计算机应用研究
Application Research of Computers
ISSN 1001-3695,CN 51-1196/TP

《计算机应用研究》网络首发论文

题目：基于 SAC 的多服务移动边缘计算中任务卸载和资源配置算法
作者：彭姿馀，王高才，农望
DOI：10.19734/j.issn.1001-3695.2022.08.0443
收稿日期：2022-08-06
网络首发日期：2023-01-19
引用格式：彭姿馀，王高才，农望．基于 SAC 的多服务移动边缘计算中任务卸载和资源配置算法[J/OL]．计算机应用研究．
https://doi.org/10.19734/j.issn.1001-3695.2022.08.0443

基于 SAC 的多服务移动边缘计算中任务卸载和资源配置算法 *

彭姿馀，王高才†，农　望

(广西大学 计算机与电子信息学院，南宁 530004)

摘　要：多服务移动边缘计算网络环境中的不同服务的缓存要求、受欢迎程度、计算要求以及从用户传输到边缘服务器的数据量是随时间变化的。如何在资源有限的边缘服务器中调整总服务类型的缓存子集，并确定任务卸载目的地和资源分配决策，以获得最佳的系统整体性能是一个具有挑战性的问题。为了解决这一难题，首先将优化问题转换为马尔可夫决策过程，然后提出了一种基于软演员-评论家(Soft Actor-Critic，SAC)的深度强化学习算法来同时确定服务缓存和任务卸载的离散决策以及上下带宽和计算资源的连续分配决策。算法采用了将多个连续动作输出转换为离散的动作选择的有效技巧，以应对连续-离散混合行动空间所带来的关键设计挑战，提高算法决策的准确性。此外，算法集成了一个高效的奖励函数，增加辅助奖励项来提高资源利用率。广泛的数值结果表明，与其他基线算法相比，提出的算法在有效地减少任务的长期平均完成延迟的同时也具有良好的稳定性。

关键词：移动边缘计算；服务缓存；任务卸载；资源分配

中图分类号：TP301.6　　doi: 10.19734/j.issn.1001-3695.2022.08.0443

Sac based algorithm for task offloading and resource provisioning in multiple-services mobile edge computing

Peng Ziyu, Wang Gaocai†, Nong Wang

(*School of Computer & Electronic Information*, *Guangxi University*, *Nanning* 530004, *China*)

Abstract: In a Multiple-Services mobile edge computing network environment, the requirements and popularity of services, the computing requirement and the amount of data transferred from users to edge servers are dynamic with time. How to adaptively adjust the caching subset of total service types in the resource-limited edge server and determine the task offloading destination and resource allocation decisions to obtain the optimum overall system performance is a challenge problem. To solve this challenge, this paper firstly converted it into a Markov decision process, then proposed a soft actor–critic (SAC) deep reinforcement learning-based algorithm to jointly determine not only the discrete decisions of service caching and task offloading but also the continuous allocation of bandwidth and computing resource. The algorithm employed an effective trick of converting multiple continuous action outputs into discrete action choices to deal with the key design challenge that arises from continuous-discrete hybrid action space while improving the accuracy of algorithm decisions. Additionally, the algorithm integrated an efficient reward function that adds auxiliary reward terms to improve resource utilization. Extensive numerical results show that compared with other baseline algorithms, the proposed algorithm can effectively reduce the long-term average completion delay of tasks while accessing excellent performance in terms of stability.

Key words: mobile edge computing; service caching; task offloading; resource provisioning

0　引言

移动边缘计算(mobile edge computing, MEC)[1]已被广泛应用于不同的应用场景[2~4]，旨在为用户提供更好的服务质量(quality of service, QoS)。然而，随着支持 5G 应用的增长，不断增加的处理具有不同服务要求的任务的需求给边缘网络带来了巨大的压力。多服务移动边缘计算(multiple-services mobile edge computing, MSs-MEC)是一种新型的移动边缘计算网络环境，它结合了服务缓存技术和 MEC，能自适应调整每个时隙的服务缓存决策，使部署在用户侧的边缘服务器能够灵活处理属于不同服务类型的任务，提高服务器资源的利用率。

与传统的云计算相比，MEC 通过将任务卸载到与用户距离相近的边缘服务器来提升用户设备性能。为了减少任务卸载延迟，文献[5]研究了由支持多种无线接入技术的异构边缘服务器组成的 MEC 系统，通过马尔可夫决策过程找到最佳卸载位置。文献[6]提出了一种基于深度强化学习的无模型在线动态计算卸载方案，将区块链授权的移动边缘计算中的数

收稿日期：2022-08-06；**修回日期：**2022-10-31　　**基金项目：**移动边缘计算中具有能耗优化的数据迁移策略研究(62062007)

作者简介：彭姿馀(1997-)，男，广西贵港人，硕士研究生，主要研究方向为移动边缘计算、无线网络；王高才(1976-)，男(通信作者)，广西桂林人，教授，博导，主要研究方向为计算机网络、系统性能评价和随机方法(wanggegx@163.com)；农望(1994-)，男(壮族)，广西百色人，硕士研究生，主要研究方向为移动边缘计算.

