计 算 机 研 究 与 发 展
Journal of Computer Research and Development
DOI: 10.7544/issn1000-1239.202110936
60(6): 1296−1307, 2023

异构边缘资源的任务卸载和协同调度

李小平[1] 周志星[2] 陈 龙[1] 朱 洁[3]
[1](东南大学计算机科学与工程学院 南京 211189)
[2](东南大学网络空间安全学院 南京 211189)
[3](南京邮电大学计算机学院 南京 210036)
(xpli@seu.edu.cn)

Task Offloading and Cooperative Scheduling for Heterogeneous Edge Resources

Li Xiaoping[1], Zhou Zhixing[2], Chen Long[1], and Zhu Jie[3]
[1](*School of Computer Science and Engineering*, *Southeast University*, *Nanjing* 211189)
[2](*School of Cyber Science and Engineering*, *Southeast University*, *Nanjing* 211189)
[3](*School of Computer Science*, *Nanjing University of Posts and Telecommunications*, *Nanjing* 210036)

Abstract Edge computing is commonly applied in emerging fields such as the Internet of things, the Internet of vehicles, and online games. Edge computing provides low-latency computing services for terminal devices by deploying computing resources at network edges. How to offload tasks to balance execution time and communication time and how to schedule tasks with different deadlines with the objective of minimizing the total tardiness are challenging problems. In this paper, a task offloading and scheduling framework is proposed for the heterogeneous edge computing. There are five components included in the framework: sequencing edge network nodes, sequencing offloaded task, task offloading strategies, task scheduling and the solution improvement. Multiple task offloading and task scheduling strategies are designed and embedded. ANOVA (multi-factor analysis of variance) is used to calibrate the algorithmic components and parameters over a large number of random instances. The algorithm with the best component combination is obtained. Based on the EdgeCloudSim simulation platform, several variants of the proposed algorithm are compared with the proposed algorithm from the perspectives of the number of edge nodes, the number of tasks, the distribution of tasks, and the interval of deadlines. Experimental results show that the proposed algorithm outperforms the other comparisons in all cases.

Key words edge computing; task offloading; task scheduling; deadline; total tardiness

摘 要 边缘计算广泛应用于物联网、车联网和在线游戏等新兴领域,通过网络边缘部署计算资源为终端设备提供低延迟计算服务.针对如何进行任务卸载以权衡任务执行时间与传输时间、如何调度多个不同截止期任务以最小化总延迟时间等挑战性问题,提出1种异构边缘协同的任务卸载和调度框架,包括边缘网络拓扑节点排序、边缘节点内任务排序、任务卸载策略、任务调度和结果调优等算法组件;设计多种任务卸载策略和任务调度策略;借助多因素方差分析(multi-factor analysis of variance, ANOVA)技术在大规模随机实例上校正算法算子和参数,得到统计意义上的最佳调度算法.基于EdgeCloudSim仿真平台,将所提出调度算法与其3个变种算法从边缘节点数量、任务数量、任务分布、截止期取值区间等角度进行性能比较.实验结果表明,所提出调度算法在各种情形下性能都优于对比算法.

收稿日期: 2021-09-15; **修回日期**: 2022-08-17
基金项目: 国家重点研发计划项目(2018YFB1402500); 国家自然科学基金项目(61872077); 国家自然科学基金重点项目(61832004)
This work was supported by the National Key Research and Development Program of China (2018YFB1402500), the National Natural Science Foundation of China (61872077) and the Key Program of the National Natural Science Foundation of China (61832004).

关键词 边缘计算；任务卸载；任务调度；截止期；延迟时间

中图法分类号 TP311

移动边缘计算广泛应用于物联网、车联网和在线游戏等领域，通过获取分布在网络边缘的计算能力和存储空间，可执行终端设备产生具有不同资源需求和延迟敏感的任务[1]. 如何减少延迟、提高用户体验是移动边缘计算关注的主要目标. 任务通过邻近接入点(access point, AP)传输至边缘服务器执行，以缓解终端设备有限的计算能力与高资源和高延迟敏感需求间的矛盾[2]；但单个边缘计算资源能力有限[3-4]，难以同时满足多个用户提交的任务，边缘服务器任务量过多容易陷入过载状态，导致待处理任务长时间等待. 通过多边缘协同调度[5]将过载边缘服务器上的任务转移至较为空闲的服务器上执行可减少任务的完成时间，但会产生传输代价，根据服务器状态如何卸载任务以权衡任务执行时间和传输时间难以决策；不同任务具有不同计算量和截止期，如何合理调度卸载分配到服务器资源上的任务以尽量满足截止期约束也是关键问题. 因此，任务卸载和调度是移动边缘计算的 2 个关键问题.

本文考虑异构边缘环境下带截止期约束任务卸载和协同调度问题，二者紧密关联. 终端设备通过邻近 AP 节点转发任务请求[6]，通过卸载决策将任务指派到具体边缘服务器上执行，但每个边缘节点计算资源有限，且边缘节点的计算能力和传输能力等具有差异性；各任务有不同的计算量、数据量和截止期需求；有限资源的约束导致卸载和调度通常不能完全满足截止期约束，即部分任务延迟，如何最小化总延迟时间是用户关注的优化目标. 这些问题是典型的 NP-hard 问题，其主要挑战为：1)异构边缘环境下各边缘节点的计算能力和传输能力不同，导致相同任务在不同边缘节点上执行时间和传输时间不同，如何进行任务卸载以实现任务执行时间与传输时间的权衡是一个难题；2)相同边缘节点的能力有限，如何调度多个不同截止期任务以最小化总延迟时间是一个挑战性问题.

考虑到这些问题的特点，本文首先基于现有边缘计算中通用架构，设计异构边缘协同的任务卸载和调度框架；分析所考虑问题的特征，建立相应的数学模型；提出一个异构边缘环境下带截止期的约束任务卸载和调度算法，包括边缘网络拓扑节点排序、边缘节点内任务排序、任务卸载决策、任务调度和结果调优 5 个部分；设计多种任务卸载策略和任务调度策略.

1 相关工作

不同边缘计算架构[7]下的任务卸载有较多研究，本文所考虑的问题涉及多服务器边缘计算、任务卸载、截止期约束调度等方面.

1)多服务器边缘计算方面. 文献 [8] 研究应用程序在本地执行或传输到边缘服务器以最小化移动设备能耗. 文献 [9] 研究单服务器边缘系统中的计算任务调度，设计双时间尺度随机优化调度策略，大时间尺度决定任务卸载，小时间尺度考虑具体任务传输过程，采用马尔可夫决策以优化延迟. 文献 [10] 提出将用户连接至最近边缘服务器，以提供最快的响应时间和最佳通信条件. 这些工作都集中于单服务器边缘场景，忽略边缘服务器在处理多任务时造成的拥塞，易于陷入过载状态. 文献 [11] 考虑多边缘计算系统，当单个服务器陷入过载状态时将计算负载转移到邻近空闲服务器，以最小化任务执行延迟和用户能耗. 文献 [12] 考虑用户通过异构网络将任务卸载至任意边缘服务器，综合考虑用户卸载决策、用户传输功率及服务器计算能力等. 文献 [13] 考虑边缘服务器的有限计算资源，提出分层边缘计算调度架构，利用邻近边缘节点空闲的计算能力，在邻近区域引入备份服务器解决边缘服务器的计算瓶颈问题，采用 Stackelberg 博弈论方法设计多层级卸载方案. 文献 [14] 研究终端设备可通过无线局域网获得多个边缘服务器的服务，用户可直接将所需卸载任务传输至云服务器，提出一种基于本地/卸载执行代价最小化的鲁棒计算卸载选择策略.

