计算机科学
COMPUTER SCIENCE

基于深度确定性策略梯度的服务器可靠性任务卸载策略

李梦菲，毛莺池，屠子健，王瑄，徐淑芳

引用本文

李梦菲，毛莺池，屠子健，王瑄，徐淑芳. 基于深度确定性策略梯度的服务器可靠性任务卸载策略[J]. 计算机科学, 2022, 49(7): 271-279.

LI Meng-fei, MAO Ying-chi, TU Zi-jian, WANG Xuan, XU Shu-fang. Server-reliability Task Offloading Strategy Based on Deep Deterministic Policy Gradient [J]. Computer Science, 2022, 49(7): 271-279.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.210600040

基于深度确定性策略梯度的服务器可靠性任务卸载策略

李梦菲[1] 毛莺池[1,2] 屠子健[1] 王 瑄[1] 徐淑芳[1,2]
1 河海大学计算机与信息学院 南京 210098
2 水利部水利大数据技术重点实验室 南京 210098
(2028004563@qq.com)

摘 要 随着智能移动设备的普及,新一代移动应用如人脸识别、虚拟现实等逐渐兴起,但移动设备因计算能力和电池容量有限,无法支持这类计算需求高且延迟敏感的应用。因此,移动边缘计算被提出以解决该问题。然而,在MEC环境中,边缘服务器可靠性较低,若发生设备故障会导致已有的卸载决策失效,使得应用程序响应时间增加,用户体验感降低。针对边缘服务器可能发生故障的问题,同时考虑到深度确定性策略梯度算法通过网络拟合策略函数,可以较好地应对高维动作空间的问题,提出了基于深度确定性策略梯度的服务器可靠性任务卸载策略。首先,通过复制子任务进行二次卸载的方式来降低应用执行的失败率;其次,将服务器可靠性约束下最小化应用时延的任务卸载和资源分配问题建模为马尔可夫决策过程;最后,利用基于深度确定性策略梯度的算法来求解任务卸载策略。仿真结果表明,SRTO-DDPG策略能有效地与环境交互并获得最优卸载决策,其性能优于本地执行策略,且相比基于DDPG的单卸载地点任务卸载策略,所提策略在可靠性约束下能实现低约26.16%的总延迟,能够更好地适应多服务器场景中边缘服务器的可靠性问题。

关键词:移动边缘计算;任务卸载;资源分配;深度强化学习;依赖性任务
中图法分类号 TP399

Server-reliability Task Offloading Strategy Based on Deep Deterministic Policy Gradient

LI Meng-fei[1], MAO Ying-chi[1,2], TU Zi-jian[1], WANG Xuan[1] and XU Shu-fang[1,2]
1 College of Computer and Information, Hohai University, Nanjing 210098, China
2 Key Laboratory of Water Big Data Technology of Ministry of Water Resources, Nanjing 210098, China

Abstract With the popularization of smart mobile devices, a new generation of mobile applications such as face recognition and virtual reality have gradually emerged. The limited computing power and battery capacity of mobile devices cannot support applications with high computing requirements and latency-sensitive applications. Therefore, mobile edge computing (MEC) is proposed to solve this problem. However, in the MEC environment, the reliability of the edge server is low, and the possible equipment failure will lead to the existing offloading decision failure, which increases the application response time and reduces the user experience. In view of the possible failure of edge servers, and considering that the deep deterministic policy gradient (DDPG) algorithm can better deal with the problem of high-dimensional action space through the network fitting strategy function, this paper proposes a server-reliability task offloading based on deep deterministic policy gradient (SRTO-DDPG). The main work is as follows. Firstly, the failure rate of application execution is reduced by duplicating subtasks for secondary offload. Secondly, the task offloading and resource allocation problems with server reliability constraints to minimize application delay are modeled as Markov decision process (MDP). Finally, an algorithm based on DDPG is used to solve the problem. Simulation results show that the SRTO-DDPG strategy can effectively interact with the environment to obtain the optimal offloading decision, and its performance is better than the local execution strategy (LE). Compared with the single location task offloading based on deep deterministic policy gradient (SLTO-DDPG), this strategy can achieve a low total delay of about 26.16% under reliability constraints, and can better adapt to the reliability problems of edge servers in multi-server scenarios.

Keywords Mobile edge computing, Task offloading, Resource allocation, Deep reinforcement learning, Dependent tasks

到稿日期:2021-06-04 返修日期:2021-12-07
基金项目:江苏省重点研发项目(BE2020729);姑苏创新领军人才专项(ZXL2020210);2020年昆山祖冲之攻关计划项目;中国华能集团关键技术项目(HNKJ19-H12,HNKJ20-H64)
This work was supported by the Key Research and Development Project of Jiangsu Province, China (BE2020729), Gusu Innovation Leading Talents Special Project (ZXL2020210), 2020 Kunshan Zu Chongzhi Key Tack Project and Key Technology Project of China Huaneng Group (HNKJ19-H12, HNKJ20-H64).
通信作者:毛莺池(yingchimao@hhu.edu.cn)

1 引言

各种新型移动应用的快速发展，如虚拟现实(VR)、增强现实(AR)和工业控制等应用场景[1]都要求实时完成计算密集型处理任务[2]。在以往的解决方案中，云计算通过将数据传输到远程云中心，利用云中心丰富的计算资源极大地提高了执行效率，但由于用户设备和云服务器在逻辑拓扑结构上距离较远，导致通信时延较长。此外，很多新型移动应用具有不可中断的特性，例如在工厂自动化应用中，任务执行需要极高的可靠性。为解决以上问题，移动边缘计算作为一种新模式开始兴起，它是一种将计算从云数据中心迁移到附近网络边缘的设备，扩展了终端设备的计算能力，以应对云计算方案中传输时间过长的问题。

考虑到移动边缘计算(Mobile Edge Computing, MEC)网络的复杂性，在实际场景中多边缘节点是常见的执行计算任务模式。与用户设备只和一台边缘服务器连接的情况相比，多个边缘服务器协同作业能进一步提升系统的整体性能，缩短应用程序延迟。然而，移动边缘计算环境网络的异构性使卸载过程可靠性降低，在卸载过程中可能遇到边缘服务器崩溃或无法访问的情况，而应用程序只有所有子任务都执行完成才能成功得到整个应用程序的运行结果。以上因素导致多个边缘服务器场景的出错概率更大，因此，在计算卸载过程中考虑边缘服务器的可靠性变得尤为重要。

为了应对边缘服务器的可靠性问题，本文提出基于DDPG的服务器可靠性任务卸载策略(Server-Reliability Task Offloading Based on Deep Deterministic Policy Gradient, SR-TO-DDPG)。其主要思想是将卸载子任务进行复制，并发送到不同的服务器，以提高任务执行的可靠性。在该策略场景中，任务可选择在本地执行，也可卸载到单个边缘服务器或复制后卸载到不同的两个边缘服务器。卸载过程大致为：首先基站接收任务信息作出卸载决策；然后选择卸载决策中需要二次卸载的子任务进行复制；最后将任务发送到相应的边缘服务器进行处理，以此减少边缘服务器故障带来的影响。