据挖掘和数据处理等任务卸载问题建模为马尔可夫决策过程，使系统的长期卸载性能最大化。文献[7]将混合光纤无线网络引入边缘计算，以解决 MEC 网络单一接入方式带来的高拥堵和高能耗问题。文献[8]研究了卫星网络边缘计算的多低轨卫星计算卸载问题，利用拉格朗日乘子法和凸优化方法，获得无线资源分配和卸载决策的最优策略。

尽管任务的卸载能保证用户任务被顺利执行，但突变的网络环境、有限的设备能量等因素能极大地影响用户的卸载体验。因此，系统必须能够合理利用用户设备和边缘服务器上的有限资源，针对不同的环境状态对资源进行合理调控。例如，文献[9]考虑了任务生成的动态性和连续性，提出了一种基于深度强化学习的资源管理方案，使平均任务延迟最小。文献[10]提出了一种基于斯塔克尔伯格动态博弈的资源定价和交易方案，以优化边缘计算站和无人机之间的资源分配，从而在无人机网络中为用户提供更高质量和更满意的服务。文献[11]开发了一个基于深度强化学习的协同计算卸载和资源分配框架，共同优化基于区块链的移动边缘计算系统中的卸载决策、功率分配、区块大小和区块间隔，以提高任务处理速度和交易吞吐量。文献[12]研究了移动边缘计算的区块链资源分配问题，建立边缘计算能耗迁移与资源分配联合优化模型，并使用蚁群算法寻找最低能耗的资源分配解。

处在同一基站覆盖范围内的用户会存在不同的计算需求，缺乏对不同类型任务的处理能力成为 MEC 限制任务卸载的优势发挥的关键因素。为此，将某类任务相关的代码库或数据库缓存到服务器中来保证对应类型的任务能被执行的服务缓存技术已经被一些工作考虑到。例如，为了解决 MEC 系统中的服务异质性、未知系统动态、空间需求耦合和分散协调等问题，文献[13]提出了一种高效的在线算法来联合优化动态服务缓存和任务卸载。文献[14]研究了服务缓存和任务的依赖性卸载问题，并设计了一个基于凸优化的卸载方案，以降低任务完成成本。文献[15]提出了一种基于正则化技术和李雅普诺夫优化理论的优化方案，共同优化服务缓存和计算卸载问题，以实现系统利润最大化。为了优化具有异质任务请求、应用数据预存储和基站协调的 MEC 系统中的计算卸载、服务缓存和资源分配问题，文献[16]提出了一种高效的协同服务缓存和计算卸载算法，以最小化任务的平均执行时间，文献[17]提出了一种基于半有限松弛方法和交替优化的高效近似算法，以最小化所有用户的整体计算和延迟成本。

深度强化学习(Deep Reinforcement Learning, DRL)作为强化学习和高容量函数近似器的结合，已经成功地应用于广泛的挑战性领域，如路径规划和机器人控制。由于机器学习在函数逼近方面的突破，DRL 算法可以利用神经网络强大的感知和表征能力，准确识别状态之间的差异[18]，准确地掌握问题的解空间。因此，采用深度强化学习来解决 MEC 中的优化问题是一种强有力的方法。例如，文献[19]开发了一种基于深度 Q-learning 算法的车对车通信的新型分散资源分配机制，为传输找到最佳子频段和功率水平。文献[12]提出了一个在线深度强化学习方案，以寻找在时变的无线信道环境下的最佳的任务卸载决策和无线资源分配。

将 DRL 算法整合到 MEC 领域的大多数工作[21~23]，主要集中在任务卸载或资源分配问题上，较少考虑到服务缓存、任务卸载以及资源分配的联合决策问题。此外，在处理大规模连续-离散混合行动空间的问题上，传统的 DRL 算法，如深度 Q 网络[24]、深度确定性策略梯度[25]，仅关注离散或连续行动空间，无法处理混合动作空间的决策问题且动作决策空间容易受到维度诅咒的影响。为此，本文结合将离策略更新纳入演员-评论家框架中的 SAC 算法，通过把部分连续动作输出最大化作为其中离散的选择动作，提出在异构服务、任务动态生成和有限资源约束条件下，同时考虑离散的缓存与卸载目的地选择和连续的资源分配的服务缓存、任务卸载以及资源配置的算法(soft actor-critic based algorithm for service caching, task offloading and resource provisioning, SACSTR)，实现在最小化任务平均完成时延的同时保证稳定的训练过程。