2)任务卸载方面. 文献 [15] 考虑在终端设备附近范围内有多个可用边缘服务器场景，提出一种通过代理服务器进行能耗和延迟感知的边缘服务器选择策略. 文献 [16] 考虑将过载服务器任务迁移到空闲服务器，以平衡多服务器负载，设计一种调度算法为超过负载阈值服务器选择可迁移的候选虚拟机，提出根据迁移效率选择最合适服务器的策略；这些研究只考虑任务卸载，而不考虑任务调度. 文献 [17] 考虑共享信道和有限计算资源，构建混合整数线性优化模型，提出基于拥塞感知的动态启发式算法，包括生成初始卸载决策、任务分配、冲突时的重调度策略. 文献 [18] 考虑多用户通过共享信道与边缘服

务器连接时的带宽分配影响，提出一种高效卸载决策算法以最小化平均响应时间. 文献 [19] 考虑大量用户对云计算资源的竞争，提出一种离线启发式算法以最小化用户平均完成时间. 文献 [20] 进一步研究移动边缘云中延迟敏感应用的计算卸载和资源分配，考虑计算资源和网络带宽二维资源分配，提出一种高效启发式算法. 文献 [8–20] 考虑不同场景下任务卸载和资源分配，当任务增加截止期等约束时，这些算法不再适用. 文献 [21] 考虑了任务迁移引起的大量数据传输问题，提出了基于延时传输机制的多目标工作流调度遗传算法，能够有效地优化移动设备的能耗和工作流的完工时间. 文献 [22] 考虑数据流调度优化问题，将该问题转换成为一个新的连续合作博弈，设计出快速收敛的基于博弈论的调度算法，保证资源效率和公平度.

3)截止期约束调度方面. 文献 [23] 同时考虑通信资源分配和作业到服务器的映射，提出基于合作博弈的调度方法，在截止期约束下最大化任务成功执行数量. 文献 [24] 以最小化总延迟时间为目标，采用匹配理论求解任务卸载，提出启发式算法. 文献 [25] 研究边缘计算中在线延迟感知任务分派和调度，最大化满足截止期约束的任务数量，提出贪心算法，将新到任务插入到当前队列以满足任务截止期. 文献 [26] 基于最高残留密度优先规则，提出云边环境下在线任务分派和调度算法，最小化加权响应时间. 文献 [27] 考虑任务的硬截止期约束，设计最小化总代价的启发式算法. 文献 [28] 考虑边缘计算场景中用户各自的截止期约束，采用博弈论方式分别优化，达到最终的纳什平衡. 文献 [29] 将并行数据处理应用上的请求封装成任务包，提出一个调度策略组 AutoBoT，包括实时决策定价、虚拟机获取和释放、任务放置、检查点设置和迁移，以保证在硬截止日期约束下任务的按期完成. 文献 [30] 将任务包应用在混合云下的异构资源分配问题模型化成二元线性规划问题，具有截止日期和资源约束，以所有任务包应用总代价为优化目标，利用 CPLEX(IBM 的建模优化引擎)计算解决方案. 文献 [31] 研究具有截止日期约束的云工作流调度问题，提出了一种新的自适应惯性权重计算方法更加精确地描述粒子状态，提高权重的自适应性，在此基础上提出了改进粒子群算法，该算法可以更好地平衡粒子全局与局部搜索，避免陷入局部最优. 文献 [32] 提出具有机器启动时间感知的虚拟机扩展策略，以缓解机器启动时间造成的实时任务延误问题，设计具有启动时间感知能力的调度算法来调度实时任务和资源.

综上所述，异构边缘计算环境下综合考虑任务截止期约束的多边缘协同的任务卸载和调度问题尚未有相关的研究.

2 系统框架和数学模型

针对本文考虑问题，基于现有边缘计算调度框架，提出如图 1 所示的改进调度框架，主要包含用户层、边缘层和云数据中心层. 用户层的终端设备产生资源需求高且延迟敏感任务，终端设备计算能力有限且电池寿命并不足以支持此类任务执行，通过邻近 AP 节点将任务传输至边缘层或云数据中心；边缘层包含部署在网络边缘的 AP 节点及位于 AP 节点上的边缘服务器，边缘服务器所部署的位置及所能处理的任务数量不同，当 1 个边缘节点任务数量过多导致长时间等待时，可将部分待处理任务传输至其他边缘节点执行；云数据中心层为边缘节点提供计算支持与各 AP 节点通过网络链路直接相连，拥有充足的计算资源，当边缘层中负载过大时将任务传输至云中执行以缓解负载压力，但由于云服务器距离边缘层较远需付出较高的传输代价.

为方便起见，做 4 点假设：1)终端设备仅通过最近 AP 将任务上传边缘系统，边缘系统可获得所有任务计算量、数据量和截止期约束等信息；2)任务具有软截止期约束，即允许任务超过截止期执行；3)边缘系统中不同边缘节点间的网络带宽和延迟事先给定，且调度时不发生变化；4)边缘服务器的异构性体现为计算能力、服务器中虚拟机个数等不同；5)任务执行不可中断，且不考虑终端设备的移动性.

边缘环境的网络拓扑可视为一个无向图$G=(V,E)$，其中$V=\{V_1,V_2,\cdots,V_M\}$表示边缘环境下 AP 节点集合，$E=\{e_{j,k}\,|\,Edge(V_j,V_k),V_j,V_k\in V\}$表示 AP 节点间的网络链路，边$e_{j,k}$连通代表$V_j,V_k$间可互相传输数据，设$\mathcal{B}_{j,k}$为链的带宽，$\mathcal{L}_{j,k}$为$e_{j,k}$传播延迟，$\mathcal{P}_{j,k}$为$V_j,V_k$间的最短路径(当需要从节点$V_j$传输任务到$V_k$时多条可达路径中的最短路径). 设每个 AP 节点有且仅有 1 台服务器，即共有M台异构边缘服务器$S=\{S_1,S_2,\cdots,S_M\}$，$Vm_j=\{Vm_j^1,Vm_j^2,\cdots,Vm_j^{|Vm_j|}\}$表示服务器$S_j$的虚拟机，$f_j^l$为服务器$S_j$上第$l$个虚拟机$Vm_j^l$的处理速度. 终端设备产生的独立任务需传输至服务器执行，设当前有N个分布在不同 AP 节点覆盖范围内的任务$T=\{T_1,T_2,\cdots,T_N\}$需执行，每个任务T_i先通过邻近的 AP 节点θ_i(集合V中的 1 个元素)上传至边缘

