} 1, & \text{卸载到MEC服务器的历史任务} \\ -1, & \text{留在用户设备处理的历史任务} \end{cases} \tag{11}$$

任务负载预测模型表示为:

$$dl_i = \frac{\sum_{j=1}^{k} SD_i^j \cdot h_j^{\text{cp}}}{\sum_{j=1}^{k} w_i^j} = \frac{\sum_{j=1}^{k} \frac{1}{Dc_i^j} \cdot h_j^{\text{cp}}}{\sum_{j=1}^{k} \frac{1}{Dc_i^j}} \tag{12}$$

其中, SD_i^j表示当前任务d_i与历史任务h_j之间的相似度. Dc_i^j表示当前任务d_i与历史任务h_j之间的距离. k 表示选择当前任务d_i与历史任务h_j距离最小的前 k 个历史任务. 相似度SD_i^j具体表示为:

$$SD_i^j = \frac{1}{Dc_i^j} \tag{13}$$

当距离Dc_i^j越小, 任务d_i与h_j之间的相似度就越高, 代表该h_j对d_i的影响越大. 距离Dc_i^j具体表示为:

$$Dc_i^j = \left[\left(d_i^{\text{cpu}} - h_j^{\text{cpu}}\right)^2 + \left(d_i^{\text{memo}} - h_j^{\text{memo}}\right)^2 + \left(d_i^{\text{io}} - h_j^{\text{io}}\right)^2 \right]^{\frac{1}{2}} \tag{14}$$

式 (12) 的整体含义: 该预测模型将历史任务作为训练集, 计算任务d_i与所有历史任务的相似度, 然后选择前 k 个相似度最大的历史任务, 根据这 k 个历史任务的负载大小h_j^{cp}与其相似度SD_i^j得到任务d_i的负载大小dl_i, dl_i值的范围在 [−1, 1].

4.2.2 预测模型评估

本文需要对预测结果进行评估, 通过预测模型得到任务的预测负载目的是根据该预测负载值计算出任务负载类型, 结果一共分为以下 4 类.

(1) *TP*: 高负载任务正确预测为高负载任务的数量.

(2) *FN*: 高负载任务错误预测为非高负载任务的数量.

(3) *TN*: 非高负载任务正确预测为非高负载任务的数量.

(4) *FP*: 非高负载任务错误预测为高负载任务的数量.

以上 4 类可得到混淆矩阵 *L* 表示为:

$$L=\begin{bmatrix} TP & FP \\ FN & TN \end{bmatrix} \tag{15}$$

基于该混淆矩阵 *L*, 任务负载预测模型采用 *Kappa* 系数来评估预测结果, *Kappa* 越接近 1 反映预测模型性能越优. *Kappa* 系数具体表示为:

$$\begin{cases} Kappa=\dfrac{p_0-p_e}{1-p_e} \\ p_0=\dfrac{TP+TN}{TP+TN+FP+FN} \\ p_e=\dfrac{(TP+FN)\cdot(TP+FP)+(FP+TN)\cdot(FN+TN)}{(TP+TN+FP+FN)^2} \end{cases} \tag{16}$$

其中, p_0为实际准确率, p_e为随机情况下的准确率.

4.3 改进 KNN 任务负载预测算法

各类用户终端设备已具备处理低负载任务的能力, 仅将高负载任务卸载到边缘服务器进行处理, 有助于缓解边缘服务器压力、降低资源竞争, 减小负载不均衡风险, 提升系统整体性能和用户体验. 因此, 迫切需要建立可靠的负载预测算法, 预测任务的负载大小并得到高负载任务.

KNN 是一种基于实例的非参数监督机器学习算法, 常用于预测计算. 传统方法如线性回归对异常值非常敏感, 在边缘环境中容易因为异常数据导致较大预测误差. 相比之下, 机器学习方法如神经网络虽然强大, 但训练过程耗时且需要大量计算资源, 特别是深度神经网络. 而且参数调优复杂, 数据更新时需要重新训练, 增加了计算资源和时间成本, 不适合资源有限的边缘服务器, 容易造成服务器负载过大, 影响负载均衡. KNN 无需复杂的训练过程, 也不需要模型参数调整, 在边缘设备上部署时, 可以实时快速地更新和预测, 而无需重新训练模型. 在数据分布未知或复杂等情况下, KNN 的准确率、精确度等得分方面比其他方法如朴素贝叶斯、支持向量机、决策树表现得更好[27]. 因此, KNN 在边缘计算环境中非常高效和实用, 能够在有限的资源下保持良好的性能.

但是现有的 KNN 算法普遍忽视了历史数据与输入数据之间不同距离对 KNN 算法准确度的影响, 在进行预测时, 基于前 *k* 个最近邻样本的标签进行投票, 选择得票最多的类别作为未知样本的预测标签, 导致预测结果的准确性差且对近邻样本十分敏感.

在第 4.2 节任务负载预测模型基础上, 设计改进 KNN 任务负载预测算法用于预测任务负载, 提高预测准确性, 并根据预测结果计算任务负载类型, 最终得到高负载任务. 这一方法有望在提高边缘服务器效能的同时, 减小 MEC 服务器负载压力, 有效降低整体系统负载不均衡的风险. 本文对 KNN 算法的改进如下.

(1) 考虑到样本间距离带来的影响, 引入样本与未知样本之间的相似度来提升预测的准确度. 具体而言, 通过式 (13) 计算相似度. 再利用式 (12) 计算前 *k* 个最近邻历史任务的相似度负载和的平均值作为未知样本 (任务集 *D*) 的任务负载预测值.

(2) 根据任务负载预测值计算任务负载类型时, 为了真实贴近边缘设备环境, 考虑到用户设备处理能力有限的情况, 引入用户设备与服务器设备处理能力的权重, 并利用式 (8) 计算出任务负载类型.

通过以下两种不同情况分析 KNN 算法改进的原因.