1　模型与问题描述

1.1　任务模型

支持 MSs-MEC 的系统模型如图 1 所示，包括一个 ES，一个 CC，N 个用户，K 种不同类型服务，其中 $N=\{1,2...N\}$，$K=\{1,2...K\}$。系统中的 ES 的存储和计算能力有限，只能缓存一个服务子集，如果任务对应的服务类型没有被缓存，该任务则会被转发到 CC 执行。本文应用一个时隙模型，表示为 $t\in\{1,2...T\}$，其中时隙 t 与服务缓存决策的更新时间尺度相匹配[12]。在时隙 t，用户 $i\in N$ 产生了一个独立的属于服务 $j\in K$ 的任务，用 $x_{ij}(t)\in\{0,1\}$ 表示，若用户 i 在时隙 t 产生了属于服务 j 的任务则 $x_{ij}(t)=1$，否则 $x_{ij}(t)=0$。

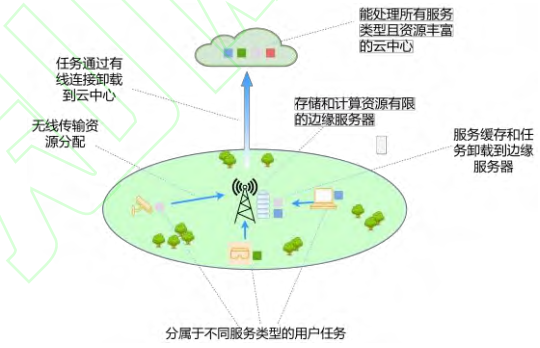

图 1　MSs-MEC 支持的系统架构

Fig. 1　The mss-MEC-enabled system architecture

从系统的角度来看，系统中的每个用户都会在每个时隙中动态地产生属于某种服务类型的任务，也就是说，在时隙 t 中，系统会产生 N 个属于不同服务类型的任务。与大多数现有的考虑每个任务的卸载决策的工作不同，本文的决策问题是针对系统中的每项服务进行优化。设 $Q_j(t)$ 表示用户在时隙 t 产生的属于服务 j 的任务，$w_j(t)$ 表示 $Q_j(t)$ 的计算总需求，$z_j^U(t)$ 和 $z_j^D(t)$ 分别代表 $Q_j(t)$ 的输入数据大小和计算结果大小，$w_{ij}(t)$ 表示用户 i 产生的服务 j 的任务的计算需求，$w_{ij}(t)=0$ 表明用户 i 在时隙 t 产生的任务不属于服务 j。$z_{ij}^U(t)$ 和 $z_{ij}^D(t)$ 分别表示用户 i 生成的属于服务 j 的任务的输入数据大小和结果数据大小。$w_j(t)$、$z_j^U(t)$、$z_j^D(t)$ 可以分别由式(1)~(3)计算得出：

$$\sum_{i=1}^{N}x_{ij}(t)w_{ij}(t)=w_j(t) \tag{1}$$

$$\sum_{i=1}^{N}x_{ij}(t)z_{ij}^U(t)=z_j^U(t) \tag{2}$$

$$\sum_{i=1}^{N}x_{ij}(t)z_{ij}^D(t)=z_j^D(t) \tag{3}$$

在每个时隙 t，服务将被决定是否在 ES 中缓存，服务对应的任务将被决定是否卸载到 ES 中执行。由于 ES 的空间有限，只有部分服务可以被缓存，因此，必须根据每个时隙的任务需求，智能地选择服务进行缓存。$a_j(t)\in\{0,1\}$ 用来表示 ES 在时隙 t 中是否缓存了服务 j，若 ES 在时隙 t 中缓存了

服务 j 则 $a_{ij}(t)=1$，否则 $a_{ij}(t)=0$。假设 ES 的最大缓存容量为 R，安装服务 j 占用的空间为 r_j，服务缓存需要满足以下约束条件式(4)：

$$\sum_{j=1}^{K} a_j(t) r_j \le R \tag{4}$$

1.2 传输模型

每个时隙产生的任务都需要卸载到 ES 或 CC 上执行，因此需要为任务分配带宽资源，用于数据的上传和下载。$b_{ij}^{U}(t)$ 和 $b_{ij}^{D}(t)$ 分别表示任务 $x_{ij}(t)=1$ 在时隙 t 获得的上传和下载带宽。同样，$b_j^{U}(t)$ 和 $b_j^{D}(t)$ 分别用来表示分配给任务 $Q_j(t)$ 的上行和下行带宽。$b_j^{U}(t)$、$b_j^{D}(t)$ 与 $b_{ij}^{U}(t)$、$b_{ij}^{D}(t)$ 的对应关系可以分别描述为式(5)和(6)：