本文考虑一个多用户多任务服务器集群的移动边缘计算场景，如图1所示，边缘服务器集群通过基站与用户相连，区域内有 m 个边缘服务器可以为用户设备(User Equipment, UE)提供服务，表示为 $\delta=\{s_1,s_2,s_3,\cdots,s_m\}$，用户设备用集合表示为 $N=\{1,2,3,\cdots,N\}$。用户设备可以将任务卸载到任一正常运行的边缘服务器进行处理，或复制后卸载到不同的两个边缘服务器，同时也可以在本地执行任务。由于该场景中有多个边缘服务器，在卸载过程中，不同子任务可以选择不同边缘服务器卸载进行并行化处理来降低时延，同时子任务可以通过二次卸载的方法来提高执行可靠性。本文的主要贡献如下：

(1)提出利用二次卸载的方法来提高任务卸载的可靠性，建立可靠性约束模型，将卸载子任务复制后，分别发送到两个不同边缘服务器进行二次卸载，以此降低边缘服务器故障因素导致的卸载失败概率，提高执行可靠性。

(2)在多用户服务器集群场景中，将服务器可靠性约束下最小化时延的任务卸载与资源分配问题建模为马尔可夫决策过程(Markov Decision Process, MDP)。

(3)在动作空间较大的前提下，利用基于深度确定性策略梯度(Deep Deterministic Policy Gradient, DDPG)的算法求解任务卸载策略，仿真实验结果表明，SRTO-DDPG策略能有效地与环境交互从而获得最优卸载决策，其性能优于本地执行策略(Local Execution, LE)，且相比只有单卸载地点的任务卸载策略(Single Location Task Offloading based on Deep Deterministic Policy Gradient, SLTO-DDPG)，所提策略在可靠性约束下能实现低约26.16%的总延迟，能够更好地适应多服务器场景中边缘服务器的可靠性问题。

本文第2节介绍了相关工作；第3节介绍了服务器可靠性任务卸载问题模型；第4节给出了基于深度确定性策略梯度的服务器可靠性任务卸载算法；第5节进行了仿真实验和结果分析；最后总结全文。

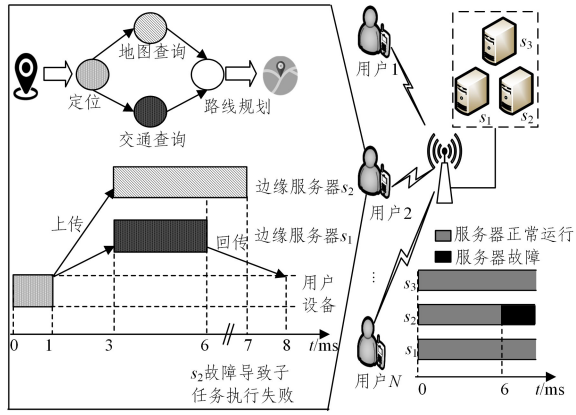


图1 多用户多服务器依赖性任务卸载场景

Fig. 1 Multi-user multi-server dependency task offload scenario

2 相关工作

根据卸载的边缘节点数量，可将卸载决策分为单边缘服务器节点的卸载和多边缘服务器节点的卸载。其中，单边缘节点卸载通常是在静态或准静态的场景下，而多边缘节点卸载的有关研究通常会考虑环境中的动态因素[3-4]，如用户移动性、无线链路连接、服务器可靠性等。

2.1 单服务器场景下的任务卸载

多用户单服务器会导致资源不足的情况发生，因此该场景下的研究常集中在用户之间的资源竞争上[5-6]，如无线信道资源、服务器计算资源等。Mao等[7]结合能量收集技术，提出了一种基于Lyapunov优化的在线优化计算卸载策略。Chen等[8]针对多用户场景中的信道干扰问题，提出了具有纳什均衡的算法。Cao等[9]考虑到个人设备难以获得某些信息，将多用户的计算卸载问题建模为非合作博弈，提出了一种基于机器学习的算法，以实现在完全分布式环境中的卸载。Huang等[10]提出了一个基于深度强化学习的在线卸载框架DROO，其目标是使所有无线设备的计算速率的加权和最大。

2.2 多服务器场景下的任务卸载

目前，一些研究将卸载场景扩展到多服务器场景，以适应MEC环境中任务量大、连接密度高以及应用程序的高时延

需求的特点。Guo 等[11]提出了一种启发式贪婪算法，该算法能显著降低移动设备能耗，且随着任务增加，趋势越明显。Zhu 等[12]综合考虑了工作流的完成时间和能耗，设计了用户体验函数，并提出了一种基于 DQN(deep Q-network)算法来求解最优卸载决策，以提高用户体验。Chen 等[13]提出了一种基于双深度 Q 网络的策略计算卸载算法，该算法能够选择将任务卸载到多个边缘服务器来达到最小化时延和任务丢弃率的目的。Huang 等[14]考虑了将任务卸载到边缘服务器可能产生的外部安全威胁因素，提出了一种在 MEC 环境中以风险概率为约束，目标为最小化延迟、最小化能耗以及最小化任务丢弃率的总体代价，将计算卸载问题建模成一个 MDP，利用 DQN 得到最优解。

2.3 讨论

目前卸载工作大多旨在降低应用程序时延和移动设备能量消耗，对于可靠性的研究较少。可靠性指一个应用程序成功完成调度的概率[15-16]，而在 MEC 环境中，每个边缘服务器都存在故障的风险，这极大程度地降低了卸载环境的执行可靠性。尤其对于一些高可靠性要求的安全关键型应用，如工业自动化[17]、增强现实等，执行可靠性是保证用户体验的重要因素。为了满足上述应用需求，目前有部分计算卸载工作在研究中关注了提高边缘计算架构的健壮性，结果发现导致任务执行失败的情况大致可以分为在无线链路通信的过程中出错[18-19]以及 MEC 服务器发生故障两种。针对以上问题，本文聚焦于多用户依赖性任务场景，并将服务器可靠性因素考虑到卸载决策的制定中。

3 服务器可靠性任务卸载问题建模

3.1 可靠性约束模型

为了提高任务执行过程中的可靠性，本文提出了二次卸载的卸载方案，使得应用程序子任务除了能够在本地执行和常规的单地点卸载之外，也能够将子任务复制后卸载到两个边缘服务器进行处理。在卸载过程中复制需要二次卸载的子任务，将原子任务和复制后的子任务卸载到两个不同的边缘服务器的二次卸载，能够降低该子任务执行时出错的概率。具体过程如图 2 所示。

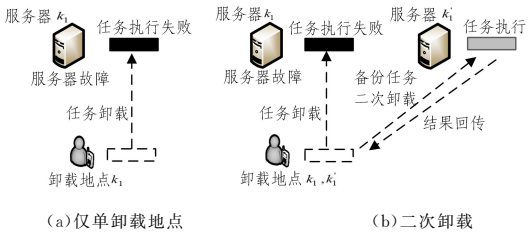


图 2 不同卸载方式下服务器故障后的卸载结果

Fig. 2 Uninstall results after server failure under different unloading modes

定义第 k 个边缘服务器的故障概率为 p_k^{err}，本文设置 p_k^{err} 为一个常量，并将其作为卸载决策需要满足的具体的可靠性约束，则当任务整体卸载到同一个边缘服务器 k 时，正常运行的概率是 $(1-p_k^{\mathrm{err}})$；若分别卸载到两个边缘服务器 k 和边缘服务器 l 时，两个服务器至少有一个正常运行能完成任务执行的概率为 $(1-p_k^{\mathrm{err}}p_l^{\mathrm{err}})>(1-p_k^{\mathrm{err}})$。