如图 2 所示, *k*=6, 假设式 (9) 计算得到$\lambda=0$. 距待测任务d_i最近的这 6 个历史任务中, 有 4 个为低负载历史任务 ($h_j^{\mathrm{cp}}=-1$), 2 个为高负载历史任务 ($h_j^{\mathrm{cp}}=1$). 根据距离公式 (14) 计算待测任务d_i与低负载历史任务的距离都为 0.7, 与高负载历史任务的距离都为 0.5. 通过式 (12) 计算得到任务负载预测值$dl_i=-0.176$, 根据式 (8) 计算出$d_i^{\mathrm{chl}}=-1$, 待测任务为低负载任务.

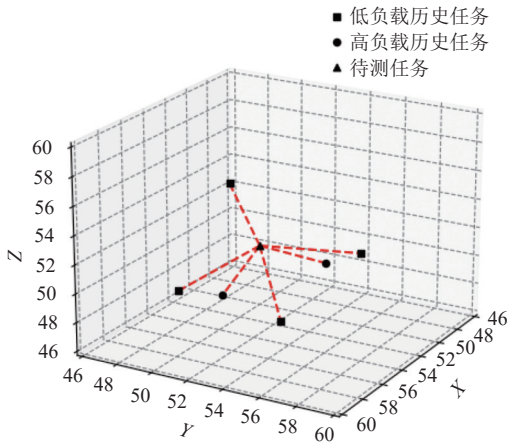

图 2 预测情况 a

如图 3 所示, 设置与图 2 相同的条件, *k*=6, 假设式 (9) 计算得到$\lambda=0$. 距待测任务d_i最近的这 6 个历史任务中, 有 4 个为低负载历史任务, 2 个为高负载历史

任务. 根据距离式 (14) 计算待测任务d_i与低负载历史任务的距离都为 0.7, 与高负载历史任务的距离都为 0.2. 通过式 (12) 计算得到负载预测值$dl_i = 0.273$, 根据式 (8) 计算出$d_i^{\text{chl}} = 1$, 待测任务为高负载任务.

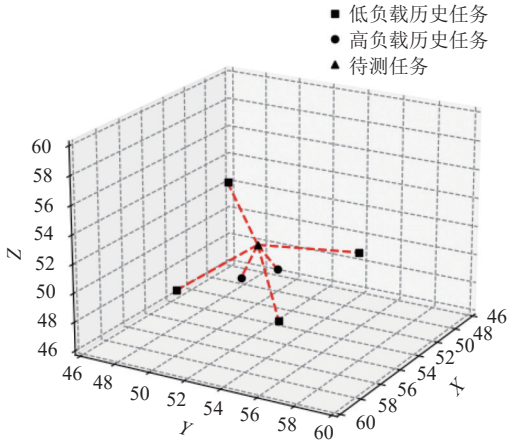

图 3 预测情况 b

上述分析的两种情况中, 高负载历史任务与待测任务的距离不一样. 若使用未改进的 KNN 算法, 两种情况的预测结果均为高负载任务, 这显然忽略了距离的影响. 待预测结果与相似度有关, 不能仅简单地利用前 k 个数量最多的历史任务的类型作为预测结果, 因为容易受到噪声数据的影响而导致预测误差过大. 通过本文的改进, 当某类历史任务与待测任务之间的距离越近时, 它们的相似度也越高, 这意味着该类历史任务对待测任务的影响越大. 因此, 预测结果受相似度影响. 在图 3 情况下, 尽管低负载历史任务数量为高负载历史任务数量的两倍, 但由于所有低负载历史任务对待测任务的影响力不及所有高负载历史任务, 最终预测结果为高负载任务.

改进 KNN 任务负载预测算法分为 4 个步骤. 第 1 步对用户任务集和历史任务集进行均值方差归一化, 并初始化历史任务负载h_j^{cp}. 第 2 步预测任务负载. 计算与当前任务距离最近的前 k 个历史任务的相似度SD_i^j, 利用h_j^{cp}与SD_i^j通过式 (12) 计算出任务负载预测值dl_i. 第 3 步根据用户设备处理能力权重 a 和服务器处理能力权重 b 计算出划分任务负载类型的阈值λ, 再结合任务负载预测值dl_i得到任务负载类型d_i^{chl}. 第 4 步检查停止条件, 输出任务负载类型. 改进 KNN 任务负载预测算法具体如算法 1 所示.

算法 1. 改进 KNN 任务负载预测算法

输入: 任务集 D, 近邻数 k, 历史任务集 H.
输出: 任务负载类型d_i^{chl}.

步骤 1. 数据准备
(1) 对用户任务集 D、历史任务集 H 进行归一化处理.
(2) 根据式 (11) 初始化历史任务负载h_j^{cp}.
步骤 2. 任务负载预测
(1) for d_i in D
(2) for h_j in H
(3) 根据式 (14) 计算任务d_i与所有历史任务的距离Dc_i^j.
(4) end for
(5) 对距离Dc_i^j从小到大排序并选取前 k 个最短距离的历史任务, M_k=$sort\_min(Dc_i^j)$.
(6) 根据式 (13) 计算任务d_i与前 k 个最短距离的历史任务 M_k 的相似度SD_i^j, 结合历史任务负载h_j^{cp}, 根据式 (12) 计算任务负载预测值dl_i.
(7) end for
步骤 3. 计算任务负载类型
(1) 初始化用户设备与服务器处理能力权重 a、b.
(2) 使用 a、b 权重并根据式 (9) 计算区间变量λ.
(3) 如果dl_i大于或等于λ, 任务负载类型d_i^{chl}=1. 如果dl_i小于λ, 任务负载类型d_i^{chl}=−1. 如下所示:

```
if dl_i≥λ then
    d_i^chl=1
else
    d_i^chl=−1
end if
```

步骤 4. 输出任务负载类型d_i^{chl}.