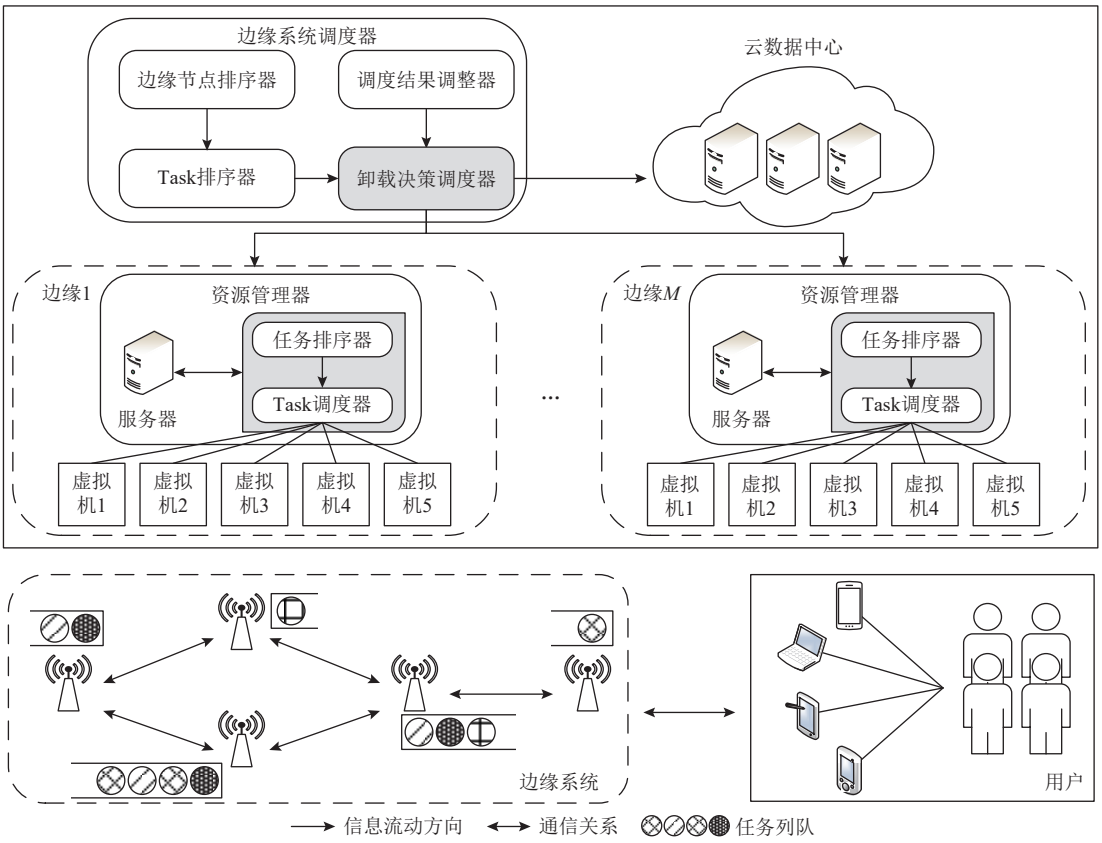


Fig. 1 Multi-edge cooperative scheduling framework

图 1 多边缘协同调度框架

系统，T_i可表示为四元组$(\theta_i, \gamma_i, c_i, d_i)$，其中$\gamma_i$为数据量大小、$d_i$为截止期、$c_i$为计算量大小. 任务提交后，通过最短路径将该任务传输至目的边缘服务器上执行. 云数据中心S_0通过网络链路与每个 AP 节点相连，计算资源充足，f_0为S_0虚拟机的计算速度，$\mathcal{B}_0$和$\mathcal{L}_0$分别为 AP 节点到S_0的网络带宽和延迟. 边缘系统通过卸载决策决定任务分配到哪台服务器上，决策变量$x_i \in \{0, 1\}(i = 1, 2, \cdots, N)$，$x_i = 0$表示任务$T_i$在云服务器执行，$x_i = 1$表示在边缘服务器执行；决策变量$y_{i,j} \in \{0, 1\}(j = 1, 2, \cdots, M)$，$y_{i,j} = 1$表示任务$T_i$指派到边缘服务器$S_j$上执行；决策变量$z_{i,j,l} \in \{0, 1\}(l = 1, 2, \cdots, |Vm_j|)$ $z_{i,j,l} = 1$，表示T_i匹配至边缘服务器S_j的虚拟机Vm_j^l. 本文考虑最小化边缘系统总延迟时间δ，即$\min\delta = \sum_{i=1}^{N}\max\{FT_i - d_i, 0\}$，$FT_i = ST_i + PT_i + DT_i$为$T_i$的完成时间，其中$ST_i = \max\{TT_i, EST_i\}$为$T_i$的开始时间，$PT_i$和$DT_i$为$T_i$的执行时间和执行结果回传时间，$TT_i$，$EST_i$为$T_i$的传输时间与最早可执行时间；

$$TT_i = (1 - x_i)\left(\frac{\gamma_i}{\mathcal{B}_0} + \mathcal{L}_0\right) + x_i\sum_{j=1}^{M}\left\{y_{i,j}\times\left(\frac{\gamma_i}{\min\limits_{e_{k,t}\in\mathcal{P}_{\theta_i,j}}\mathcal{B}_{k,t}} + \sum_{e_{k,t}\in\mathcal{P}_{\theta_i,j}}\mathcal{L}_{k,t}\right)\right\},$$

$$EST_i = x_i\sum_{j=1}^{M}\left\{y_{i,j}\times\sum_{l=1}^{|Vm_j|}\left(z_{i,j,l}\times\max_{T_{i'}\in\varphi^{j,l}}\{FT_{i'}\}\right)\right\},$$

执行时间和执行结果回传时间分别由$PT_i = (1 - x_i)\dfrac{c_i}{f_0} + x_i\sum_{j=1}^{M}\{y_{i,j}\times\sum_{l=1}^{|Vm_j|}\left(z_{i,j,l}\times\dfrac{c_i}{f_j^l}\right)\}$，$DT_i = (1 - x_i)\mathcal{L}_0 + x_i\sum_{j=1}^{M}\left(y_{i,j}\times\sum_{e_{k,t}\in\mathcal{P}_{\theta_i,j}}\mathcal{L}_{k,t}\right)$计算；决策变量满足$\sum_{j=1}^{M}y_{i,j} \leqslant 1$，$\sum_{j=1}^{M}\sum_{l=1}^{|Vm_j|}z_{i,j,l} \leqslant 1$.

3 异构边缘协同任务卸载和调度

针对所考虑问题，提出异构边缘环境下带截止期约束任务卸载和调度（task offloading and scheduling with deadline, TOSD）算法，主要包括边缘网络拓扑节点排序、被卸载任务排序、卸载目的选择策略、任务调度、结果调优等组件. 首先要确定哪些边缘网络节点的任务太多，需要卸载，即对边缘网络拓扑节点排序；其次确定各节点上需卸载哪些任务，即对卸载任务排序；再次，确定任务卸载到哪些服务器，即卸载目的选择策略；随后依据卸载至相同服务器的任务

优先级规则和任务的可选资源,产生任务调度序列;最后选用调整方法改进调度结果. TOSD 算法框架如算法 1 所示:

算法 1. TOSD 算法.

输入:任务集合 T,服务器集合 S,边缘计算系统信息 $G=(V,E)$,最大迭代次数 $\tau_{\max}$,收敛准则,调度结果调优算法参数 K,ω,c_1,c_2;

输出:总的任务延迟时间 δ_{best}.

① while $Q\neq\varnothing$ do

② 生成边缘网络拓扑的节点序列 Q;

③ $S_j\leftarrow Q$队首元素;

④ $\varphi^j\leftarrow$节点S_j上待卸载任务列表;

⑤ 卸载任务列表排序 $Sort(\varphi^j)$;

⑥ foreach $T_i\in\Phi^j$ do

⑦ 调用卸载目的选择策略为任务T_i指派具体服务器 S_{target};

⑧ end for

⑨ end while

⑩获得初始化卸载决策序列 P;

⑪ foreach $S_j\in S$ do

⑫ 调用任务调度算法调度节点S_j上的任务;

⑬ end for

⑭计算总延迟时间 δ_{best};

⑮调用调度结果调优算法获取最优调度结果 δ_{best};

⑯ return 总延迟时间 δ_{best}.

3.1 边缘网络拓扑节点排序

边缘节点承接任务过多时会陷入过载,需将过载节点任务卸载到其他节点,以减少任务的等待时间. 首先对边缘网络拓扑节点赋予不同优先级,优先级高的节点先任务卸载. 设计优先级规则时需考虑 3 点因素:1)节点计算量. 设 Φ^j为边缘节点 $S_j(j=1,2,\cdots,M)$ 的任务集合,$n_j=\left|\Phi^j\right|$为其任务数,即 $\Phi^j=\{T_r|r=1,2,\cdots,n_j\}$,$CL_j=\sum_{r=1}^{n_j}c_r$为$S_j$的计算负载. 若$CL_j>CL_{j'}$,则节点$S_j$的计算负载压力越大,陷入过载的可能性越大,优先级应更高. 2)节点度数. 与边缘节点直接相连的邻近节点数量考虑最小化任务卸载成本,度数越大其邻近节点数量越多,可有更多选择将任务卸载至邻近边缘节点;相反,度数越小其可选择余地越少. 因此节点度数越小优先级越高. 3)节点空闲程度. 异构边缘协同计算场景下的多个边缘服务器的负载动态变化,设节点S_j上任务T_r的预估完成时间为$\widehat{FT}_r$,定义 $FT_{\max}^j=\max\limits_{r=1,2,\cdots,n_j}\left\{\widehat{FT}_r\right\}$,所有节点最晚完成时间 $FT_{\max}=\max\limits_{j=1,2,\cdots,M}\left\{FT_{\max}^j\right\}$,节点$S_j$的空闲程度 $\Delta_j=\dfrac{FT_{\max}-FT_{\max}^j}{FT_{\max}}\in[0,1]$,$\Delta_j$值越大表明该节点$S_j$空闲程度越高,特别是 $\Delta_j=1$表明节点处于完全空闲.

基于这 3 点因素,提出 3 种边缘节点排序方法:

① 节点计算负载排序法(computation load based sequencing, CLBS). 按照计算量降序排列边缘节点.

②节点度数排序法(node degree based sequencing, NDBS). 按照度数升序排列边缘节点.

③节点空闲程度排序法(idle degree based sequencing, IDBS). 按照空闲程度升序排列边缘节点.

这 3 种方法采用简单规则对边缘网络边缘节点排序,平均时间复杂度为 $O(M\log M)$(其中 M 为边缘网络的节点数量).