拓展到依赖关系任务的场景中，假设任务 T_1 有 n 个依赖关系子任务，只要其中一个子任务无法正常执行，T_1 就会执行失败。定义原任务的子任务卸载地点向量表示为 $\boldsymbol{k}=\{k_1,k_2,\cdots,k_n\}$，复制后的任务卸载的地点向量为 $\boldsymbol{k}'=\{k_1',k_2',\cdots,k_n'\}$，特别地，当 $k_x=0,1\leqslant x\leqslant n$ 时，则 $k_x'=0$，即卸载到本地的子任务不参与复制过程。

在不复制的情况下，应用程序在任一子任务无法执行的情况下无法正常运行，因此任务执行时，正常运行的概率为所有子任务正常执行概率之积。在仅单次卸载的情况下，应用程序故障概率为：

$$p_{\mathrm{once}}^{\mathrm{err}}=1-\prod_{x=1}^{n}(1-p_{k_x}^{\mathrm{err}}) \tag{1}$$

同样地，将场景拓展到二次卸载，即每个卸载子任务会同时卸载原子任务和复制后子任务到不同服务器。此时，应用程序无法执行，意味着其中一个或多个子任务的两次卸载均失败，即原子任务和复制后卸载的子任务被卸载到的两个边缘服务器同时发生故障。

以第 x 个子任务为例，它卸载到的两个地点为第 k_x 个边缘服务器和第 k_x' 个边缘服务器，则该子任务导致应用程序无法正常执行的概率为 $p_{k_x}^{\mathrm{err}}p_{k_x'}^{\mathrm{err}}$。此时，单个子任务正常执行的概率由 $(1-p_{k_x}^{\mathrm{err}})$ 变为 $(1-p_{k_x}^{\mathrm{err}}p_{k_x'}^{\mathrm{err}})$，特别地，当第 x 个子任务选择在本地进行处理，即 $k_x=k_x'=0$ 时，那么 $p_{k_x}^{\mathrm{err}}=p_{k_x'}^{\mathrm{err}}=0$，代表若第 x 个子任务不进行卸载处理，则应用程序不会因该子任务而执行出错；而若该子任务只进行单地点卸载，则设置 $k_x=k_x'\neq 0$，而该子任务正常执行的概率仍然为 $(1-p_{k_x}^{\mathrm{err}})$。本文研究的卸载决策需要满足执行可靠性约束。

3.2 系统模型

考虑到一个多用户多任务服务器集群的移动边缘计算场景，边缘服务器集群通过基站与用户相连，区域内有 m 个边缘服务器可以为用户设备提供服务，表示为 $\delta=\{s_1,s_2,s_3,\cdots,s_m\}$，用户设备用集合表示为 $N=\{1,2,3,\cdots,N\}$。用户设备可以将任务卸载到任一正常运行的边缘服务器进行处理，或复制后卸载到不同的两个边缘服务器，同时也可以在本地执行任务。其中，用户在进行多副本卸载时，将任务在两个服务器上都进行卸载，并且设计为两个边缘服务器同时开始处理，选择执行完成较快的或者任务执行成功的边缘服务器的执行结果进行回传。

3.2.1 任务模型

假设在每个时隙内，用户设备要执行一个计算密集型且时延敏感的应用程序级任务，不同的子任务可以根据卸载决策选择在本地执行，或是卸载到边缘处理。由此将粗粒度卸载问题转换成细粒度卸载问题，提高了系统并发性。应用程序 i 的整个任务被分解成 M_i 个有依赖关系的子任务，子任务间依赖关系可以用一个 DAG 图 $G=(T,E)$ 表示，子任务集可以表示为 $T_i=\{T_{i,1},T_{i,2},T_{i,3},\cdots,T_{i,M_i}\}$，$\forall i\in N$，每个子任务的参数可以描述为 $\{d_{i,j},c_{i,j}\}$，$\forall j\in M_i$，$c_{i,j}$ 表示完成任务 $T_{i,j}$ 的执行所需的 CPU 周期总数，$d_{i,j}$ 表示任务 $T_{i,j}$ 的输入

数据大小。子任务之间依赖关系的有向边集 $E_i=\{e_{j,l}(s_p,s_q)\}$，每条边 $e_{j,l}(s_p,s_q)$代表子任务 $T_{i,j}$是 $T_{i,l}$的前置子任务，$T_{i,l}$是 $T_{i,j}$的后继任务，即在 $T_{i,j}$完成前 $T_{i,l}$不能启动执行。同时，$T_{i,j}$和 $T_{i,l}$分别在边缘服务器 s_p 和 s_q 上执行，将用户设备同样视为 MEC 中的一台计算设备，定义为 $s_0 \circ e_{j,l}(s_p,s_q)$的值，代表两个子任务之间的数据传输量，当 $e_{j,l}(s_p,s_q)=0$ 时，代表这两个子任务之间没有依赖关系。用户设备 i 所要执行的应用程序级任务可以表示为 $T_i=(Task_i,Edge_i,D_i,C_i,Err_i)$，其中 $Task_i$ 和 $Edge_i$ 分别是子任务集合和子任务间的依赖关系集合，D_i 和 C_i 分别是子任务的输入数据大小 $d_{i,j}$和请求 CPU 周期数 $c_{i,j}$的集合，Err_i 表示不同应用程序的最大允许的故障概率，该值体现了应用程序的执行可靠性要求。

对于每个用户设备而言，用户能够进行细粒度的卸载，通过无线通信将不同子任务卸载到不同边缘服务器。在边缘服务器侧，同一个应用程序的子任务可以并行执行，极大程度地降低了时延。将应用程序 i 中第 j 个子任务的卸载决策表示为 $a_{i,j,k}$，则当 $a_{i,j,k}=1$ 时，代表该子任务被卸载到边缘服务器 s_k 执行。对于应用程序 i，整体卸载决策为：

$$a_i=\begin{bmatrix} a_{i,1,0},a_{i,1,1} & \cdots & a_{i,1,m} \\ \vdots & \cdots & \vdots \\ a_{i,M_i,0},a_{i,M_i,1} & \cdots & a_{i,M_i,m} \end{bmatrix} \tag{2}$$

为了提高执行的可靠性，选择卸载的子任务除了能够单次卸载到某个边缘服务器以外，还能够复制一份进行二次卸载，这时原子任务和复制的子任务会同时发送到两个不同的边缘服务器。