5 任务分配模型及算法

5.1 MEC 负载均衡模型

MEC 服务器因其资源利用情况不同而产生不同的负载, 一部分服务器资源利用过高, 而另一部分服务器资源利用又过低, 这样造成整体负载不均衡. 本文通过负载均衡标准差σ对 MEC 服务器的整体负载情况进行定量分析, 该标准差可以反映 MEC 服务器负载的离散程度, 负载均衡标准差越小代表 MEC 服务器之间的负载越均衡. MEC 负载均衡模型具体表示为:

$$\sigma = \sqrt{\frac{\sum_{m=1}^{M}\left[\left(B_m^{\text{cpu}}-\overline{B_{\text{cpu}}}\right)^2+\left(B_m^{\text{memo}}-\overline{B_{\text{memo}}}\right)^2+\left(B_m^{\text{io}}-\overline{B_{\text{io}}}\right)^2\right]}{M}} \tag{17}$$

其中, $\overline{B_{\text{cpu}}}$、$\overline{B_{\text{memo}}}$、$\overline{B_{\text{io}}}$分别表示所有服务器的 CPU 资源均值、内存资源均值、IO 资源均值, M 为服务器数. $\overline{B_{\text{cpu}}}$、$\overline{B_{\text{memo}}}$、$\overline{B_{\text{io}}}$的具体表示为:

$$\begin{cases} \overline{B_{\text{cpu}}} = \dfrac{\sum_{m=1}^{M} B_m^{\text{cpu}}}{M} \\ \overline{B_{\text{memo}}} = \dfrac{\sum_{m=1}^{M} B_m^{\text{memo}}}{M} \\ \overline{B_{\text{io}}} = \dfrac{\sum_{m=1}^{M} B_m^{\text{io}}}{M} \end{cases} \tag{18}$$

5.2 任务分配模型

为缓解服务器压力, 防止因大量任务卸载速率与边缘服务器处理任务速率不匹配而造成服务器瞬间过载, 避免服务器性能降低和影响负载均衡. 通过任务类型与任务负载的类型构建任务缓存队列, 该缓存队列由 SDN 控制器管理. 任务缓存队列具体表示为:

$$Q = \begin{cases} Q_{\text{cpu}}, & d_i^{\text{class}} = 0, d_i^{\text{chl}} = 1 \\ Q_{\text{memo}}, & d_i^{\text{class}} = 1, d_i^{\text{chl}} = 1 \\ Q_{\text{io}}, & d_i^{\text{class}} = 2, d_i^{\text{chl}} = 1 \end{cases} \tag{19}$$

其中, Q_{cpu}、Q_{memo}、Q_{io}分别为 CPU 任务缓存队列、内存任务缓存队列、IO 任务缓存队列. $d_i^{\text{class}} = 0, d_i^{\text{chl}} = 1$属于计算密集型任务中的高负载任务, 将其缓存在$Q_{\text{cpu}}$队列; $d_i^{\text{class}} = 1, d_i^{\text{chl}} = 1$属于内存密集型任务中的高负载任务, 将其缓存在$Q_{\text{memo}}$队列; $d_i^{\text{class}} = 2, d_i^{\text{chl}} = 1$属于读写密集型任务中的高负载任务, 将其缓存在$Q_{\text{io}}$队列.

MEC 服务器异构且资源有限, 缓存队列里的任务需要高效合理地被分配到不同类型的 MEC 服务器上, 以实现 MEC 服务器之间的负载均衡最优、资源利用率最高和任务处理率最大. SDN 控制器实时收集 MEC 服务器可用资源信息, 并计算 MEC 服务器类型, MEC 服务器类型计算公式为:

$$B_m^{\text{class}} = \text{argmax}\left\{B_m^{\text{cpu}}, B_m^{\text{memo}}, B_m^{\text{io}}\right\} \tag{20}$$

当 MEC 服务器处理缓存队列里的任务时, 需要消耗不同类型的资源. 因此, MEC 服务器资源的消耗具体表示为:

$$\begin{aligned} Rc_m = & \frac{\left|B_m^{\text{class}} - 2\right| \cdot \left|B_m^{\text{class}} - 1\right|}{2} \cdot Q_{\text{cpu}} \\ & + B_m^{\text{class}} \cdot \left|B_m^{\text{class}} - 2\right| \cdot Q_{\text{memo}} \\ & + B_m^{\text{class}} \cdot \frac{\left|B_m^{\text{class}} - 1\right|}{2} \cdot Q_{\text{io}} \end{aligned} \tag{21}$$

$$\begin{cases} B_m^{\text{cpu}} = B_m^{\text{cpu}} - Rc_m^{\text{cpu}} \ \left(B_m^{\text{cpu}} > Rc_m^{\text{cpu}}\right) \\ B_m^{\text{memo}} = B_m^{\text{memo}} - Rc_m^{\text{memo}} \ \left(B_m^{\text{memo}} > Rc_m^{\text{memo}}\right) \\ B_m^{\text{io}} = B_m^{\text{io}} - Rc_m^{\text{io}} \ \left(B_m^{\text{io}} > Rc_m^{\text{io}}\right) \end{cases} \tag{22}$$

其中, 式 (21) 表示根据第 m 个服务器类型计算出该服务器需要处理的任务, 此任务用Rc_m表示. 式 (21) 具体含义为: 当$B_m^{\text{class}} = 0$时, 服务器为计算密集型, 处理Q_{cpu}队列的任务. 当$B_m^{\text{class}} = 1$时, 服务器为内存密集型, 处理Q_{memo}队列的任务. 当$B_m^{\text{class}} = 2$时, 服务器为读写密集型, 处理Q_{io}队列的任务. 式 (22) 表示更新服务器资源消耗, Rc_m^{cpu}、Rc_m^{memo}、Rc_m^{io}分别为Rc_m的 CPU、内存、IO 资源量.

MEC 服务器集群的资源利用率可以表示为:

$$R_t = \frac{\sum_{m=1}^{M} \dfrac{\left(R_m^{\text{cpu}} - B_m^{\text{cpu}}\right)}{R_m^{\text{cpu}}} + \dfrac{\left(R_m^{\text{memo}} - B_m^{\text{memo}}\right)}{R_m^{\text{memo}}} + \dfrac{\left(R_m^{\text{io}} - B_m^{\text{io}}\right)}{R_m^{\text{io}}}}{3 \cdot M} \tag{23}$$

其中, R_m^{cpu}表示B_m服务器初始的 CPU 资源, R_m^{memo}表示B_m服务器初始的内存资源, R_m^{io}表示B_m服务器初始的 IO 资源.