$$b_{ij}^{D}(t)=\frac{z_{ij}^{D}(t)}{z_j^{D}(t)} b_j^{D}(t) \tag{5}$$

$$b_{ij}^{U}(t)=\frac{z_{ij}^{U}(t)}{z_j^{U}(t)} b_j^{U}(t) \tag{6}$$

则属于服务 j 的任务获得的上行和下行带宽可以分别表示为式(7)和(8)：

$$\sum_{i=1}^{N} b_{ij}^{U}(t)=b_j^{U}(t) \tag{7}$$

$$\sum_{i=1}^{N} b_{ij}^{D}(t)=b_j^{D}(t) \tag{8}$$

$Q_j(t)$ 从用户到 ES 的上传和下载时间可以分别由式(9)和(10)计算得出。

$$T_j^{U}(t)=\frac{z_j^{U}(t)}{\eta_j^{U} b_j^{U}(t)} \tag{9}$$

$$T_j^{D}(t)=\frac{z_j^{D}(t)}{\eta_j^{D} b_j^{D}(t)} \tag{10}$$

其中 η_j^{U} 和 η_j^{D} 分别为上行链路和下行链路的频谱效率，其数值可以用香农公式 $log(1+SNR)$ 来近似，SNR 为用户设备与 ES 之间的信道噪声比，类似的假设参考文献[17]。设 B^{U} 和 B^{D} 分别表示系统中的总上传和下载带宽容量，带宽的分配需要满足式(11)和(12)表示的约束条件。

$$\sum_{j=1}^{k} b_j^{U}(t) \le B^{U} \tag{11}$$

$$\sum_{j=1}^{k} b_j^{D}(t) \le B^{D} \tag{12}$$

当 CC 被选为卸载的目的地时，任务首先通过无线通道传输给 ES，然后由 ES 转发到 CC。与无线连接相比，ES 和 CC 之间的连接通常是一个稳定的、大容量的有线连接。因此，从 ES 到 CC 的传输速度可以被认为是一个固定值。与文献[15]相似，本文假设 ES 和 CC 之间的上行传输速率与 ES 和 CC 之间的下行传输速率相等，两者都用 b^{ec} 表示。任务 $Q_j(t)$ 从 ES 到 CC 的上传和下载时间可以表示为 T_j^{uec} 和 T_j^{dec}，它们可以分别描述为式(13)和(14)：

$$T_j^{uec}=\frac{z_j^{U}(t)}{b^{ec}} \tag{13}$$

$$T_j^{dec}=\frac{z_j^{D}(t)}{b^{ec}} \tag{14}$$

任务 $Q_j(t)$ 可以在 ES 缓存了服务 j 所需要的执行环境后被卸载到 ES 上执行。设 $o_j(t)\in\{0,1\}$ 代表 $Q_j(t)$ 的卸载决策，$o_j(t)=1$ 表示 $Q_j(t)$ 被卸载到 ES 上执行，否则 $Q_j(t)$ 将被卸载到 CC 上。由于服务缓存是在 ES 上执行任务的前提条件，系统需要满足式(15)表示的约束条件：

$$o_j(t) \le a_j(t) \tag{15}$$

1.3 计算模型

当一个任务在时隙 t 被卸载到 ES 时，系统将根据当前系统状态为该任务分配执行所需的计算资源。将任务 $Q_j(t)$ 获得的计算资源和用户 i 在时隙 t 产生的服务 j 任务获得的计算资源分别表示为和 $f_{ij}^{e}(t)$，其关系可表述为(16)：

$$f_{ij}^{e}(t)=\frac{w_{ij}(t)}{w_j(t)} f_j^{e}(t) \tag{16}$$

由于 ES 的计算资源有限，分配给任务的计算资源之和不能超过其总的计算能力。令 f^{E} 代表 ES 的总的计算能力，分配的计算资源需遵循(17)表示的约束：

$$\sum_{j=1}^{K} f_j^{e}(t) \le f^{E}\ f_j^{e}(t) \tag{17}$$

$T_j^{e}(t)$ 表示属于服务 j 的任务被卸载到 ES 时的处理时间，那么 $T_j^{e}(t)$ 可由式(18)计算得出：

$$T_j^{e}(t)=\frac{w_j(t)}{f_j^{e}(t)} \tag{18}$$

由于 CC 包括所有类型的服务，并且有足够的计算能力，假设 CC 的计算速率是一个常数，用 f^{c} 表示，让 $T_j^{c}(t)$ 代表任务 $Q_j(t)$ 的处理时间，$T_j^{c}(t)$ 可以表示为(19)：