3.2 被卸载任务排序

因待卸载任务可能较多,如何选择被卸载任务将对目标函数产生重要影响,选择策略受截止期、计算量和数据量等因素的影响. 尽早卸载截止期临近的任务,计算密集型任务(计算量大、数据量相对较小)需卸载至性能更好、等待时间更短的节点;数据密集型任务(计算量小而数据量大)更适合本地边缘节点. 由此,构造 2 种被卸载任务排序法:

1)计算密集程度排列法(degree of computation intensive sequencing, DCIS). 被卸载任务按照计算密集程度 $DCIS_i=\dfrac{c_i}{\gamma_i}\times100\%$降序排列.

2)预估松弛时间与数据量比值排列法(ratio of slack time to data-volume based sequencing, RSDS). 任务T_i的预估松弛时间ST_i为任务在当前节点上最快虚拟机 $\widehat{PT}_{i,j}^*=\dfrac{c_i}{\max\limits_{l=1,2,\cdots,\left|Vm_j\right|}\left\{f_j^l\right\}}$上执行时间与任务自身截止期$d_i$间的距离,即 $ST_i=d_i-\widehat{PT}_{i,j}^*$,比值 $STD_i=\dfrac{ST_i}{\gamma_i}$的值越大表明任务松弛时间越长或数据量越小进行卸载传输代价越小,可推迟其任务卸载. RSDS 按照 STD_i 升序排列任务.

2 种待卸载任务排序法平均时间复杂度为 $O\left(n_j\log(n_j)\right)$,$n_j$为边缘节点$S_j$上待卸载的任务数.

3.3 卸载目的选择策略

在确定哪些任务需要卸载后,选择卸载目的服务器. 本地节点服务器任务可传输至其他节点或云数据中心. 选择策略主要考虑处理时间和传输时间等因素. 异构性主要表现为边缘服务器虚拟机处理速度的差异,即同一任务在不同虚拟机上执行的时间不同.任务传输到网络拓扑不同位置边缘节点,传

输代价也不同. 任务卸载决策(task offloading decision, TOD)算法基本框架如算法 2 所示：

算法 2. TOD 算法.

输入：任务T_i, 边缘服务器集合S;

输出：任务T_i的卸载目标服务器S_{target}.

①目标服务器：$S_{\text{target}} \leftarrow S_{\theta_i}$;

②调用服务器选择策略;

③ for $j=0$ to M and $j \neq \theta_i$do

④ if S_j更加合适 then

⑤ $S_{\text{target}} \leftarrow S_j$;

⑥ end if

⑦ end for

⑧ if 云上执行更适合 then

/*云数据中心服务器在边缘节点计算能力匮乏时使用*/

⑨ $S_{\text{target}} \leftarrow S_0$;

⑩ end if

⑪ return S_{target}.

设边缘节点$S_j(j=1,2,\cdots,M)$上待卸载任务集合为Φ^j, 任务$T_i(i=1,2,\cdots,|\Phi^j|)$的初始位置为$\theta_i$(初始化为用户提交任务时的最邻近边缘节点), 服务器选择策略从$j(j\neq\theta_i)$中选择最适合T_i卸载的边缘服务器S_{target}. 根据不同选择准则提出不同服务器选择：

1)贪心选择法(greedy selection rule, GSR). 贪心选择T_i预估完成时间最早的服务器, 先计算T_i在初始边缘节点服务器上执行的最早完成时间$\widehat{EFT}_{i,\theta_i}=\widehat{PT}_{i,\theta_i}$(此时传输代价为 0); 依次估计$T_i$卸载至服务器$S_j$的最早完成时间$\widehat{EFT}_{i,j}=TT_{i,j}+\widehat{PT}_{i,j}+\overline{WT}_j$, 其中$TT_{i,j}(i\neq j)$为传输时间, 采用处理速度最快虚拟机$\max\limits_{l=1,2,\cdots,|Vm_j|}\{f_j^l\}$预估任务执行时间$\widehat{PT}_{i,j}$, 平均等待时间$\overline{WT}_j$为在节点$S_j$上尝试执行所有待卸载任务的平均等待时间$\dfrac{1}{|\Phi^j|}\times\sum\limits_{T_{i'}\in\Phi^j} wt_{i'}$.

2)最大收益选择法(maximum benefit selection, MBR). 任务T_i的初始预估执行时间为$\widehat{PT}_{i,\theta_i}$, 将其传输至其他边缘节点时, 执行时间减少量与传输时间增加量的差值定义为卸载收益$BT_{i,j}=\widehat{PT}_{i,\theta_i}+\overline{WT}_{\theta_i}-\left(\widehat{PT}_{i,j}+\overline{WT}_j\right)-(TT_{i,j}+DT_{i,j})$, 当$BT_{i,j}>0$时, 表明卸载到节点$S_j$能缩短任务完成时间, 选择其中收益最大的边缘服务器作为该任务的目的服务器.

3)最近最小负载法(minimum nearest and load rule, MNR). 定义$SR_{i,j}=\sum\limits_{e_{k,t}\in\mathcal{P}_{\theta_i,j}}\mathcal{L}_{k,t}+\dfrac{1}{\max\limits_{l=1,2,\cdots,|Vm_j|}\{f_j^l\}}\times\sum\limits_{T_{i'}\in\Phi^j}c_{i'}$度量不同边缘节点服务器的负载状态和距离当前任务所在节点的距离远近程度, 选择SR最小边缘节点.

卸载目的服务器的选择需遍历所有边缘节点, 这 3 个准则的时间复杂度均为$O(M)$.

3.4 任务调度

通常服务器有多个虚拟机, 接收到多个任务, 任务都有截止期约束, 如何合理调度多个任务到多个虚拟机是最小化总延迟时间、尽可能减少任务超期时间的关键. 涉及 2 个基本问题：任务按照什么顺序调度、每个任务安排到哪台虚拟机上执行, 即任务排序和资源分配. 任务调度(task scheduling, TS)过程如算法 3 所示：

算法 3: TS 算法.

输入：服务器S_j上待处理任务集合Φ^j, 虚拟机列表Vm_j;

输出：任务排序序列P, 任务虚拟机匹配序列Π, 任务完成时间FT.

① $P_0 \leftarrow \Phi^j$, $\Pi \leftarrow \varnothing$;

②根据任务排序规则对Φ^j中任务排序;

③ $P \leftarrow Sequence(P_0)$;

④ for $T_i \in \Phi^j$ do

⑤ 根据资源分配规则选择虚拟机Vm_j^*;

⑥ $FT_i \leftarrow \max\left\{TT_{i,j}, AVT_j^*\right\}+\dfrac{c_i}{f_j^*}$;

⑦ $\Pi \leftarrow \Pi \cup (T_i, Vm_j^*)$;

⑧ end for

⑨ return P, Π.

3.4.1 任务排序

考虑任务截止期约束和属性设计不同的任务排序规则：

1)先来先服务(first come first service, FCFS). 根据任务到达目的服务器的先后顺序排序.

2)截止期最逼近优先(nearest deadline first, NDF). 按照接近截止期程度由紧到松对任务排序.

3)最小松弛时间优先(minimum slack time first, MSTF). 按照任务松弛时间$ST_i=d_i-\widehat{PT}_i-AST_i$由小到大的顺序排序, $\widehat{PT}_i$为任务在此边缘服务器上的预估执行时间, AST_i表示任务到达该边缘节点的时间.

这 3 种排序方法的平均时间复杂度均为$O\left(|\Phi^j|\log\left(|\Phi^j|\right)\right)$.

3.4.2 资源分配

任务具有不同计算负载、不同截止期、不同截止期逼近程度. 不同边缘节点的计算能力不同、同一边缘服务器上多种虚拟机计算能力不同导致同一任

务在不同节点或不同虚拟机的处理时间也不同，某时刻不同虚拟机的可用时间不同，综合考虑这些因素设计资源分配规则：

1)最早完成时间优先（earliest finish time first, EFTF）. 尝试将任务$T_{i'}(i' \in \{1, 2, \cdots, n_j\})$放到当前边缘服务器中所有虚拟机上执行，计算其完成时间$FT_{i'}^{j,l}$，选择具有最早完成时间的虚拟机，即$Vm_j^* = \underset{l\in\{1,2,\cdots,|Vm_j|\}}{\arg\min}\left\{FT_{i'}^{j,l}\right\}$.