在平衡 MEC 服务器负载过程中, 需要使 MEC 服务器的任务处理率最大. 任务处理率表示所有 MEC 服务器能够同时处理最多任务数的能力. 任务处理率越大, 代表服务器能一次性处理的任务越多, 服务器执行效率越高. MEC 服务器任务处理率具体表示为:

$$T_\mu = \frac{\sum_{m=1}^{M} n(B_m)}{\left|Q_{\text{cpu}}\right| + \left|Q_{\text{memo}}\right| + \left|Q_{\text{io}}\right|} \tag{24}$$

其中, $n(B_m)$为B_m服务器处理的任务数量, $\left|Q_{\text{cpu}}\right|$、$\left|Q_{\text{memo}}\right|$、$\left|Q_{\text{io}}\right|$分别表示任务队列$Q_{\text{cpu}}$、$Q_{\text{memo}}$、$Q_{\text{io}}$的长度.

在任务分配过程中, 旨在实现 MEC 服务器负载均衡, 并以资源利用率最高和任务处理率最大为目标, 将缓存队列的任务高效合理地分配到 MEC 服务器. 任务分配模型可具体表示为:

$$\min \sigma, \max R_t, \max T_\mu \tag{25}$$

s.t.

$$Rc_m^{\text{total}} \leqslant B_m^{\text{total}} \tag{26}$$

$$L' \leqslant \left|Q_{\text{cpu}}\right| + \left|Q_{\text{memo}}\right| + \left|Q_{\text{io}}\right| \leqslant N \tag{27}$$

$$\sum_{m=1}^{M} n(B_m) \leqslant L' \tag{28}$$

$$B_m^{\text{cpu}} \neq B_m^{\text{memo}} \neq B_m^{\text{io}} \tag{29}$$

$$B_m^{\text{class}} \in \{0,1,2\} \tag{30}$$

式 (26) 表示卸载给服务器的任务总资源需求量不应该超过该服务器资源总拥有量. 式 (27) 为任务缓存队列有足够容量存储高负载任务, L' 为高负载任务数. 式 (28) 为所有服务器同时处理的任务数应该小于或者等于高负载任务总数. 式 (29) 表示 MEC 服务器各类资源保持不相等. 式 (30) 表示 MEC 服务器类型有 3 类, 且只能为其中某一类.

5.3 基于自适应任务映射算法

针对任务缓存队列 Q_{cpu}、Q_{memo} 和 Q_{io}, 需要设计一种合理的算法, 将任务缓存队列分配给 MEC 服务器. 需要考虑以下几点.

(1) MEC 服务器异构性

MEC 服务器分为: 计算密集型、内存密集型和读写密集型. 服务器应根据自身类型有针对性地处理对应类型的任务缓存队列. 以内存密集型服务器为例, 其内存资源相对更丰富, 而 CPU 资源相对较少. Q_{memo} 的任务对内存资源需求较高, 其他资源需求量相对较低, 所以该 MEC 服务器应处理 Q_{memo} 的任务.

(2) MEC 服务器资源有限性

MEC 服务器所拥有的资源有限, 需要充分利用服务器资源, 最大化提升服务器的任务处理效率.

(3) MEC 服务器类型变化性

MEC 服务器类型随着资源使用情况的变化而发生变化. 因此, SDN 控制器实时收集服务器可用资源信息, 并动态更新服务器类型, 以确保系统能够灵活应对不同的负载情况和任务资源需求. 这样的动态更新有助于更真实合理地处理任务, 使服务器能够更有效地适应不同类型任务的变化, 从而进一步提升整体性能.

(4) MEC 服务器资源共享

当 MEC 服务器通过映射关系处理的某类任务缓存队列为空时, 该服务器应去处理其他类型的非空任务缓存队列, 与其他服务器共享资源. 能避免资源浪费, 出现一部分服务器资源不足, 另一部分服务器资源空闲的情况. 同时也能减轻其他服务器负载以及缩短任务缓存队列的排队时间.

上述 4 点因素涵盖了任务缓存队列与 MEC 服务器之间的映射关系, 同时考虑了自适应动态环境变化对映射关系的调整. 基于此, 结合 MEC 服务器负载均衡模型与任务分配模型, 提出基于自适应任务映射算法, 以资源利用率最高和任务处理率最大为目标, 动态实现 MEC 服务器负载均衡.

基于自适应任务映射算法分为 4 个步骤. 首先, 第 1 步通过任务类型模型以及调用算法 1 的结果初始化任务缓存队列. 然后, 第 2 步根据实时更新服务器的类型情况, 动态地调整任务缓存队列与服务器之间的映射关系, 以及共享服务器之间的资源. 接着, 第 3 步评估负载均衡标准差、服务器资源利用率以及任务处理效率等指标. 最后, 第 4 步检查停止条件, 并返回负载均衡标准差、服务器资源利用率、任务处理效率以及最优的决策解. 基于自适应任务映射算法具体如算法 2 所示.

算法 2. 基于自适应任务映射算法

输入: Q_{cpu}、Q_{memo}、Q_{io}, MEC 服务器集合 B.

输出: 负载均衡标准差 σ, 资源利用率 R_t, 任务处理率 T_μ, 最优决策数组 Z.

步骤 1. 数据准备

(1.1) 通过任务类型模型计算任务类型 d_i^{class}, 调用算法 1 预测任务负载大小并计算出负载类型 d_i^{chl}.