3.2.2 时延模型

(1)本地计算时延。假设用户设备有足够的任务执行所需的计算资源，任务可以选择在本地执行。此时，有 $a_{i,j,0}=1$。考虑在移动边缘计算环境中，用户设备计算能力恒定，但不同设备异构且计算能力不同，定义用户设备 i 在本地的计算能力为 f_i^l，因此，任务在本地运行的时长为：

$$t_{i,j}^l=\frac{c_{i,j}}{f_i^l} \tag{3}$$

(2)通信时延。在多个边缘服务器协作完成应用程序的过程中，若子任务 $T_{i,j}$和 $T_{i,l}$之间存在依赖关系，$T_{i,j}$是 $T_{i,l}$的前置子任务，两者之间的通信数据量为 $e_{j,l}(s_p,s_q)$，当 $T_{i,j}$和 $T_{i,l}$不在一个地点运行时，通信过程会产生相应的时延，分别为：1)当 $T_{i,j}$在本地，$T_{i,l}$选择卸载时，产生通过无线链路上传任务数据的通信延迟；2)当 $T_{i,l}$在本地，$T_{i,j}$选择卸载时，产生计算结果从边缘服务器传回用户设备的传输延迟；3)当 $T_{i,j}$和 $T_{i,l}$分别在不同两个边缘服务器上执行时，产生边缘服务器之间的通信延迟，传输延迟表示如下：

$$t_{jl,pq}^{\text{tran}}=\begin{cases} 0, & p=q \\ e_{j,l}(s_p,s_q)/r_{i,k}^u, & p=0,q>0 \\ e_{j,l}(s_p,s_q)/r_{i,k}^d, & p>0,q=0 \\ e_{j,l}(s_p,s_q)/r^b, & \text{otherwise} \end{cases} \tag{4}$$

其中，$r_{i,k}^u$和 $r_{i,k}^d$分别为用户设备 i 与第 k 个边缘服务器间数据上传和下载的速率，r^b 为两个边缘服务器之间的传输速率。

(3)边缘服务器计算时延。当任务到达边缘侧时，边缘服务器要为其分配计算资源。假设第 k 个边缘服务器提供给用户 i 的计算资源为 $f_{i,k}^e$，定义第 k 个边缘服务器计算能力为 F_k，$k\in S$。对于 $\forall k$，分配的资源都不能超过服务器自身的计算资源，则有：

$$\sum_{i=1}^{N} f_{i,k}^e \leqslant F_k \tag{5}$$

由于一个边缘服务器需要为多个用户设备提供计算资源，但其计算能力有限，因此会导致子任务间计算资源的争夺。假设每个边缘服务器维护 N 个边缘队列，每个边缘队列对应一个用户设备，边缘服务器在接收到卸载的任务后，会将它放入对应的队列中。边缘服务器会遵循先到先服务(First-In-First-Out，FIFO)的原则，先行处理到达时间较早的子任务，每个子任务的等待时间定义为 $t_{i,j}^{\text{wait}}$。此外，任务在第 k 个边缘服务器的计算时间表示为：

$$t_{i,j}^e=\frac{c_{i,j}}{f_{i,k}^e} \tag{6}$$

(4)应用程序总时延。定义 $t_{i,j}^{\text{start}}$和 $t_{i,j}^{\text{end}}$表示用户设备 i 第 j 个子任务的实际开始时间和实际完成时间，则设 $pre(T_{i,j})$和 $sub(T_{i,j})$分别是子任务 $T_{i,j}$的所有前置子任务和后继子任务集合。若某个子任务无前置子任务，则称其为起始子任务，用 $T_{i,j^{\text{in}}}$表示；若某个子任务无后继子任务，则为退出子任务，用 $T_{i,j^{\text{out}}}$表示。对于 $T_{i,j^{\text{in}}}$，有 $t_{i,j^{\text{in}}}^{\text{start}}=0$。

所有子任务的实际开始时间不会早于其任一前置子任务的完成时间，若该子任务选择卸载到边缘服务器，则还需等待到对应边缘队列为空才可执行。综上，$t_{i,j}^{\text{start}}$可以表示为：

$$t_{i,j}^{\text{start}}=\max\{t_{i,j}^{\text{queue}},\max_{j'\in pre(j)}(t_{i,j'}^{\text{end}}+t_{jj'}^{\text{tran}})\} \tag{7}$$

其中，$t_{i,j}^{\text{queue}}$是子任务对应边缘队列为空后能够执行 $T_{i,j}$的时间，$t_{jj'}^{\text{tran}}$是两个子任务之间的传输延迟。为了提高可靠性，选择卸载到边缘的子任务可能会在复制后发送给两个不同的边缘服务器。由于这两者不是同时完成，为进一步缩短延迟、降低计算量，只选择较早完成的子任务。假设 $T_{i,j}$选择二次卸载，其卸载决策为卸载到边缘服务器 k 和边缘服务器 l，则 $T_{i,j}$的实际完成时间为：

$$t_{i,j}^{\text{end}}=t_{i,j}^{\text{start}}+\min\left\{\frac{c_{i,j}}{f_{i,k}^e},\frac{c_{i,j}}{f_{i,l}^e}\right\} \tag{8}$$

最后整个应用程序的总时延可表示为：

$$T_i=t_{i,j^{\text{out}}}^{\text{end}}-t_{i,j^{\text{in}}}^{\text{start}} \tag{9}$$

3.3 问题建模

本文旨在为在可靠性约束下最小化所有用户应用程序的总时延，计算卸载问题可以转化为最小化时延问题，即：

$$\begin{aligned} &\min \sum_{i=1}^{N} T_i \\ &\text{s.t. } (1)a_{i,j,k}\in\{0,1\},\forall i\in M_i,\forall k\in m \\ &\quad (2)\sum_{k=0}^{m} a_{i,j,k}\in\{0,1,2\},\forall i\in N,\forall j\in M_i \\ &\quad (3)\sum_{i=1}^{N} f_{i,k}^e\leqslant F_k,\forall k \\ &\quad (4)p_i^{\max}\leqslant Err_i,\forall i \end{aligned} \tag{10}$$

其中，约束条件(1)和约束条件(2)表示用户设备的卸载结果限制，任务能够选择在本地执行或卸载进行处理，而选择卸载后子任务可能有一个或者两个对应的卸载边缘服务器地点；

约束条件(3)是边缘服务器侧的计算资源约束，表示每个边缘服务器分配给任务的计算资源都不能超过自身计算能力；约束条件(4)是用户设备的可靠性约束。根据 3.1 节提出的可靠性模型，以及根据卸载决策可以求得用户设备 i 卸载方案的任务失败概率 $p_i^{\max}$，此概率要小于规定的最大容忍失败概率 Err_i，以保证任务的执行可靠性满足用户需求。

4 基于深度确定性策略梯度的服务器可靠性任务卸载算法

本文将卸载场景扩展到了多台边缘服务器，同时子任务可以选择一次卸载和二次卸载两种卸载模式。相比单边缘服务器场景的卸载问题，本文的子任务卸载有更多可能的卸载情况。以 5 台边缘服务器为例，每个子任务卸载的结果有本地执行和卸载到其中一台边缘服务器，以及卸载到其中两台不同的边缘服务器，共计 $1+5+C_5^2=16$ 种情况。

本文提出了基于深度确定性策略梯度的服务器可靠性卸载策略，该算法通过网络管理者(Agent)来收集用户任务信息、实时上行和下行链路数据传输速率以及边缘服务器分配后剩余的计算资源，组合成系统状态，输出最优卸载方案。基于服务器可靠性的任务卸载算法过程如图 3 所示。