2)最早开始时间优先（earliest start time first, ESTF）. 选择具有最早空闲时间的虚拟机$Vm_j^* = \underset{l\in\{1,2,\cdots,|Vm_j|\}}{\arg\min}\{\max\{FT_{j,l}\}\}$执行任务$T_{i'}(i' \in \{1, 2, \cdots, n_j\})$.

3)负载均衡优先（load balance first, LBF）. 选择当前虚拟机列表中负载均衡度最小的虚拟机执行任务，负载均衡程度$LBR = \dfrac{1}{|Vm_j|} \times \sum_{l=1}^{|Vm_j|}\left(RT_j^l - \dfrac{1}{|Vm_j|} \times \sum_{l=1}^{|Vm_j|} RT_j^l\right)^2$，其中$RT_j^l$为服务器第$l(l \in \{1, 2, \cdots, |Vm_j|\})$个虚拟机的实际运行时间. 图 2 所示某节点有$VM_1, VM_2, VM_3$这3种不同处理速度的虚拟机，初始 LBR 值为 0.060. 任务调度到VM_1的执行时间为 0.75, LBR 值为 0.039(情况 1)；调度到VM_2的执行时间为 0.5, LBR 值为 0.110(情况 2)；调度到VM_3的执行时间为 0.25, LBR 值为 0.125(情况 3). 情况 1 使得负载均衡度最小，故选择虚拟机VM_1执行任务.

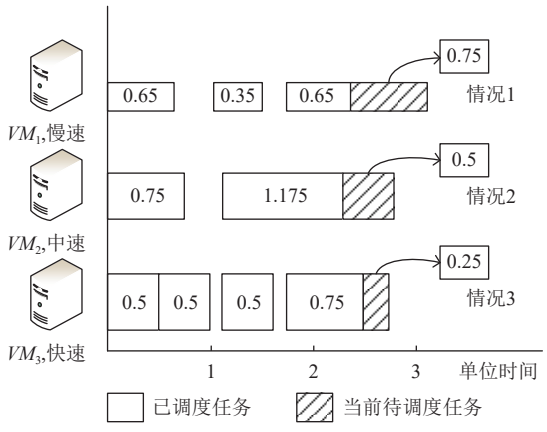


Fig. 2 Illustration of resource allocation

图 2 资源分配示意图

以上资源分配都需遍历节点中的所有$|Vm_j|$台虚拟机，故时间复杂度为$O(|Vm_j|)$.

3.5 调度结果调优

不同任务卸载和调度策略直接影响最终调度结果，为此提出调度结果调优算法以期提高解的质量. 迭代过程$\tau_{\max}$次：

1)除已经产生的初始解$X(0)$外，随机产生$K-1$个解；

2)以概率$p_m \in (0, 1)$随机选择 2 个解$\mathcal{K}_1, \mathcal{K}_2$进行合成操作，即随机挑选一个位置作为合成点，将$\mathcal{K}_1$的前半段和$\mathcal{K}_2$后半段合成为一个新解，同理剩余部分组成另外一个新解；

3)以概率$p_d \in (0, 1)$随机选择一个解$\mathcal{K}$进行分解操作，即随机选择一个位置作为分解点，分解为新解$\mathcal{K}_1, \mathcal{K}_2$，其中$\mathcal{K}_1$继承其前半段，$\mathcal{K}_2$继承其后半段，$\mathcal{K}_1, \mathcal{K}_2$的其他位置随机产生；

4)从所有解中选择最好解作为下次迭代的初始解$X(0)$.

4 实验结果与分析

为评价和分析所提出算法的性能，采取相对百分比偏差（relative percentage deviation, RPD）作为评估标准，借助多因素方差分析（analysis of variance, ANOVA)方法分析算法参数和算子校正结果、比较算法. 对独立任务集合T，设当前算法获得解π_ω的总延迟时间为$\delta(\pi_\omega)$，所有算法运行最优解π_ω^*的总延迟时间为$\delta(\pi_\omega^*)$，定义$RPD = \dfrac{\delta(\pi_\omega) - \delta(\pi_\omega^*)}{\delta(\pi_\omega)} \times 100\%$.

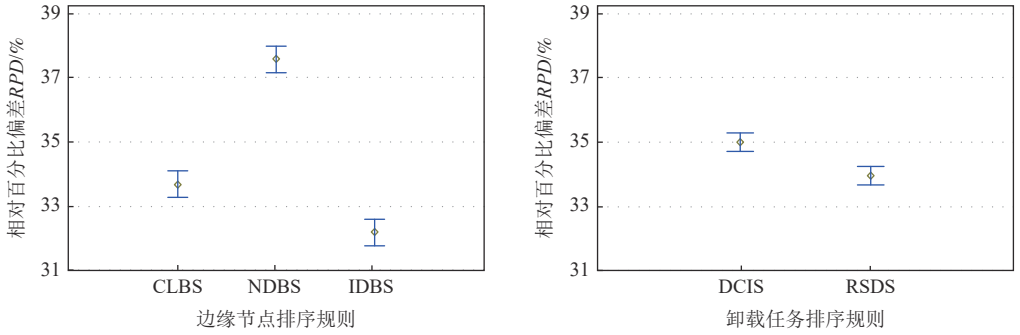
采用 EdgeCloudSim[33] 实验平台作为边缘计算仿真工具，模拟边缘环境，所有算法均采用 Java 编写，在同一配置（Intel®Core(TM) i5-8265CPU @1.60 GHz, 内存 8 GB, Window10 操作系统）的机器上运行. 所有实验参数设置借鉴文献 [23]，采用 BRITE[34] 拓扑生成器生成各边缘的位置分布，各边缘节点任务服从分布，任务的数量规模从 1 000 到 2 000 每次增长 200，任务数据量大小服从均匀分布$U \sim (1\,000, 2\,000)$(单位为 KB)，任务计算量大小服从均匀分布$U \sim (0.4, 2.4)$($\times 10^9$CPU 周期数)；边缘服务器上虚拟机的处理速度从{0.5, 0.6, ⋯, 1.0}(单位为 GHz)中取值，各边缘 AP 节点间的网络带宽服从$U \sim (30, 50)$(单位为 Mbps)，链路间延迟$\mathcal{L}_{j,k}(j, k \in \{1, 2, \cdots, M\})$服从均匀分布$U \sim (0.01, 0.05)$(单位为 s)，云数据中心与边缘 AP 节点间的带宽为 100 Mbps, 链路网络延迟为 0.5 s.

4.1 算法算子和参数校正

为得到统计意义上的校正结果，每个任务数随机生成 10 组实例；任务 Zipf 分布不均匀程度因子$\lambda \in \{0, 0.1, 0.2, 0.3, 0.4\}$；边缘节点数 M 随机从{15, 20, 30, 40}中取值，分别为M_1, M_2, M_3, M_4；任务截止期 D 从 [0.5, 2], [0.5, 3], [0.5, 4], [0.5, 5] 中取值，分别称为D_1, D_2, D_3, D_4；故共有$10\times4\times5\times4\times10 = 8000$组随机

实例. 另外, 考虑 3 种边缘节点排序方法(CLBS, NDBS, IDBS)、2 种被卸载任务排序法(DCIS, RSDS)、3 种卸载目的选择策略(GSR, MBR, MNR)、3 种任务排序方法(FCFS, NDF, MSTF)和 3 种资源分配规则(EFTF, ESTF, LBF), $\tau_{\max}=\mu\times N$, $K=v\times N$, 其中 $\mu\in\{0.1, 0.2, 0.3, 0.4, 0.5\}$, $v\in\{0.05, 0.1, 0.15, 0.2, 0.25\}$. 共 $3\times2\times3\times3\times3\times5\times5=4050$种组合; 总共进行 $8000\times4050=32400000$次实验. 经多次实验发现 $p_m=0.4$, $p_d=0.6$效果最好.