(1.2) 初始化缓存队列 Q_{cpu}、Q_{memo}、Q_{io}

```
for d_i in D
  if d_i^class == 0 and d_i^chl == 1 then
    Q_cpu.push(d_i)
  else if d_i^class == 1 and d_i^chl == 1 then
    Q_memo.push(d_i)
  else if d_i^class == 2 and d_i^chl == 1 then
    Q_io.push(d_i)
  end if
end for
```

步骤 2. 自适应映射关系调整与服务器资源共享

(2.1) 在每次迭代时, 计算经过上一次迭代更新资源消耗后的服务器类型 B_m^{class}. 如果前后两次迭代的 B_m^{class} 一样, 则映射关系保持不变, 否则参照以下映射关系改变:

```
c'=B_m^class //获取当前服务器类型
if (Q_cpu is empty and c'==0) then
  c'=1 or c'=2
else if (Q_memo is empty and c'==1) then
  c'=0 or c'=2
else if (Q_io is empty and c'==2) then
  c'=0 or c'=1
end if
//任务分配和执行
if (c'==0 and Q_cpu not empty and B_m availabe) then
```

$Q_{\text{cpu}} \leftarrow B_m$

if (c' = = 1 and Q_{memo} not empty and B_m availabe) then

$Q_{\text{memo}} \leftarrow B_m$

if (c' = = 2 and Q_{io} not empty and B_m availabe) then

$Q_{\text{io}} \leftarrow B_m$

end if

(2.2) 任务分配后, 根据式 (22) 更新服务器 CPU 资源B_m^{cpu}、内存资源B_m^{memo}、IO 资源B_m^{io}. 根据式 (20) 更新B_m^{class}.

(2.3) 保存分配决策数组: $Z_{c'}^{k}$=*vector*(m), k 为迭代数.

步骤 3. 根据式 (17)、式 (23)、式 (24) 分别计算负载均衡标准差σ、服务器资源利用率R_t、任务处理效率T_μ.

步骤 4. 检查停止条件. 返回负载均衡标准差、服务器资源利用率、任务处理效率和最优决策解.

6 实验结果与分析

6.1 实验准备

仿真实验在内存为 8 GB、处理器为 Intel(R) Core (TM) i7-7700 HQ CPU @ 2.80 GHz 的 Windows 10 操作系统下进行. 本文在多个用户与多个 MEC 的异构场景中, 为了更真实地模拟任务与服务器类型的异构性, 并确保各种类任务的数量、各种类服务器的数量保持平衡, 减小实验误差, 任务资源需求量和服务器可用资源量在给定范围内随机选取. 本文实验具体参数配置见表 1.

6.2 对比方案

为了评估本文提出的基于多类型任务预测的负载均衡任务卸载策略的有效性, 采用以下方案与本文方案进行对比.

(1) MMF (min-min scheduling algorithm based on FCM clustering)[10]. 在原方案上做了改进, 加入了划分任务类型后的任务缓存队列, 然后使用基于工作流调度的 min-min 算法卸载任务.

(2) Random (random offloading of tasks). 该方案随机卸载任务到服务器.

(3) LBA-EC (load balancing algorithm based on weighted bipartite graph for edge computing)[28]. 该方案分为两个阶段. 第 1 阶段, 任务被匹配到不同的边缘服务器. 第 2 阶段, 将任务优化分配到边缘服务器的不同容器中执行.

(4) TLIN (task allocation and load balancing based on intermediate nodes)[29]. 该方案根据边缘节点的固有属性和实时属性, 将边缘节点分为轻载、正常载和重载 3 种类型. 然后提出了一个任务分配模型, 将新的任务分配给负载相对最轻的节点.

(5) BMT (balanced assignment scheme for multi-type tasks, BMT). 面向多类型任务的均衡分配方案, 该方案指本文 LBMT 中没有考虑负载预测.

(6) LBMT-WQ (load prediction and balanced assignment scheme for multi-type tasks without cached queues, LBMT-WQ). LBMT-WQ 指本文 LBMT 中没有设计任务缓存队列.

表 1 实验参数设置

参数	值	含义
M	35	服务器数
N	330	用户数
d_i^{cpu}	[0, 100]	任务CPU需求量
d_i^{memo}	[0, 100]	任务内存需求量
d_i^{io}	[0, 100]	任务IO需求量
h_j^{cp}	{−1, 1}	历史任务负载
k	3, 4, 5, 6, 7, 8, 9	近邻数
B_m^{cpu}	[300, 500]	服务器CPU可用量
B_m^{memo}	[300, 500]	服务器内存可用量
B_m^{io}	[300, 500]	服务器IO可用量

6.3 实验结果分析

表 2 显示不同的 MEC 服务器处理能力权重 a 和用户设备处理能力权重 b 对预测结果的影响. *Kappa* 系数用来评估预测结果, *Kappa* 越接近 1 代表预测结果越优, 同时也反映 a、b 权重越接近 MEC 服务器与用户设备的处理能力, 留在用户设备的任务在用户设备的处理能力范围之内. 本组实验通过设置不同的 a、b 权重计算 *Kappa* 系数, 得到 *Kappa* 最接近 1 的那组 a、b 权重.

从表 2 的结果看出当$a < b$时, *Kappa* 较小, 是因为 MEC 服务器处理能力通常比用户设备处理能力大, 但是设置的 MEC 服务器处理能力权重比用户设备处理能力小, 用户设备无法处理留在本地的任务, 所以造成预测模型结果与真实结果差距较大. 随着不断增加 MEC 服务器处理能力权重 a, 同时减小用户设备处理能力的权重 b. 当$a = 0.7, b = 0.3$时, *Kappa* 系数最大, 得出$a = 0.7, b = 0.3$时预测模型最优. 在$a = 0.8, b = 0.2$时, *Kappa* 系数减小. *Kappa* 减小是因为低估了用户设备处理能力, 没有更好利用本地设备资源, 导致用户设备本可以处理的低负载任务计算成高负载任务, 影响预测结果.