$$T_j^{c}(t)=\frac{w_j(t)}{f^{c}} \tag{19}$$

1.4 问题描述

本文的优化目标是在一段时间内最小化任务的平均完成延迟。其中，当服务 j 的任务被卸载到 ES 时，其处理延迟由 $d_j^{e}(t)$ 表示，可以表示为式(20)：

$$d_j^{e}(t)=T_j^{U}(t)+T_j^{D}(t)+T_j^{e}(t) \tag{20}$$

同样，当服务 j 的任务被卸载到 CC 时，其处理延迟为 $d_j^{c}(t)$，可描述为式(21)：

$$d_j^{c}(t)=T_j^{U}(t)+T_j^{D}(t)+T_j^{c}(t)+T_j^{uec}(t)+T_j^{dec}(t) \tag{21}$$

在每个时隙，系统需要确定每个服务类型的 $b_j^{U}(t)$、$b_j^{D}(t)$、$a_j(t)$、$o_j(t)$、$f_j^{e}(t)$，以最小化任务的长期平均延迟。对于时隙 t，用 $d(t)$ 表示的系统任务的完成延迟，且可以将其定义为式(22)：

$$d(t)=\sum_{j=1}^{K}\left[a_j(t)o_j(t)d_j^{e}(t)+(1-o_j(t))d_j^{c}(t)\right] \tag{22}$$

在系统中，将需要优化的目标转换为 T 个时间段内的优化问题，表述为

$$P1:\min_{b_j^{U}(t),b_j^{D}(t),a_j(t),o_j(t),f_j^{e}(t)} \sum_{t=1}^{T} d(t)$$

$$\text{s.t. } C1:(4),(11),(12),(15),(17)$$

$$C2:\begin{cases} f_j^{e}(t)>0, a_j(t)o_j(t)=1, \forall j\in K \\ f_j^{e}(t)>0, otherwise \end{cases}$$

$$C3: b_j^{U}(t), b_j^{D}(t)>0\ \forall j\in K$$

$$C4: a_j(t), o_j(t)\in\{0,1\}\ \forall j\in K$$

其中 C1 约束中的式(4)表示缓存服务所占用的总存储空间不能超过 ES 的总缓存容量。式(11)和(12)表明，在每个时隙分配给每个服务的上行和下行带宽不能超过系统的总带宽。式(15)表明，当相应的服务没有在 ES 中缓存时，任务不能被卸载到 ES 中。式(17)表示分配的计算资源之和不能超过 ES 所拥有的可分配计算资源。C2 表明，只有当任务被卸载到 ES 上执行时，ES 才会为任务分配计算资源。C3 表示系统应该为每个任务分配带宽资源用于数据传输。C4 表明，服务缓存决策和任务卸载决策是二元决策。由于问题 $P1$ 是一个动态规划非凸问题，需首先将其转换为马尔可夫决策过程(Markov Decision Process, MDP)，才能利用提出的基于深度强化学习

的算法来解决上述问题。

1.5　马尔可夫决策过程

一个标准的马尔可夫决策过程可以定义为 $\{S,A,P,R\}$，S 是由环境产生的状态集合。A 是动作空间，P 是状态转移概率，R 是奖励函数。在每个时隙 t，属于不同服务类型的任务被生成，即在时隙 t 生成一个状态 $s_t \in S$，然后根据当前状态选择一个行动 $a_t \in A$。执行 a_t 后，系统进入下一个状态 s_{t+1}，并根据奖励函数获得一个标量奖励 $r \in R$。本文对马尔可夫决策过程的各个元素的详细定义如下。

1.5.1　状态空间

在每个时隙 t 的开始，系统中的每个用户都会产生一个属于某种服务类型的计算任务，传输数据大小或任务的计算需求组合起来形成任务 $Q_j(t)$ 的传输量和计算量。因此，系统的状态空间 s_t 可以定义为式(23)：

$$s_t=\{w(t),z^U(t),z^D(t),R,B^U,B^D,f^E\} \tag{23}$$

其中 $w_j(t)=[w_1(t),\ldots,w_K(t)]$ 代表每个服务的计算需求，$z^U(t)=[z_1^U(t),\ldots,z_K^U(t)]$ 和 $z^D(t)=[z_1^D(t),\ldots,z_K^D(t)]$ 分别代表每个服务的上传数据大小和计算结果。R 是 ES 的总缓存容量，B^U 和 B^D 分别是系统的总上传和下载带宽，f^E 是 ES 的最大计算能力。