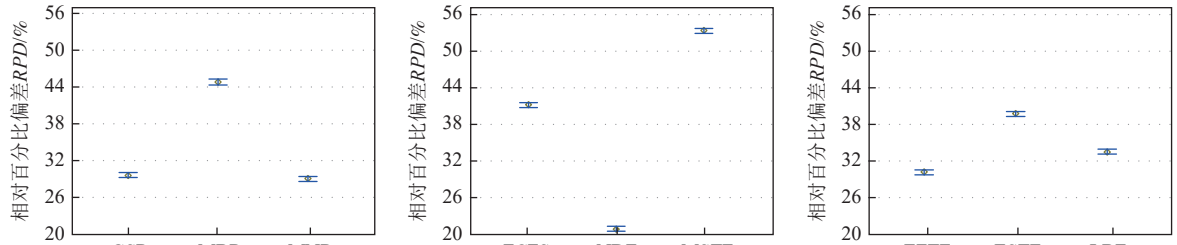
采用 ANOVA 分析实验结果, 3 个主要假设(正态性、齐次性、残差独立性)都做了验证, 实验结果表明 3 个假设都可接受, p 值都小于 0.05, 表明所有因素在 95% 置信水平上都有显著性差别. 相应结果如图 3~5 所示:



(a) 边缘网络拓扑节点排序规则对比结果 (b) 卸载任务排序规则对比结果

Fig. 3 Comparison results of edge network node sequencing rules and offloaded task sequencing rules with 95% Tukey HSD confidence intervals

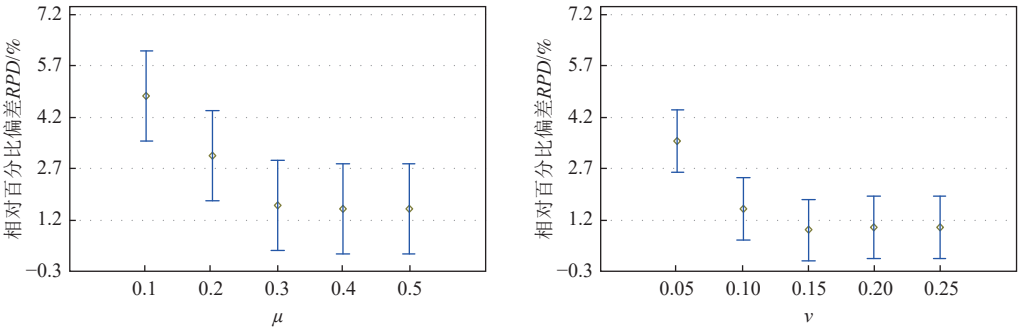
图 3 95% Tukey HSD 置信区间下边缘网络节点排序规则和卸载任务排序规则对比结果



(a) 卸载目的选择对比结果 (b) 任务排序规则对比结果 (c) 资源分配规则对比结果

Fig. 4 Comparison results of offloading destination selecting rules, task sequencing strategies rules and resource allocation rules with 95% Tukey HSD confidence intervals

图 4 95% Tukey HSD 置信区间下卸载目的选择规则、任务排序规则和资源分配规则对比结果



(a) 迭代次数因子μ对比结果 (b) 规模因子v对比结果

Fig. 5 Mean plots of *RPD* with difference μ, v under 95% Tukey HSD confidence intervals

图 5 95%Tukey HSD 置信区间下不同μ, v的 *RPD* 均值图

由图 3(a)可以看出, CLBS 和 IDBS 的 *RPD* 值小于 NDBS 的 *RPD* 值. CLBS 优先处理负载压力大的节点任务以缓解过载压力; NDBS 考虑边缘网络结构, 度数小的节点传输任务至其他节点上需付出更大传输代价, 但未考虑边缘系统中不同节点的任务分布; IDBS 考虑节点的任务负载和其计算能力, 更适合于减少总的任务延迟. 本文选择 IDBS 排序边缘网络拓扑节点.

由图 3(b)可以看出, DCIS 和 RSDS 间 *RPD* 值的差距较小, 但 RSDS 显著优于 DCIS. 本文将选择 RSDS.

由图 4(a)可以看出, 选择策略 GSR 和 MNR 的 *RPD* 值远小于 MBR 的. GSR 考虑多任务间的竞争及节点中负载的动态变化; MBR 卸载任务至收益最高的节点, 但会导致多个任务卸载至高收益的节点上致其陷入过载状态; MNR 将任务优先卸载至最近最小负载的节点上, 综合考虑节点负载和多任务间的竞争. GSR 和 MNR 更加适合于减少总的任务延迟时间. MNR 将用于选择卸载目的服务器.

由图 4(b)可以看出, NDF 显著优于 FCFS 和 MSTF, 其 *RPD* 值远小于 FCFS 和 MSTF 的; FCFS 的 *RPD* 值小于 MSTF 的, 表明 MSTF 效果最差. FCFS 优先处理先到来任务, 使计算资源可能空闲, 但未考虑任务自身属性, 尤其是其截止期约束, 影响总延迟时间; NDF 基于截止期逼近程度进行任务排序, 充分考虑任务达到边缘节点时间和截止期约束, 可尽量减小总延迟时间; MSTF 将松弛时间大的任务推迟执行, 在一定程度上减少任务延迟时间, 但当任务数多时将造成多个任务都超期完成, 使总延迟时间更大. 因此, 选择 NDF 进行任务排序较好.

由图 4(c)可以看出, EFTF 相比 ESTF 和 LBF 其 *RPD* 值更小. EFTF 为任务分配具有最早完成时间的虚拟机, 使任务具有最短执行时间; ESTF 规则选择最早可用的虚拟机, 尽早地开始执行任务, 避免任务长时间等待, 造成任务超期完成; LBF 均衡节点中各虚拟机负载, 避免高性能虚拟机过度占用, 导致后续截止期逼近程度大的任务无法获取高性能计算资源而超出截止期. 本文选择 EFTF 规则进行资源分配.

由图 5 可以看出, 随着 μ, ν 取值的增加, *RPD* 值呈下降趋势, 但在统计意义上 *RPD* 没有显著性差别, 因此本文 $\mu = 0.3$, $\nu = 0.15$.

4.2 算法比较

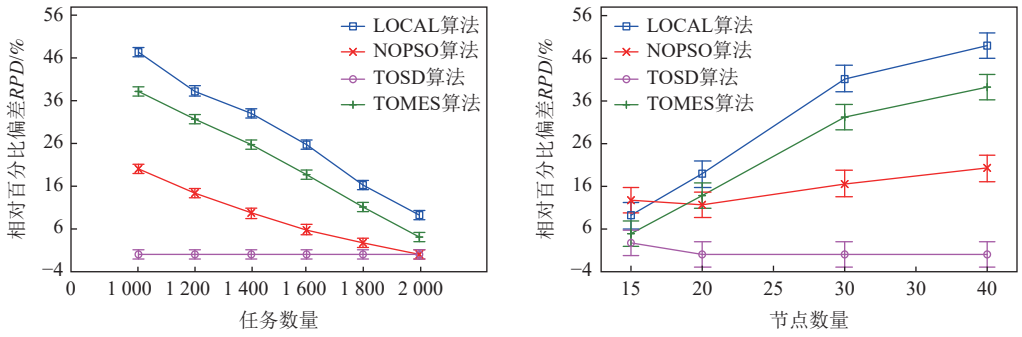
由于所考虑的问题是新问题, 尚未有现成的方法可以参考, 因此我们将 3 个同类相关问题的常见策略与所提出的 TOSD 算法进行比较:

1)单边缘本地独立卸载法 LOCAL. 每个边缘节点覆盖范围内的任务只能传输至当前边缘节点或云数据中心执行, 不能卸载至其他边缘节点上执行, 这种策略属于边缘.

2)最早完成时间优先的多边缘协同卸载方法 TOMES. 每个边缘节点都可以接收其他边缘节点附近的任务, 按照最早完成时间优先规则进行任务指派.

3)最早开始时间优先的多边缘协同卸载方法 NOPSO. 每个边缘节点都可以接收其他边缘节点附近的任务, 按照最早开始时间优先规则进行任务指派, NOPSO 与 TOSD 类似, 但无结果调优步骤, 仍然采用该方法产生随机实例, 比较这 4 个算法与任务数、边缘节点数、任务分布和截止期等关键参数在 95%Tukey HSD 置信区间统计意义下的相互作用, *RPD* 结果如图 6~7 所示.