表 2 a、b 权重分析

a	b	*Kappa*
0.3	0.7	0.31
0.4	0.6	0.46
0.5	0.5	0.73
0.6	0.4	0.85
0.7	0.3	0.97
0.8	0.2	0.89

图 4 显示基于改进 KNN 任务负载预测算法中, 不同的 k 近邻数对负载均衡标准差的影响. k 值过大或者过小都可能导致不理想的结果. 如果预测不理想导致大量低资源需求任务被卸载到 MEC 服务器, 会造成 MEC 服务器过载的问题. 当 k 值过小时, 预测模型会对近邻的实例点非常敏感, 如果实例点恰好是噪声, 预测结果就会出错. 如果 k 值较大, 会增加近邻范围内噪声的数量, 忽略数据局部结构, 降低预测的准确性. 图 4 中显示最优 k 值等于 6, 因此本实验选择 k 近邻数为 6.

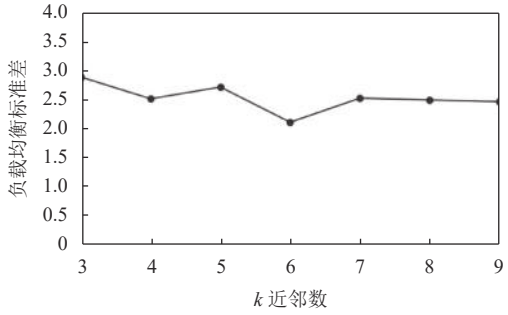

图 4 k 近邻数对负载均衡标准差影响

如图 5 所示, 本组消融实验对比了本文方案 LBMT 与 BMT、LBMT-WQ 在不同边缘服务器数下负载均衡标准差变化情况. 该实验中, 当边缘服务器数一定时, 负载均衡标准差越小, 代表整体服务器之间的负载越均衡. 横向分析, 随着边缘服务器数增加, 负载均衡标准差变化程度反映服务器之间负载均衡的稳定程度. 图 5 中显示 3 种方案的负载均衡标准差呈现出整体下降的趋势. 因为随着边缘服务器数增加, 整体服务器处理能力会提高, 在均衡分配任务时, 能为任务提供更多的资源选择. 本文方案 LBMT 在负载均衡以及负载均衡稳定性方面都优于其他两种方案. BMT 负载均衡标准差最大是因为没有考虑任务负载预测, 将低负载任务卸载到边缘服务器会加重服务器负载压力, 导致服务器性能下降, 甚至出现服务中断, 同时资源竞争激烈, 任务被均衡卸载的机会减少. LBMT-WQ 没有考虑任务缓存队列, 当大量任务卸载到服务器时, 导致服务器出现瞬间过载而影响负载均衡.

图 6 对比了不同方案下的用户数和 MEC 服务器负载均衡标准差的关系. 一个用户产生一个任务. 该实验中, 纵向分析, 当用户数一定时, MEC 服务器的负载均衡标准差越小, 代表整体服务器之间的负载越均衡. 横向分析, 随着用户数的增加, 用户所产生的任务随之增加, 负载均衡标准差的变化程度代表该方案下负载均衡的稳定程度. 实验结果表明本文方案 LBMT 有着显著的优势. 通过负载预测, 有效降低了服务器的负载压力. 任务缓存队列的运用避免了潜在的服务器瞬时过载问题. 同时, 通过任务的自适应映射机制, 确保了服务器负载的动态平衡. LBA-EC 方案的负载均衡相比其他 3 个方案更好, 该方案的两个阶段都有负载均衡. 首先, 边缘节点之间的负载均衡将任务匹配到特定的边缘服务器, 通过将任务调度问题建模为动态加权二部图的最大完美匹配问题, 并找到当前二部图的最大完美匹配来实现. 其次, 容器之间的负载均衡将任务分配给特定的容器, 利用粒子群优化算法找到最优解来确定任务的分配. Random 方案在卸载的过程中没有考虑负载均衡, 因此负载均衡标准差最大且极为不稳定. MMF 方案和 TLIN 都考虑了负载均衡, 但随着用户数的增多, MMF 方案的负载均衡标准差在增大, 因为该方案是静态实现负载均衡, 在用户数越来越多时, 无法实时根据服务器情况作出最优卸载策略.

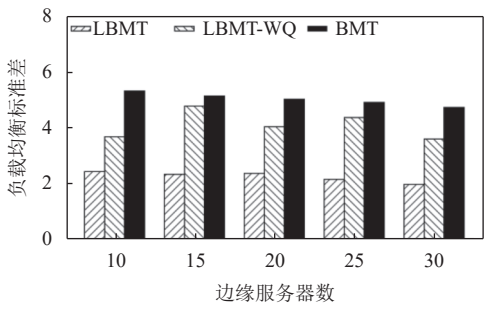

图 5 边缘服务器数与负载均衡标准差关系

图 7 描绘了不同方案的资源利用率. 图 7 中显示, 随着用户数量的增加, 资源利用率呈现出整体上升的趋势. Random 方案由于随机卸载任务而没有考虑资源的合理利用, 导致其资源利用率表现最低且最不稳定. LBA-EC 方案在任务卸载的两个阶段相对于 TLIN 方案更为复杂, 随着用户数的增加, 任务数量也在不断增多, 这种复杂性会随之上升, 最终导致资源利用率较

TLIN 方案更低. 本文方案 LBMT 通过预测任务负载预先了解任务对 MEC 的压力, 可以更好地规划和管理资源. 同时, 考虑了任务类型, 不同类型任务的资源需求不一样, 在任务卸载过程中实时更新服务器类型, 将任务卸载到相同类型的服务器. 因此, 提高了系统灵活性和资源利用率. 与 LBA-EC、TLIN、Random 方案对比, 资源利用率分别提高了 12.5%、14%、23.1%.