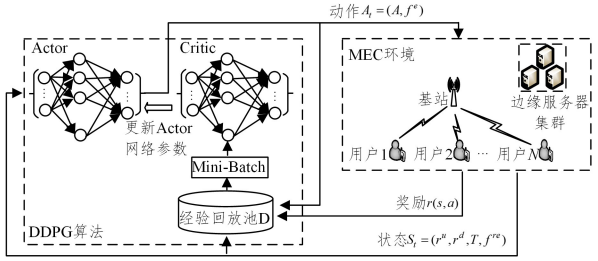


图 3 基于 DDPG 的服务器可靠性任务卸载算法

Fig. 3 Server reliability task offloading algorithm based on DDPG

4.1 算法设计思路

考虑到在动态的 MEC 环境中，传统算法无法根据时时变化的环境状态自适应制定卸载决策，而 RL 可以在无状态转移概率模型的情况下，有效地从经验中学习。因此，强化学习算法自然适用于在各种复杂的 MEC 场景中获得最优的卸载策略。本文利用深度强化学习这一特性，提出了基于深度确定性策略梯度的服务器可靠性卸载策略，该算法通过 DDPG 算法中的网络管理者来收集用户任务信息、实时上行和下行链路数据传输速率以及边缘服务器分配后剩余的计算资源，组合成系统状态输入到 DDPG 算法的 Actor 网络中，通过 DDPG 算法进行自主学习，实时动态地选出较为可靠的边缘服务器执行任务、使任务卸载执行时间最短的网络参数和任务 DAG 图执行的最短路径。通过实时地动态调整这些参数，以做出最优的任务卸载决策。本文将强化学习 3 要素定义如下。

(1)状态空间

时隙 t 的系统状态 S_t 由实时上行和下行链路数据传输速率 r^u 和 r^d 以及每个用户设备任务信息 T 和边缘服务器分配后剩余的计算资源 f^{re} 组成。状态空间表示为 $S=\{S_t \mid S_t=(r^u,r^d,T,f^{re})\}$。其中 r^u 和 r^d 包含所有用户设备连接到每个服务器的数据传输速率，r^u 可以表示为：

$$r^u=\begin{bmatrix} r^u_{1,1}, r^u_{1,2} & \cdots & r^u_{1,S} \\ \vdots & \cdots & \vdots \\ r^u_{N,1}, r^u_{N,2} & \cdots & r^u_{N,S} \end{bmatrix} \tag{11}$$

同理，可以得到 r^d。T 是每个用户设备的任务信息，包括任务间依赖关系、任务参数以及可靠性要求，表示为 $T=[T_1, T_2, \cdots, T_N]$。其中包含有执行任务的 DAG 图信息，有初始子任务和退出子任务，具体内部哪个子任务先执行强化学习会进行动态决策，只要满足输入的依赖关系就能学习出 DAG 图的最短路径，并最终输出最优卸载策略。f_k^{re} 为每个边缘服务器分配后剩余的计算资源，对于边缘服务器 k 来说，剩余的计算资源为 $f_k^{re}=F_k-\sum_{i=1}^{n} f_{i,k}^e$，则 $f^{re}=[f_1^{re}, f_2^{re}, \cdots, f_S^{re}]$。

(2)动作空间

系统根据当前的状态空间来选择合适动作，目标是最大化长期效益。本文的动作空间包括所有用户设备的卸载决策 A，以及边缘服务器给每个用户设备提供的计算资源 f^e。动作空间表示为 $A=\{A_t \mid A_t=(A,f^e)\}$，卸载动作 $A=[a_1, a_2, \cdots, a_N]$，其中 $a_i, i\in N$，表示用户设备 i 的整体的卸载决策向量，如式(2)所示。此外，动作空间还应该包括边缘服务器给每个用户设备提供的计算资源 f^e，表示如下：

$$f^e=\begin{bmatrix} f^e_{1,1}, f^e_{1,2} & \cdots & f^e_{1,S} \\ \vdots & \cdots & \vdots \\ f^e_{N,1}, f^e_{N,2} & \cdots & f^e_{N,S} \end{bmatrix} \tag{12}$$

(3)奖励函数

在状态 s_t 时选择动作 a_t，获得奖励值 r_t。本文的目标是在可靠性约束下最小化所有用户的应用时延，因此奖励函数应与应用延迟相关。具体而言，优化问题的目标是获得应用程序的最小完成时间，而强化学习的目的是获得最大奖励值。因此，$\sum_{i=1}^{N} T_i$ 越小意味着奖励值越大。此外，在 MEC 实际场景中，若不能保证应用程序的执行可靠性，会导致严重后果。为了满足问题中的可靠性约束，若任务可能失败的概率超过了规定阈值，即 $P_i^{\max}>Err_i$ 时，定义一个常数 $\rho<0$ 来做惩罚，可以得到奖励函数为：

$$r(s,a)=\begin{cases} \rho, & \text{if } \exists\, P_i^{\max}>Err_i \\ \varrho-\sum_{i=1}^{N} T_i, & \text{otherwise} \end{cases} \tag{13}$$

其中，ϱ 是一个始终大于所有应用延迟之和的正数，可以保证应用满足可靠性约束时始终为正反馈。

4.2 算法描述

DDPG 引入 DQN 中双网络的机制，建立了 4 个网(见图 4)。

(1)Actor 动作估计网络 μ：用于更新参数 θ，在状态 s 时根据该网络选择动作，获得下一时刻的状态 s' 和奖励值 r。

(2)Actor 动作目标网络 μ'：根据在经验回放池获取的样本，在 s' 状态时选择有最大值的 a'，其参数 θ' 在一定步数后由 θ 复制而来。

(3)Critic 状态估计网络 Q：用于更新参数 ω，同时当前 Q

值也是通过该网络得到的。

(4)Critic 状态目标网络 Q':用于计算目标 Q 值,其参数 ω' 在一定步数后由 ω 复制而来。

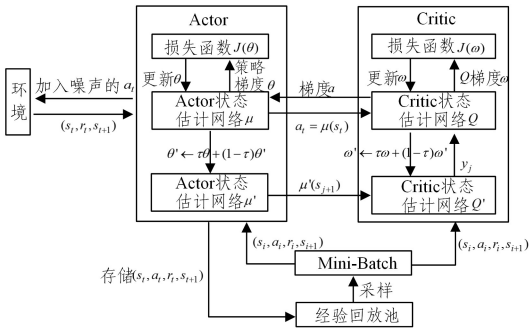


图 4 DDPG 算法的过程

Fig. 4 Process of DDPG algorithm

与 DQN 每隔几步将 Q_{predict} 参数赋值给 Q_{target} 不同,DDPG 每次都对目标网络中的参数进行少量更新,即:

$$\theta'\leftarrow\tau\theta+(1-\tau)\theta' \tag{14}$$

$$\omega'\leftarrow\tau\omega+(1-\tau)\omega' \tag{15}$$

其中,更新系数 τ 取值很小。此外,DDPG 还对采取的动作加入一定程度的噪声 $\mathcal{N}$,之后再与环境交互。对于 Critic 状态估计网络更新,使用类似 DQN 的均方差损失函数,即:

$$J(\omega)=\frac{1}{m}\sum_{j=1}^{m}(r_j+\gamma Q'(\phi_{j+1},\mu'(\phi_{j+1}|\theta');\omega')-Q(\phi_j,a_j;\omega))^2 \tag{16}$$