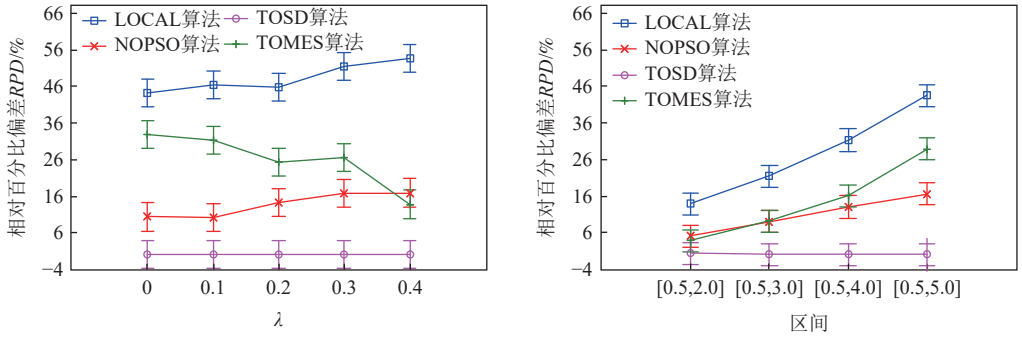
图 6(a)表明随着任务数量的增加, 各算法的 *RPD* 值下降. 任意任务数量规模下, TOSD 的性能都最佳; LOCAL 性能最差, 其主要原因在于 LOCAL 未



(a) 任务数量比对结果 (b) 边缘节点数量对比结果

Fig. 6 Interactions between instance parameters (task number and node number) and the compared algorithms with 95.0% Tukey HSD confidence intervals

图 6 95%Tukey HSD 置信区间下实例参数(任务数量和节点数量)和比较算法的相互作用



(a) 任务分布参数λ对比结果　　(b) 截止期取值区间对比结果

Fig. 7 Interactions between instance parameters （task distribution and deadline） and the compared algorithms with 95.0% Tukey HSD confidence intervals

图 7 95%Tukey HSD 置信区间下实例参数（任务分布和截止日期）和比较算法的相互作用

进行多边缘协同调度，没有充分利用异构边缘系统中其他空闲边缘节点的计算能力；TOMES 的 *RPD* 均值相较 TOSD 的有一定差距，其原因在于 TOMES 仅考虑任务卸载，权衡任务传输时间和执行时间，但未考虑多用户间对资源的竞争；NOPSO 相较于 TOSD 的 *RPD* 均值较高，表明进行调度结果调优减少任务超出截止期的比例. 随任务数的增大，*RPD* 均值间的差距越来越小，其原因在于任务数量增多时，每个节点上的负载增加，计算资源逐渐到达瓶颈，每个任务在执行过程中超出截止期的可能性增大，导致总延迟时间差值变小.

由图 6(b)可以看出，在所有边缘节点数，TOSD 的性能都最佳. 随着边缘节点数增加，TOSD 的 *RPD* 均值呈下降趋势、NOPSO 的 *RPD* 均值先降后升、LOCAL 和 TOMES的 *RPD* 均值增加；其他 3 种算法与 TOSD 算法的 *RPD* 均值差距变大，其原因在于当边缘节点数增加时，计算资源更加丰富，任务卸载和任务调度对结果影响更大.

图 7(a)表明，随着参数λ的变化，TOSD 算法性能最稳定；NOPSO 和 LOCAL 的 *RPD* 均值随λ值增加而增加；尽管 TOMES 的 *RPD* 均值随λ值增加而减少，但其初始值远大于 TOSD 的初始值. 采取 ZipF 分布模拟边缘系统中任务分布，λ值越大表明任务分布越不均匀，$\lambda=0$表示任务均匀分布在边缘系统中各节点覆盖范围内. 随参数λ值的增大，任务分布不均匀程度变大，LOCAL 算法 *RPD* 均值逐渐增大的原因在于部分边缘节点过载，导致在其上执行的任务延迟时间较长，未采用卸载策略充分利用其他较为空闲的边缘节点. TOSD 比 NOPSO 算法的 *RPD* 均值小，其原因在于其采取调度结果调优，性能得以提升.

由图 7(b)可以看出，TOSD 算法对任务截止期区间不敏感且任何情形平均 *RPD* 值都最小；其他 3 个算法随着截止期区间的增加 *RPD* 值也增加(性能下降)，其原因在于随着任务延迟敏感程度的降低，任务传输至其他边缘节点上执行所需付出的代价较大，造成任务延迟时间增长. 当任务截止期在 [0.5, 2] 中取值时，4 种算法的 *RPD* 均值基本差不多.

经过实验比较可以看出，单边缘本地独立卸载法 LOCAL 在所有测试实例参数变化下表现最差，其主要原因是边缘节点只能处理其附近产生的任务，虽然就近策略可以产生较少的通信延迟时间，但是如果边缘节点附近产生任务过多，容易产生任务积压，从而造成总延迟时间变长. TOMES 和 NOPSO 能够很好地在空闲边缘节点与繁忙边缘节点之间进行协同任务调度，因此性能较 LOCAL 更好. 但是 TOMES 和 NOPSO 均未像 TOSD那样考虑多用户间对资源的竞争，因此性能也有所降低. TOSD 的优势主要在于：1)度量了边缘节点的空闲程度，保证了负载均衡；2)优先卸载松弛度低的任务，降低了任务超期执行的概率；3)在为任务决定指派节点时，综合考虑了节点距离和负载情况，以尽量降低通信延迟和等待时间；4)引入了具有一定随机性的调优策略，避免上述贪婪选择陷入局部最优，有一定概率跳出局部最优，逼近全局最优.

5 总　　结

本文考虑异构边缘环境下带截止期约束的任务卸载和调度问题，提出异构边缘协同的任务卸载和调度算法框架 TOSD，解决确定哪些边缘网络节点的任务太多需要卸载、确定各节点上需卸载哪些任务、确定任务卸载到哪些服务器、任务按照什么顺序调

度、如何为每个任务分配资源等核心问题. 采用多因素方差分析技术 ANOVA 对算法算子和参数进行校正，选取最佳的算子和参数组合作为解决所考虑问题的算法. 将 TOSD 算法与其变种算法对比，通过实验结果可以看出，所提出算法在不同参数设置下都明显优于其他对比算法，验证了所提出算法算子对所考虑问题都有效.

本文从异构边缘环境资源层面考虑边缘服务器和云数据中心服务器，还有很多值得研究的问题，如：1)终端设备往往拥有一定的计算能力，可执行一定数量的任务，如何进行任务调度值得研究；2)终端设备具有移动性，任务提交位置与结果返回位置可能不同，如何考虑终端设备移动的任务调度值得研究.

作者贡献声明：李小平负责算法设计、论文观点的提炼和论文撰写工作；周志星负责算法开发与实验环境搭建；陈龙负责实验设计与实验数据分析；朱洁负责研究实验调研、论文修改.

参 考 文 献

[1] Mao Yuyi, You Changsheng, Zhang Jun, et al. A survey on mobile edge computing: The communication perspective[J]. IEEE Communications Surveys & Tutorials, 2017, 19(4): 2322−2358

[2] Wu Huaming, Sun Yi, Wolter K. Energy-efficient decision making for mobile cloud offloading[J]. IEEE Transactions on Cloud Computing, 2018, 8(2): 570−584

[3] Huang Peiqiu, Wang Yong, Wang Kezhi, et al. A bilevel optimization approach for joint offloading decision and resource allocation in cooperative mobile edge computing[J]. IEEE Transactions on cybernetics, 2019, 50(10): 4228−4241

[4] Liu Wei, Huang Yucheng, Du Wei, et al. Resource-constrained serial task offload strategy in mobile edge computing[J]. Journal of Software, 2020, 31(6): 1889−1908 (in Chinese)
(刘伟, 黄宇成, 杜薇, 等. 移动边缘计算中资源受限的串行任务卸载策略[J]. 软件学报, 2020, 31(6): 1889−1908)

[5] Yang Lei, Yao Haipeng, Wang Jingjing, et al. Multi-UAV-enabled mobile edge computing for time-constrained IoT applications[J]. IEEE Internet of Things Journal, 2020, 7(8): 6898−6908

[6] Lin Hai, Zeadally S, Chen Zhihong, et al. A survey on computation offloading modeling for edge computing[J]. Journal of Network and Computer Applications, 2020, 169: 102781