1.5.2　动作空间

为了使任务的长期平均延迟最小化，并尽可能地利用 ES 的可利用资源，必须根据每个时隙 t 的不同状态适当地作出行动决策。本文不仅考虑离散的服务缓存和任务卸载决策，还考虑了连续的资源分配决策，因此，行动空间可以定义为式(24)：

$$a_t=\{b^U(t),b^D(t),ao(t),f(t)\} \tag{24}$$

其中 $b^U(t)=[b_1^U(t),\ldots,b_K^U(t)]$ 和 $b^D(t)=[b_1^D(t),\ldots,b_K^D(t)]$ 分别代表在时隙 t 中分配给每个服务的上行和下行带宽决策。$ao(t)$ 代表服务缓存和任务卸载的联合决策。如约束式(15)所述，只有当任务对应的服务环境已经被缓存在 ES 中时，任务才能被卸载到 ES 中执行，因此需要剔除非法的行动决策，即服务不缓存而任务卸载的动作，并根据约束条件式(15)将 $ao(t)$ 离散化，因此，可选行动可以描述为：服务缓存任务卸载、服务缓存任务不卸载、服务不缓存任务不卸载。为了处理问题中包含连续离散混合行动空间的挑战，这里为每个服务输出上述三个行动，然后选择其中最大的一个值作为正式的服务缓存和任务卸载联合决策，即 $ao(t)=[ao_j^1(t),ao_j^2(t),ao_j^3(t)],\forall j\in K$，其中 $ao_j^1(t)$，$ao_j^2(t)$ 和 $ao_j^3(t)$ 分别表示服务缓存任务卸载，服务缓存任务不卸载，以及服务不缓存任务不卸载决策。$f(t)=[f_1(t),\ldots,f_K(t)]$ 代表 ES 在时隙 t 中分配给每个服务的计算资源。

1.5.3　状态转移概率 P

状态转移概率表示系统在 s_t 处采取 a_t 后移动到下一个状态 s_{t+1} 的概率。在本文中，时隙 t 的状态属性是相互独立的，只由环境产生，系统没有关于状态转换的先验知识，因此，状态转移概率 P 可以表示为式(25)：

$$\begin{aligned}&p(s_{t+1}|s_t,a_t)\\&=p(w(t),z^U(t),z^D(t),R,B^U,B^D,f^E|s_t,a_t)\\&=p(w(t),z^U(t),z^D(t)|s_t,a_t)\end{aligned} \tag{25}$$

1.5.4　奖励函数

奖励是环境在 s_t 处执行 a_t 后将回馈给智能体的标量信号，它可以帮助智能体改进策略。由于强化学习的目标是最大化系统的长期平均奖励，而问题 P1 的目标是最小化任务的平均延迟，文章通过最大化任务延迟的负值将奖励函数与优化目标相结合。此外，奖励函数增加辅助奖励项来使算法更容易接近优化目标。奖励函数定义为式(26)：

$$\begin{aligned}&r(t)=-d(t)+\sum_{i\in D}y_i(x)\\&D=\{left_R,left_U,left_D,left_{resource}\}\end{aligned} \tag{26}$$

其中 $d(t)$ 表示所有任务在时隙 t 中的完成延迟，其负值作为奖励函数的主要目标，这样算法就会倾向于选择任务延迟较小的行动。$y_i(x)$ 中的 x 是剩余资源与总资源的比率，作为辅助奖励引导函数，目的是在满足约束条件式(4)(11)(12)和(17)的情况下，尽可能地提高系统资源利用率。$y_i(x)$ 可写为式(27)：

$$y_i(x)=\begin{cases}\dfrac{K_i}{x}, x\ge 0.03\\ \dfrac{K_i}{0.03}, 0\le x<0.03\\ -160, x<0\end{cases} \tag{27}$$

其中 $K_i\in\{K_{left_R},K_{left_U},K_{left_D},K_{left_{resource}}\}$ 是奖励函数的系数。式(27)中的 0.03 代表资源利用率的最小阈值，-160 表示在时隙 t 分配的资源超过可用资源的惩罚。

2　基于 SAC 强化学习算法的解决方案

SAC 算法是一种基于熵最大化的演员评论家算法，它通过策略评估和策略迭代达到收敛状态。SAC 的目标是在最大化熵的同时最大化累积期望报酬，这使得 SAC 算法能够平衡探索-开发的权衡，同时通过使用重新参数化技巧实现更稳定的训练过程。SAC 通过随机抽样存储在经验回放缓冲池中的经验数据来进行训练，这打破了样本之间的联系，提高训练过程的样本效率。在本文 SAC 算法的细节描述如下。