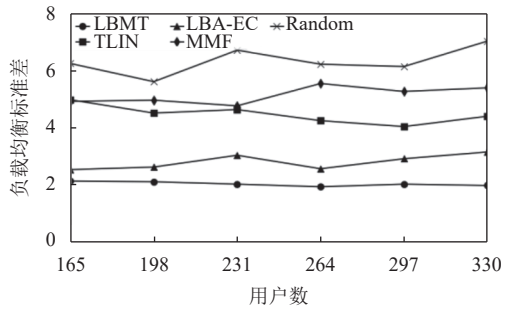

图 6 用户数与负载均衡标准差关系

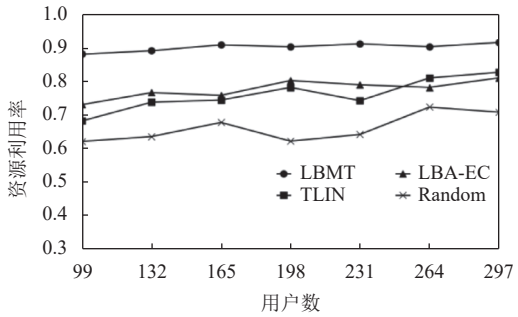

图 7 用户数与资源利用率关系

表 3 和表 4 分别对比了不同方案在服务器数量为 5 和 10 时, 每个服务器的资源利用率. 从表 3 和表 4 中可以看出本文方案 LBMT 相比其他方案能更充分利用所有服务器的资源, 使得服务器之间不会存在资源浪费, 以及各服务器之间的资源利用率差距更小, 在 5% 以内, 这种差距应至少在 10% 以内, 而其他两种方案均超过了 10%. 这是因为本文不仅考虑了动态更新服务器类型, 服务器处理同类型任务缓存队列, 还考虑了当同类型任务缓存队列为空时, 该类服务器能及时去处理其他类型的任务缓存队列. 服务器之间共享资源, 避免了资源浪费的同时, 也分担了其他类型服务器的压力, 不会出现服务器资源空闲或短缺的现象.

图 8 比较了不同方案下的任务处理率与边缘服务器数之间的关系. 任务处理率越大, 表示所有边缘服务器能一次性处理的任务越多, 执行效率越高. 从图中可以看出, 随着服务器数量不断增加, 任务处理率会逐渐上升并趋于稳定. 本文提出的方案 LBMT 始终具有比 MMF 和 TLIN 方案更高的任务处理率, 当服务器数量达到 30 时, 本方案的任务处理率比其他方案分别提高了 20.3%、24.7%、30.4%. 因为本文方案在任务映射算法中动态更新服务器类型, 为服务器合理分配各类队列任务, 服务器能处理更多的任务, 使得服务器的任务处理率最大化.

表 3 单个 MEC 资源利用率 (MEC 数量=5) (%)

MEC编号	LBMT	LBA-EC	Random
1	92	85	78
2	89	76	62
3	87	71	54
4	90	78	69
5	91	83	71

表 4 单个 MEC 资源利用率 (MEC 数量=10) (%)

MEC编号	LBMT	LBA-EC	Random
1	91	82	76
2	88	75	59
3	87	73	54
4	89	79	67
5	90	81	73
6	87	72	63
7	89	75	65
8	88	70	61
9	91	74	72
10	92	81	75

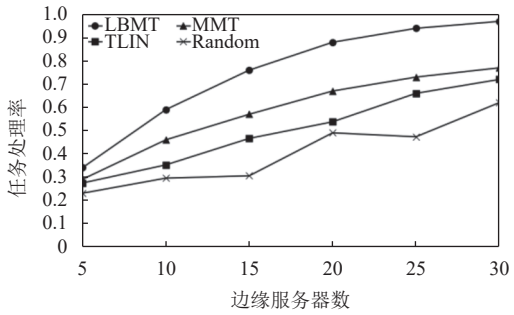

图 8 边缘服务器数与任务处理率关系

7 结束语

本文在多用户多 MEC 的边缘异构环境中, 针对多类型任务卸载与异构边缘服务器负载均衡问题, 提出一种面向多类型任务的负载预测以及均衡分配方案 LBMT. 该方案综合考虑了任务资源需求的差异性、任务类型的多样性、MEC 服务器的异构性以及 MEC 资

源的有限性等因素, 以资源利用率最高、任务处理率最大和负载均衡最优为目标得到最优负载均衡任务卸载策略. 仿真实验结果显示, 本文方案在用户数量逐渐增加的情况下取得了显著的负载均衡效果, 并成功提高了资源利用率. 此外, 在不同边缘服务器数量的情境下, 任务处理率也得到了显著提升. 在未来的研究中, 将考虑多类型任务之间的关联性以及任务卸载时延, 提出更佳的负载均衡任务卸载方案.

参考文献

1 Chen W, Zhu YQ, Liu JW, *et al*. Enhancing mobile edge computing with efficient load balancing using load estimation in ultra-dense network. Sensors, 2021, 21(9): 3135. [doi: 10.3390/s21093135]

2 Omori K, Shigemoto N, Kitagawa H, *et al*. Virtual reality as a learning tool for improving infection control procedures. American Journal of Infection Control, 2023, 51(2): 129–134. [doi: 10.1016/j.ajic.2022.05.023]

3 Xiong JH, Hsiang EL, He ZQ, *et al*. Augmented reality and virtual reality displays: Emerging technologies and future perspectives. Light: Science & Applications, 2021, 10(1): 216. [doi: 10.1038/s41377-021-00658-8]

4 Hu H, Song WW, Wang Q, *et al*. Energy efficiency and delay tradeoff in an MEC-enabled mobile IoT network. IEEE Internet of Things Journal, 2022, 9(17): 15942–15956. [doi: 10.1109/JIOT.2022.3153847]

5 Katal A, Dahiya S, Choudhury T. Energy efficiency in cloud computing data centers: A survey on software technologies. Cluster Computing, 2023, 26(3): 1845–1875. [doi: 10.1007/s10586-022-03713-0]

6 Cruz P, Achir N, Viana AC. On the edge of the deployment: A survey on multi-access edge computing. ACM Computing Surveys, 2023, 55(5): 1–34. [doi: 10.1145/3529758]

7 Cao K, Hu SY, Shi Y, *et al*. A survey on edge and edge-cloud computing assisted cyber-physical systems. IEEE Transactions on Industrial Informatics, 2021, 17(11): 7806–7819. [doi: 10.1109/TII.2021.3073066]