由于 Actor 的目的是选择合适的动作来获得更大的 Q 值,因此 Actor 动作估计网络的损失函数与反馈的 Q 值成反比,即:

$$J(\theta)=-\frac{1}{m}\sum_{j=1}^{m}Q(\phi_j,a_j;\omega) \tag{17}$$

DDPG 算法的具体描述如算法 1 所示,主要过程包括:首先,初始化所有神经网络的参数。其次,在每个时隙 t 根据策略 μ 选择动作,同时为了使学习过程增加随机性而加入噪声,执行动作得到奖励和下一状态。然后,将四元组存储到经验回放池 D 中,从中随机抽取 m 个样本,根据样本数据来分别更新两个网络。通过上述 $J(\omega)$ 和 $J(\theta)$ 分别更新 ω 和 θ。此外,按照参数 τ 的比例来更新网络参数 θ' 和 ω',并将其用于下一步预测策略和 Q 值,直至迭代过程结束。

算法 1 DDPG 算法

输入:$\theta,\omega,\theta',\omega',\gamma,\tau$,m,M,C

输出:θ,ω

随机初始化行动者网络 $\mu(s|\theta)$ 和评判家网络 $Q(s,a|\omega)$
初始化关联的目标网络,有权重 $\theta'\leftarrow\theta$ 和 $\omega'\leftarrow\omega$
初始化经验回放池 D
for each episode=1 to M do
  初始化序列 $s_1=\{x_1\}$ 和预处理序列 $\phi_1=\phi(s_1)$
  for each t=1 to T do
    $a_t=\mu(\phi_t|\theta)+\mathcal{N}$
    设置 $s_{t+1}=s_t$ 和预处理 $\phi_{t+1}=\phi(s_{t+1})$
    将 $m_t=(\phi_t,a_t,r_t,\phi_{t+1})$ 存放于 D
    从 D 中采样一个随机抽取的小批量样本 $(\phi_j,a_j,r_j,\phi_{j+1})$
    用梯度公式式(16)更新 ω
    用梯度公式式(17)更新 θ
    通过 $\theta'\leftarrow\tau\theta+(1-\tau)\theta'$ 和 $\omega'\leftarrow\tau\omega+(1-\tau)\omega'$ 每 C 步对目标网络进行更新
  end for
end for

5 仿真实验和结果分析

5.1 实验设置

5.1.1 实验参数设置

本文对基于服务器可靠性的多服务器任务卸载策略进行了仿真实验,考虑一个基站和多个边缘服务器的场景,用户设备随机分布在基站周围。主要仿真实验参数[20-23]如表 1 所列。

表 1 主要参数设置

Table 1 Main parameter settings

参数	值
用户设备数量 N	9
边缘服务器数量 m	6
每个应用程序子任务数量 M_i	[3,7]
子任务输入输出数据规模 $d_{i,j}$/kB	[100,1 500]
子任务所需 CPU 周期数 $c_{i,j}$/(M/cycles)	[100,200]
用户设备计算能力 f_i^l/GHz	1
上行链路速率 $r_{i,k}^u$/Mbps	[10,25]
下行链路速率 $r_{i,k}^d$/Mbps	[10,25]
服务器之间数据传输速率 r^b/Gbps	1
边缘服务器计算能力 F_k/GHz	[5,10]
边缘服务器故障率 p_k^{err}/%	0.1

为保证训练结果的公平性,对深度强化学习算法 DDPG 有关的参数进行了统一设置[24]。其中,Actor 和 Critic 的学习率 τ 分别为 0.0001 和 0.001,批量学习大小 $K=32$,目标网络的软更新率为 $\tau=0.01$,经验回放池大小 $D=100000$。

5.1.2 对比实验及评估指标

(1)对比实验

为了能够评估提出的基于服务器可靠性的任务卸载策略的性能,通过仿真实验实现了仅在本地设备执行的本地卸载策略(LE)、基于 DDPG 的单卸载地点的任务卸载策略(SLTO-DDPG)以及提出的基于 DDPG 的服务器可靠性的任务卸载策略(SRTO-DDPG)。全卸载策略不能够满足执行可靠性约束,因此在本文实验中不做对比。

(2)性能指标

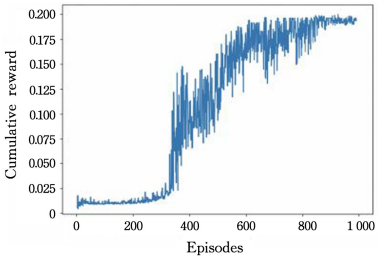
本文目标是在服务器可靠性约束下最小化时延,因此卸载决策的性能指标包括应用程序时延和执行失败率。其中,应用程序时延包括计算延迟和子任务间传输延迟,执行失败率表示由于边缘服务器故障而导致任务执行失败的概率,本次实验分为正常一次卸载方案的故障概率和改进后二次卸载方案的故障概率。

5.2 实验结果与分析

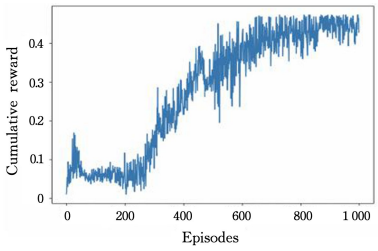
5.2.1 算法收敛性能

首先,对 SLTO-DDPG 和 SRTO-DDPG 算法的收敛性能进行对比实验,得到的累计报酬变化如图 5 所示。从图中可以看出,随着迭代次数的增加,DDPG 算法的累积回报不断

增加，SLTO-DDPG 算法和 SRTO-DDPG 算法大约都在 600 次迭代后收敛到某个范围内动荡，表明这两种算法自身训练所需耗时相差不大。此外，SLTO-DDPG 算法的累积回报在 0.15～0.20 间波动，而 SRTO-DDPG 算法累积回报在 0.35～0.45 间波动。SRTO-DDPG 算法明显获得了比 SLTO-DDPG 算法更高的累积回报，在算法性能方面表现更出色。



(a)SLTO-DDPG



(b)SRTO-DDPG

图 5　算法的收敛性能

Fig. 5　Convergence performance of algorithm

5.2.2　用户数量对卸载决策的影响

在不同数量的用户设备情况下($N=5,6,7,8,9$)，为保证结果的公平性，将所有方案都进行 100 次实验来获得平均值。在每次实验中，使用随机产生的相同一批参数，总时延结果如表 2 所列。

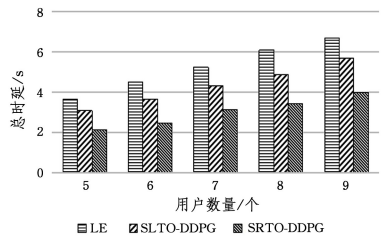
表 2　不同用户数量时的总时延

Table 2　Total delays with different numbers of users

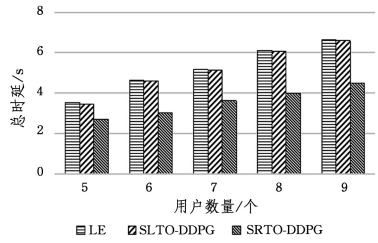
用户数量	$P^{\max}\leqslant 10^{-3}$			$P^{\max}\leqslant 10^{-5}$		
	LE	SLTO-DDPG	SRTO-DDPG	LE	SLTO-DDPG	SRTO-DDPG
5	3.67	3.09	2.15	3.52	3.57	2.72
6	4.51	3.65	2.49	4.65	4.60	3.02
7	5.23	4.31	3.13	5.17	5.14	3.62
8	6.08	4.81	3.45	6.11	6.18	3.98
9	6.69	5.67	3.98	6.63	6.89	4.51