2.1　基于最大化熵的强化学习

与现有的大多数强化学习算法相比，SAC 在最大化累积期望奖励的同时，也最大化了策略的熵，这可以使稳定性和探索性得到显著提高。SAC 中策略的优化目标可以表示为式(28)：

$$\pi^*=\arg\max_{\pi}\mathbb{E}_{(s_t,a_t)\sim p_\pi}\left[\sum_{t=0}^{T}\gamma^t\left(r(s_t,a_t,s_{t+1})+\alpha H(\pi(\cdot s_t))\right)\right] \tag{28}$$

其中 $\gamma\in(0,1)$ 是折现因子，表示未来奖励的重要性，$\alpha\in(0,1)$ 是温度系数，代表熵项和奖励之间的相对重要性。$H(\pi(\cdot s_t))$ 是熵项，由 $-log(a|s)$ 表示，代表策略的随机性程度，熵越大，策略的随机性越高。熵使得算法尽可能地探索环境而不遗漏任何有用的行动。因此，它通过随机探索对环境有更全面的了解，从而实现更强的鲁棒性，并能灵活地适应变化的环境。与传统的强化学习不同，SAC 中的状态值函数 $V^\pi(s)$ 可以描述为

$$V^\pi(s)=\mathbb{E}_{(s_t,a_t)\sim p_\pi}\left[\sum_{t=0}^{T}\gamma^t\left(r(s_t,a_t,s_{t+1})+\alpha H(\pi(\cdot s_t))\right)|\ s_0=s\right] \tag{29}$$

同时，SAC 中的行动状态值函数 $Q^\pi(s,a)$ 可以定义为式

$$\begin{aligned}Q^\pi(s,a)=\mathbb{E}_{(s_t,a_t)\sim p_\pi}[&\sum_{t=0}^{T}\gamma^t\left(r(s_t,a_t,s_{t+1})\right)\\&+\alpha\sum_{t=1}^{T}\gamma^t H(\pi(\cdot s_t))\,|\,s_0=s,a_0=a]\end{aligned} \tag{30}$$

因此，$V^\pi(s)$ 和 $Q^\pi(s,a)$ 之间的关系可以用式(31)表示。

$$V^\pi(s)=\mathbb{E}_{s_t\sim p}[\mathbb{E}_{a_t\sim\pi}[Q^\pi(s_t,a_t)]+\alpha H(\pi(\cdot|\ s_t))] \tag{31}$$

根据贝尔曼期望方程，SAC 中的 $Q^{\pi}(s,a)$ 也可以描述为

$$Q^{\pi}(s,a)=\mathbb{E}_{s_{t+1}\sim p,a_{t+1}\sim\pi}\left[r(s_t,a_t,s_{t+1})+\gamma\left(Q^{\pi}(s_{t+1},a_{t+1})+\alpha H(\pi(\cdot|s_{t+1}))\right)\right]=\mathbb{E}_{s_{t+1}\sim p}\left[r(s_t,a_t,s_{t+1})+\gamma V^{\pi}(s_{t+1})\right] \tag{32}$$

2.2 SAC 深度强化学习算法

在 SAC 算法中，三个带参数的神经网络被用来分别近似计算状态价值函数、状态动作价值函数和策略。重放经验缓冲池和随机梯度下降算法被用来更新神经网络的参数。策略网络用 π 表示，其参数用 θ 表示；状态动作价值网络用 Q 表示，其参数用 w 表示；状态价值网络用 V 表示，其参数用 φ 表示。为了保持训练过程的有利稳定性，在 SAC 算法中利用了一个可自动学习的温度系数 α，它可以自动调整以帮助算法平衡探索和利用之间的权衡。SAC 中状态动作价值网络、策略网络和温度超参数自动调整的细节介绍如下。

2.2.1 状态动作价值网络

$Q(s,a)$ 表示遇到状态 s_t 时，采取行动 a_t 后的预期累积奖励。为了能够估计连续和高维行动空间中的 $Q(s,a)$ 值，SAC 使用由参数 w 表示的神经网络来逼近 $Q(s,a)$，即 $Q(s,a)\approx Q_w(s,a)$。此外，均方误差损失函数被利用来迭代更新网络参数并提高算法的准确性。因此，$Q_w(s,a)$ 的损失函数可用式(33)表示。

$$J_Q(w)=\mathbb{E}_{(s_t,a_t,s_{t+1})\sim\mathcal{D}}\left[\frac{1}{2}\left(Q_w(s_t,a_t)-\hat{Q}(s,a)\right)^2\right] \tag{33}$$