8 Luo QY, Hu SH, Li CL, *et al*. Resource scheduling in edge computing: A survey. IEEE Communications Surveys & Tutorials, 2021, 23(4): 2131–2165. [doi: 10.1109/COMST.2021.3106401]

9 Zhang WQ, Zhang GL, Mao SW. Joint parallel offloading and load balancing for cooperative-MEC systems with delay constraints. IEEE Transactions on Vehicular Technology, 2022, 71(4): 4249–4263. [doi: 10.1109/TVT.2022.3143425]

10 Bisht J, Vampugani VS. Load and cost-aware min-min workflow scheduling algorithm for heterogeneous resources in fog, cloud, and edge scenarios. International Journal of Cloud Applications and Computing, 2022, 12(1): 1–20. [doi: 10.4018/IJCAC.2022010105]

11 Chang S, Li CL, Deng CP, *et al*. Low-latency controller load balancing strategy and offloading decision generation algorithm based on lyapunov optimization in SDN mobile edge computing environment. Cluster Computing, 2024, 27(3): 2571–2591. [doi: 10.1007/s10586-023-04012-y]

12 Park J, Lim Y. Balancing loads among MEC servers by task redirection to enhance the resource efficiency of MEC systems. Applied Sciences, 2021, 11(16): 7589. [doi: 10.3390/app11167589]

13 Nguyen QM, Phan LA, Kim T. Load-balancing of Kubernetes-based edge computing infrastructure using resource adaptive proxy. Sensors, 2022, 22(8): 2869. [doi: 10.3390/s22082869]

14 Laboni NM, Safa SJ, Sharmin S, *et al*. A hyper heuristic algorithm for efficient resource allocation in 5G mobile edge clouds. IEEE Transactions on Mobile Computing, 2024, 23(1): 29–41. [doi: 10.1109/TMC.2022.3213410]

15 Hui M, Chen J, Zhou YC, *et al*. Server deployment and load balancing in stochastic mobile edge computing networks. IEEE Communications Letters, 2022, 26(5): 1194–1198. [doi: 10.1109/LCOMM.2022.3151467]

16 Su JW, Liu ZX, Xie YA, *et al*. UEE-delay balanced online resource optimization for cooperative MEC-enabled task offloading in dynamic vehicular networks. IEEE Internet of Things Journal, 2024, 11(7): 11496–11507. [doi: 10.1109/JIOT.2023.3330122]

17 Zhang R, Shu H, Navaei YD. Load balancing in edge computing using integer linear programming based genetic algorithm and multilevel control approach. Wireless Communications and Mobile Computing, 2022, 2022(1): 6125246. [doi: 10.1155/2022/6125246]

18 Bahrami B, Khayyambashi MR, Mirjalili S. Multi-objective placement of edge servers in mec environment using a hybrid algorithm based on NSGA-II and MOPSO. IEEE Internet of Things Journal, 2024, 11(18): 29819–29837. [doi: 10.1109/JIOT.2024.3409569]

19 Xin JJ, Li X, Zhang L, *et al*. Task offloading in MEC systems interconnected by metro optical networks: A computing load balancing solution. Optical Fiber Technology, 2023, 81: 103543. [doi: 10.1016/j.yofte.2023.103543]

20 Bonab MJA, Kandovan RS. Effective resource allocation and load balancing in hierarchical HetNets: Toward QoS-aware multi-access edge computing. The Computer Journal, 2023, 66(1): 229–244. [doi: 10.1093/comjnl/bxab157]

21 Lee H, Choi SI, Lee SH, *et al*. Distributed task offloading in mobile-edge computing with virtual machines. IEEE Internet of Things Journal, 2024, 11(13): 24083–24097. [doi: 10.1109/JIOT.2024.3388452]

22 Cui XY, Chen GF. Research on load balancing model of vehicle-to-everything communication transmission resources based on improved particle swarm optimization. Proceedings of the 6th IEEE International Conference on Knowledge Innovation and Invention (ICKII). Sapporo: IEEE, 2023. 104–108. [doi: 10.1109/ICKII58656.2023.10332707]

23 Chen X, Yao ZW, Chen ZY, *et al*. Load balancing for multiedge collaboration in wireless metropolitan area networks: A two-stage decision-making approach. IEEE Internet of Things Journal, 2023, 10(19): 17124–17136. [doi: 10.1109/JIOT.2023.3272010]

24 班玉琦, 段利国, 温昊宇, 等. 面向移动感知的计算卸载及资源分配策略研究. 计算机工程, 2023, 49(8): 163–173. [doi: 10.27352/d.cnki.gylgu.2023.000131]

25 彭世明, 林士飚, 贾硕, 等. 基于负载预测的多目标优化任务卸载策略. 计算机工程, 2024, 50(1): 206–215. [doi: 10.19678/j.issn.1000-3428.0066766]

26 Li JZ, Wang Q, Hu S, *et al*. Hybrid immune whale differential evolution optimization (HIWDEO) based computation offloading in MEC for IoT. Journal of Grid Computing, 2023, 21(4): 70–81. [doi: 10.1007/s10723-023-09705-7]

27 Putrada AG, Abdurohman M, Perdana D, *et al*. Edgesl: Edge-computing architecture on smart lighting control with distilled KNN for optimum processing time. IEEE Access, 2023, 11: 64697–64712. [doi: 10.1109/ACCESS.2023.3288425]

28 Shao SS, Liu SD, Li K, *et al*. LBA-EC: Load balancing algorithm based on weighted bipartite graph for edge computing. Chinese Journal of Electronics, 2023, 32(2): 313–324. [doi: 10.23919/cje.2021.00.289]

29 Li GS, Yao YH, Wu JH, *et al*. A new load balancing strategy by task allocation in edge computing based on intermediary nodes. EURASIP Journal on Wireless Communications and Networking, 2020, 2020(1): 3. [doi: 10.1186/s13638-019-1624-9]

(校对责编: 孙君艳)