[7] Taleb T, Samdanis K, Mada B, et al. On multi-access edge computing: A survey of the emerging 5G network edge cloud architecture and orchestration[J]. IEEE Communications Surveys & Tutorials, 2017, 19(3): 1657−1681

[8] Zhang Weiwen, Wen Yonggang, Guan K, et al. Energy-optimal mobile cloud computing under stochastic wireless channel[J]. IEEE Transactions on Wireless Communications, 2013, 12(9): 4569−4581

[9] Liu Juan, Mao Yuyi, Zhang Jun, et al. Delay-optimal computation task scheduling for mobile-edge computing systems [C] //Proc of IEEE Int Symp on Information Theory. Piscataway, NJ: IEEE, 2016: 1451−1455

[10] Li Yujin, Wang Wenye. The unheralded power of cloudlet computing in the vicinity of mobile devices[C] //Proc of IEEE Global Communications Conf. Piscataway, NJ: IEEE, 2013: 4994−4999

[11] Tran T X, Pompili D. Joint task offloading and resource allocation for multi-server mobile-edge computing networks[J]. IEEE Transactions on Vehicular Technology, 2018, 68(1): 856−868

[12] Chen Xu. Decentralized computation offloading game for mobile cloud computing[J]. IEEE Transactions on Parallel and Distributed Systems, 2014, 26(4): 974−983

[13] Dai Yueyue, Xu Du, Maharjan S, et al. Joint load balancing and offloading in vehicular edge computing and networks[J]. IEEE Internet of Things Journal, 2018, 6(3): 4377−4387

[14] Chen Menggang, Guo Songtao, Liu Kai, et al. Robust computation offloading and resource scheduling in cloudlet-based mobile cloud computing[J]. IEEE Transactions on Mobile Computing, 2021, 20(5): 2025−2040

[15] Mukherjee A, De D, Roy D G. A power and latency aware cloudlet selection strategy for multi-cloudlet environment[J]. IEEE Transactions on Cloud Computing, 2016, 7(1): 141−154

[16] Ramasubbareddy S, Ramasamy S, Sahoo K S, et al. CAVMS: Application-aware cloudlet adaption and VM selection framework for multicloudlet environment[J]. IEEE Systems Journal, 2020, 15(4): 5098−5106

[17] Guo Kai, Yang Mingcong, Zhang Yongbing, et al. Efficient resource assignment in mobile edge computing: A dynamic congestion-aware offloading approach[J]. Journal of Network and Computer Applications, 2019, 134(1): 40−51

[18] Guo Kai, Yang Mingcong, Zhang Yongbing, et al. Joint computation offloading and bandwidth assignment in cloud-assisted edge computing [J/OL]. IEEE Transactions on Cloud Computing, 2019 [2022-01-25]. http://dx. doi.org/10.1109/TCC.2019.2950395

[19] Yang Lei, Cao Jiannong, Cheng Hui, et al. Multi-user computation partitioning for latency sensitive mobile cloud applications[J]. IEEE Transactions on Computers, 2014, 64(8): 2253−2266

[20] Yang Lei, Liu Bo, Cao Jiannong, et al. Joint computation partitioning and resource allocation for latency sensitive applications in mobile edge clouds[J]. IEEE Transactions on Services Computing, 2019, 14(5): 1439−1452

[21] Zhou Yemao, Li Zhongjin, Ge Jidong, et al. Multi-objective workflow scheduling based on delay transmission in mobile cloud computing[J]. Journal of Software, 2018, 29(11): 3306−3325 (in Chinese)
(周业茂, 李忠金, 葛季栋, 等. 移动云计算中基于延时传输的多目标工作流调度[J]. 软件学报, 2018, 29(11): 3306−3325)

[22] Shen Rao, Qin Xiaolin, Bao Zhifeng. Effective multi-objective scheduling strategy of dataflow in cloud[J]. Journal of Software, 2017, 28(3): 579−597 (in Chinese)
(沈尧, 秦小麟, 鲍芝峰. 一种云环境中数据流的高效多目标调度方

法[J]. 软件学报, 2017, 28(3): 579−597)

[23] Liu Chubo, Li Kenli, Liang Jie, et al. COOPER-SCHED: A cooperative scheduling framework for mobile edge computing with expected deadline guarantee [J/OL]. IEEE Transactions on Parallel and Distributed Systems, 2019 [2022-01-25]. http://dx.doi.org/10.1109/TPDS.2019.2921761

[24] Xiao Surong, Liu Chubo, Li Kenli, et al. System delay optimization for mobile edge computing[J]. Future Generation Computer Systems, 2020, 109: 17−28

[25] Meng Jiaying, Tan Haisheng, Xu Chao, et al. Dedas: Online task dispatching and scheduling with bandwidth constraint in edge computing[C] //Proc of the 18th IEEE Conf on Computer Communications. Piscataway, NJ: IEEE, 2019: 2287−2295

[26] Tan Haisheng, Han Zhenhua, Li Xiangyang, et al. Online job dispatching and scheduling in edge-clouds[C/OL] //Proc of the 16th IEEE Conf on Computer Communications. Piscataway, NJ: IEEE, 2017 [2022-02-17]. http://dx. doi.org/10.1109/INFOCOM.2017.8057116

[27] Chen M H, Dong Min, Liang Ben. Resource sharing of a computing access point for multi-user mobile cloud offloading with delay constraints[J]. IEEE Transactions on Mobile Computing, 2018, 17(12): 2868−2881

[28] Meskar E, Todd T D, Zhao Dongmei, et al. Energy aware offloading for competing users on a shared communication channel[J]. IEEE Transactions on Mobile Computing, 2016, 16(1): 87−96

[29] Varshney P, Simmhan Y. AutoBoT: Resilient and cost-effective scheduling of a bag of tasks on spot VMs[J]. IEEE Transactions on Parallel and Distributed Systems, 2019, 30(7): 1512−1527

[30] Abdi S, Pourkarimi L, Ahmadi M, et al. Cost minimization for deadline-constrained bag-of-tasks applications in federated hybrid clouds[J]. Future Generation Computer Systems, 2017, 71: 113−128

[31] Li Xuejun, Xu Jia, Zhu Erzhou, et al. A novel computation method for adaptive inertia weight of task scheduling algorithm[J]. Journal of Computer Research and Development, 2016, 53(9): 1990−1999 (in Chinese)
(李学俊, 徐佳, 朱二周, 等. 任务调度算法中新的自适应惯性权重计算方法[J]. 计算机研究与发展, 2016, 53(9): 1990−1999)

[32] Chen Huangke, Zhu Jianghan, Zhu Xiaomin, et al. Resource-delay-aware scheduling for real-time tasks in clouds[J]. Journal of Computer Research and Development, 2017, 54(2): 446−456 (in Chinese)
(陈黄科, 祝江汉, 朱晓敏, 等. 云计算中资源延迟感知的实时任务调度方法[J]. 计算机研究与发展, 2017, 54(2): 446−456)

[33] Sonmez C, Ozgovde A, Ersoy C. EdgeCloudSim: An environment for performance evaluation of edge computing systems[C] //Proc of the 2nd Int Conf on Fog and Mobile Edge Computing. Piscataway, NJ: IEEE, 2017: 39−44

[34] Medina A, Lakhina A, Matta I, et al. BRITE: An approach to universal topology generation[C] //Proc of the 9th Int Symp on Modeling, Analysis and Simulation of Computer and Telecommunication Systems. Piscataway, NJ: IEEE, 2001: 346−353

Li Xiaoping, born in 1970. PhD, distinguished professor. His main research interests include cloud computing, cloud manufacturing, and scheduling in service computing.

李小平, 1970 年生. 博士, 特聘教授. 主要研究方向为云计算、云制造和服务计算调度.

Zhou Zhixing, born in 1997. Master candidate. His main research interest includes scheduling in edge computing.

周志星, 1997 年生. 硕士研究生. 主要研究方向为边缘计算调度.

Chen Long, born in 1988. PhD. His main research interests include cloud computing, cloud manufacturing, and scheduling in service computing.

陈　龙, 1988 年生. 博士. 主要研究方向为云计算、云制造和服务计算调度.

Zhu Jie, born in 1983. PhD, associate professor. Her main research interests include cloud computing and scheduling in edge computing.

朱　洁, 1983 年生. 博士, 副教授. 主要研究方向为云计算、边缘计算调度.