从图 6 中可以看出，在规定应用容忍时延限制下，随着用户数量的增加，总时延增加。图 6(a)为应用容忍故障率为 10^{-3} 的情况，可以看到本地卸载有最高的总时延。此外，SLTO-DDPG 策略比 SRTO-DDPG 策略拥有更高的总时延。相比本地执行策略和 SLTO-DDPG 策略，SRTO-DDPG 策略能够分别减少约 44.47%和 31.23%的应用程序延迟。

图 6(b)为应用容忍故障率为 10^{-5} 时，不同卸载策略的总时延的情况，可以看出，本地执行策略 LE 和 SLTO-DDPG 的总时延相差不大。此时 SRTO-DDPG 策略通过复制子任务进行卸载能够实现边缘执行，相比另外两种方案，其总延迟降低了约 31.43%。



(a) $P^{\max}\leqslant 10^{-3}$



(b) $P^{\max}\leqslant 10^{-5}$

图 6　不同容忍故障率下用户数量对总时延的影响

Fig. 6　Influence of number of users on total delay at different tolerance failure rates

5.2.3　边缘服务器数量对卸载决策的影响

在不同数量的边缘服务器场景下($m=2,3,4,5,6$)，设置应用允许故障率为 10^{-3} 和 10^{-5} 进行以上所有卸载方案的性能对比实验。为保证结果的公平性，将所有方案都进行 100 次实验来获得平均值。在每次实验中，使用随机产生的相同一批参数，实验结果如表 3 所列。

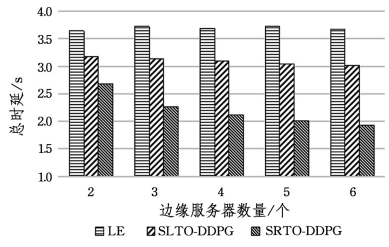
表 3　不同边缘服务器数量时的总时延

Table 3　Total delay for different number of edge servers

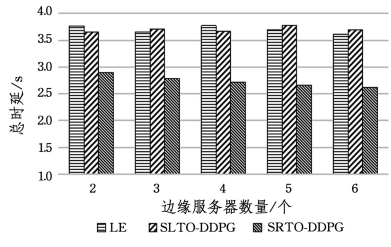
边缘服务器数量	$P^{\max}\leqslant 10^{-3}$			$P^{\max}\leqslant 10^{-5}$		
	LE	SLTO-DDPG	SRTO-DDPG	LE	SLTO-DDPG	SRTO-DDPG
2	3.64	3.17	2.68	3.76	3.65	2.89
3	3.72	3.13	2.26	3.65	3.71	2.78
4	3.69	3.09	2.12	3.77	3.67	2.71
5	3.73	3.04	2.01	3.69	3.78	2.66
6	3.67	3.01	1.93	3.61	3.69	2.62

图 7 为不同容忍故障率下边缘服务器数量对总时延的影响，可以看出，本地执行的总时延与边缘服务器数量无关。从图 7(a)中可以看出，随着边缘服务器数量的增加，SLTO-DDPG 策略的总时延缓慢下降，这是由于在执行可靠性约束下，能够卸载到边缘的任务数量不多，因此每个子任务能分配到的计算资源较多，时延变化不大。而 SRTO-DDPG 策略下降趋势比 SLTO-DDPG 更明显，且下降趋势也随着边缘服务器数量的增加而逐渐变缓，这是由于 SRTO-DDPG 策略通过二次卸载会将更多的子任务量卸载到边缘，因此需要更多的计算资源，而边缘服务器数量的增加使得每个子任务能够分配到的资源增加，时延迅速减小。

在图 7(b)中，由于可靠性要求较高，SLTO-DDPG 策略只能将子任务放在本地执行，因此总时延与本地执行策略接近。而相比容忍故障率为 10^{-3} 的情况，容忍故障率为 10^{-5} 时，SRTO-DDPG 策略的时延更高，这是因为随着可靠性要求的提高，SRTO-DDPG 策略二次卸载的子任务数量减少，导致总时延增加了 7.80%～36.27%。



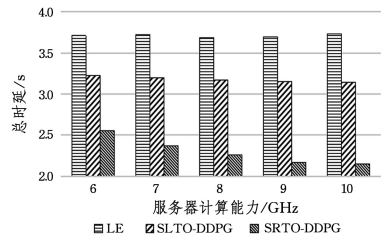
(a) $P^{max} \leqslant 10^{-3}$



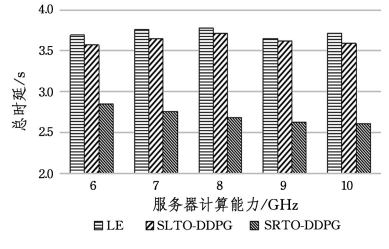
(b) $P^{max} \leqslant 10^{-5}$

图 7 不同容忍故障率下边缘服务器数量对总时延的影响

Fig. 7 Influence of number of edge servers on total delay at different tolerance failure rates



(a) $P^{max} \leqslant 10^{-3}$



(b) $P^{max} \leqslant 10^{-5}$

图 8 不同容忍故障率下服务器计算能力对总时延的影响

Fig. 8 Influence of server computing power on total delay at different tolerable failure rates

5.2.4 服务器计算能力对卸载决策的影响

在 4 台边缘服务器场景、不同服务器计算能力(F_k = 6GHz,7GHz,8GHz,9GHz,10GHz)下进行以上所有卸载方案的性能对比实验。为保证实验的公平性,对每种方案进行 100 次仿真实验得到平均值。在每次实验中,使用随机产生的同一批参数,结果如表 4 所列。

表 4 不同服务器计算能力下的总时延

Table 4 Total delay for different server computing power

服务器计算能力/GHz	$P^{max} \leqslant 10^{-3}$			$P^{max} \leqslant 10^{-5}$		
	LE	SLTO-DDPG	SRTO-DDPG	LE	SLTO-DDPG	SRTO-DDPG
6	3.71	3.23	2.55	3.69	3.57	2.85
7	3.72	3.20	2.37	3.76	3.65	2.76
8	3.69	3.17	2.26	3.78	3.72	2.68
9	3.70	3.15	2.17	3.65	3.62	2.63
10	3.73	3.14	2.15	3.71	3.59	2.61

图 8 为不同容忍故障率下服务器计算能力对总时延的影响,其中图 8(a)表示容忍故障率为 10^{-3} 的情况,可以看出,本地执行方案不受服务器故障率的影响,而 SLTO-DDPG 策略的总时延略有下降,这是由于在可靠性约束下,卸载到边缘的子任务数量不多,只有少部分子任务可以利用边缘计算能力,但随着服务器服务能力的提高,这部分子任务的执行时间缩短,因此总时延有所下降。该方案能够比本地执行增加约 13.86%~16.27% 的总时延。而对于 SRTO-DDPG 策略而言,随着边缘服务器计算能力的提升,总时延下降幅度减小,这是由于二次卸载极大程度地降低了故障概率,使得更多子任务可以卸载到边缘,从而充分利用了边缘计算能力,因此随着服务器计算能力的提升,时延下降较为明显。

图 8(b)为容忍故障率为 10^{-5} 的情况,此时几乎没有子任务可以通过单卸载地点的方式卸载到边缘,导致本地执行和 SLTO-DDPG 策略结果接近,而较小的故障率同样约束了二次卸载的子任务个数,SRTO-DDPG 策略的总时延低于 SLTO-DDPG 策略约 20.17%~27.30%。

结束语 本文在 MEC 多用户服务器集群的场景中,考虑服务器可能存在故障的情况,提出了基于深度确定性策略梯度的服务器可靠性任务卸载策略。首先,根据 MEC 环境中边缘服务器发生故障可能性高这一特点,通过复制子任务进行二次卸载的方式,来降低卸载失败的概率,从而提高任务执行可靠性;其次,将服务器可靠性约束下最小化时延的任务卸载与资源分配问题建模为 MDP 模型;最后,提出了基于 DDPG 的服务器可靠性任务卸载算法,以解决动作空间过大的问题。仿真实验结果表明,SRTO-DDPG 策略能有效地与环境交互,获得最优卸载决策,其性能优于本地执行策略(LE),且相比只有单卸载地点的任务卸载策略(SLTO-DDPG),所提策略在可靠性约束下能实现低约 26.16% 的总延迟,能够更好地解决多服务器场景中边缘服务器的可靠性问题。

参 考 文 献

[1] AL-JANABI S, AL-SHOURBAJI I, SHOJAFAR M, et al. Mobile Cloud Computing: Challenges and Future Research Directions[C] // International Conference on the Developments on Systems Engineering. IEEE, 2017: 62-67.

[2] RANADHEERA S, MAGHSUDI S, HOSSAIN E. Mobile Edge Computation Offloading Using Game Theory and Reinforcement Learning[J]. arXiv: 1711.09012, 2017.

[3] XU C B, LIU Y, LIU Y X, et al. MEC server selection scheme based on multiple indicators [J]. Journal of Chongqing University of Posts and Telecommunications (Natural Science Edition), 2020, 32(3): 329-335.

[4] LIANG Y C, CAO B. Artificial neural network methods for task offloading in VANET cloud [J]. Journal of Chongqing University of Posts and Telecommunications (Natural Science Edition), 2020, 32(3): 336-344.

[5] LI J, ZHANG Y P, PANG L, et al. Joint Resource Allocation and Task Scheduling in Mobile Edge Computing[J]. Journal of

Chongqing University of Technology(Natural Science),2020,34(11):156-163.

[6] CHEN L. Multicast resource allocation algorithm based on layered coding in sparse code multiple access systems[J]. Journal of Chongqing University of Posts and Telecommunications(Natural Science Edition),2020,32(6):917-924.

[7] MAO Y Y,ZHANG J,BEN K,et al. Dynamic Computation Offloading for Mobile-Edge Computing with Energy Harvesting Devices[J]. IEEE Journal on Selected Areas in Communications,2016,34(12):3590-3605.

[8] CEHN X,JIAO L,LI W,et al. Efficient Multi-User Computation Offloading for Mobile-Edge Cloud Computing[J]. IEEE/ACM Transactions on Networking,2016,24(5):2795-2808.

[9] CAO H,CAI J. Distributed Multiuser Computation Offloading for Cloudlet-Based Mobile Cloud Computing:A Game-Theoretic Machine Learning Approach[J]. IEEE Transactions on Vehicular Technology,2018:752-764.

[10] HUANG L,BI S,ZHANG Y J A. Deep Reinforcement Learning for Online Computation Offloading in Wireless Powered Mobile-Edge Computing Networks[J]. IEEE Transactions on Mobile Computing,2019,19(11):2581-2593.

[11] GUO H,LIU J,ZHANG J. Computation Offloading for Multi-Access Mobile Edge Computing in Ultra-Dense Networks[J]. IEEE Communications Magazine,2018,56(8):14-19.

[12] ZHU A,GUO S,MA M,et al. Computation Offloading for Workflow in Mobile Edge Computing Based on Deep Q-Learning[C]// Wireless and Optical Communications Conference. IEEE,2019:1-5.

[13] CHEN X,ZHANG H,WU C,et al. Optimized Computation Offloading Performance in Virtual Edge Computing Systems via Deep Reinforcement Learning[J]. IEEE Internet of Things Journal,2019,6(3):4005-4018.

[14] HUANG B,LI Y,LI Z,et al. Security and Cost-Aware Computation Offloading via Deep Reinforcement Learning in Mobile Edge Computing[J]. Wireless Communications and Mobile Computing,2019,2019:1-20.

[15] SHARMA Y,JAVADI B,SI W,et al. Reliability and Energy Efficiency in Cloud Computing Systems:Survey and Taxonomy[J]. Journal of Network and Computer Applications,2016,74:66-85.

[16] ALSENANI Y,CROSBY G,VELASCO T. SaRa:A Stochastic Model to Estimate Reliability of Edge Resources in Volunteer Cloud[C]// 2018 IEEE International Conference on Edge Computing(EDGE). IEEE,2018:121-124.

[17] LORENZO B,GARCIA-ROIS J,LI X,et al. A Robust Dynamic Edge Network Architecture for the Internet-of-Things[J]. IEEE Network,2018,32(1):8-15.

[18] AL-HABOB A A,IBRAHIM A,DOBRE O A,et al. Collision-Free Sequential Task Offloading for Mobile Edge Computing[J]. IEEE Communications Letters,2020,24(1):71-75.

[19] LIU J,ZHANG Q. Offloading Schemes in Mobile Edge Computing for Ultra-Reliable Low Latency Communications[J]. IEEE Access,2018,16:12825-12837.

[20] HUANG L,FENG X,QIAN L,et al. Deep Reinforcement Learning-Based Task Offloading and Resource Allocation for Mobile Edge Computing[C]// International Conference on Machine Learning and Intelligent Communications. 2018:33-42.

[21] LI J,GAO H,LV T J,et al. Deep Reinforcement Learning based Computation Offloading and Resource Allocation for MEC[C]// Wireless Communications and Networking Conference. IEEE,2018:1-6.

[22] SHU C,ZHAO Z,HAN Y,et al. Dependency-Aware and Latency-Optimal Computation Offloading for Multi-User Edge Computing Networks[C]// 2019 16th Annual IEEE International Conference on Sensing,Communication,and Networking. IEEE,2019:1-9.

[23] LIU J,ZHANG Q. Code-Partitioning Offloading Schemes in Mobile Edge Computing for Augmented Reality[J]. IEEE Access,2019,7:11222-11236.

[24] CHEN Z,WANG X. Decentralized Computation Offloading for Multi-User Mobile Edge Computing:A Deep Reinforcement Learning Approach[J]. Journal on Wireless Communications and Networking,2020,188:1-21.

LI Meng-fei, born in 1998, postgraduate, is a member of China Computer Federation. Her main research interests include distributed computing, IoT and edge intelligence computing.

MAO Ying-chi, born in 1976, Ph.D, professor, is a senior member of China Computer Federation. Her main research interests include distributed computing and edge computing.

(责任编辑:何杨)