其中 $\hat{Q}(s,a)$ 满足 $\hat{Q}(s,a)=r(s_t,a_t,s_{t+1})+\gamma V_{\bar{\varphi}}(s_{t+1}))$，其中 $V_{\bar{\varphi}}(s_{t+1})$ 是时隙 $t+1$ 的状态值。考虑到式(31)，$V_{\bar{\varphi}}(s_{t+1})$ 可以由得出，即 $V_{\bar{\varphi}}(s_{t+1})=Q_{\bar{w}}(s_{t+1},a_{t+1})-\alpha\log(\pi_\theta(a_{t+1}|s_{t+1}))$，$Q_{\bar{w}}(s_{t+1},a_{t+1})$ 是 $t+1$ 时隙的状态动作值，可以用 Q 的目标网络来近似，目标网络的参数可以写成 $\bar{w}$，用式(34)来更新。

$$\bar{w}=\tau w+(1-\tau)\bar{w} \tag{34}$$

$\mathcal{D}$ 是经验重放缓冲区，负责收集训练数据，为训练提供样本。由于 $\mathcal{D}$ 的存在，SAC 可以利用以前的记录进行训练，提高了样本的效率，打破了样本之间的关联性，提高了训练的稳定性。因此，根据式(33)，Q 的梯度可以表示为 $\nabla_w J_Q(w)$，并表述为式(35)：

$$\nabla_w J_Q(w)=\nabla_w Q_w(s_t,a_t)\left(Q_w(s_t,a_t)-\hat{Q}(s,a)\right) \tag{35}$$

因此，Q 网络在时隙 t 的参数 w 可以通过(36)更新。

$$w_{t+1}=w_t-\lambda\nabla_w J_Q(w) \tag{36}$$

其中 λ 是 Q 的学习率。

2.2.2 策略网络

策略网络是状态到行动的映射。SAC 通过最小化 KL 散度来寻找到一个新的策略 π_{new}，满足 $Q^{\pi_{new}}(s_t,a_t)\geq Q^{\pi_{old}}(s_t,a_t)$。因此，$\pi$ 的损失函数可以定义为式(37)：

$$\begin{aligned}J_\pi(\theta)&=D_{KL}\left(\pi_\theta(.s_t)\,\|\,\exp\left(\frac{1}{\alpha}Q_w(s_t,.)-\log Z(s_t)\right)\right)\\&=\mathbb{E}_{s_t\sim\mathcal{D},a_t\sim\pi_\theta}\left[\log\left(\frac{\pi_\theta(a_ts_t)}{\exp\left(\frac{1}{\alpha}Q_w(s_t,a_t)-\log Z(s_t)\right)}\right)\right]\\&=\mathbb{E}_{s_t\sim\mathcal{D},a_t\sim\pi_\theta}\left[\log\pi_\theta(a_ts_t)-\frac{1}{\alpha}Q_w(s_t,a_t)+\log Z(s_t)\right]\end{aligned} \tag{37}$$

其中 $Z(s_t)$ 是用于归一化分布的函数，它取决于状态，对策略网络的参数梯度没有影响。为了从分布中对 a_t 进行抽样并能够计算 Q 网络的梯度，SAC 使用重新参数化的技巧将 a_t 描述为式(38)：

$$a_t=f_\theta(\varepsilon_t;s_t)=f_\theta^{\mu}(s_t)+\varepsilon_t\odot f_\theta^{\sigma}(s_t) \tag{38}$$

$f_\theta^{\mu}(s_t)$ 和 $f_\theta^{\sigma}(s_t)$ 分别为神经网络输出的均值和方差，ε_t 为从正态分布中采样的噪声。因此，式(37)可以重写为式(39)：

$$J_\pi(\theta)=\mathbb{E}_{s_t\sim\mathcal{D},\varepsilon\sim\mathcal{N}}\left[\alpha\log\pi_\theta(f_\theta(\varepsilon_t;s_t)s_t)-Q_w(s_t,f_\theta(\varepsilon_t;s_t))\right] \tag{39}$$

因此，策略网络 π 在时隙 t 的梯度 $\nabla_\theta J_\pi(\theta)$ 可以描述为式(40)：

$$\begin{aligned}\nabla_\theta J_\pi(\theta)&=\nabla_\theta\alpha\log(\pi_\theta(a_ts_t))\\&+(\nabla_{a_t}\alpha\log(\pi_\theta(a_ts_t))-\nabla_{a_t}Q(s_t,a_t))\nabla_\theta f_\theta(\epsilon_t;s_t)\end{aligned} \tag{40}$$

策略网络 π 的网络参数 θ 可以通过式(41)更新。

$$\theta_{t+1}=\theta_t-\lambda\nabla_\theta J_\pi(\theta) \tag{41}$$

其中 λ 是 π 的学习率。

2.2.3 温度系数超参数自适应调整

SAC 算法对熵的温度系数非常敏感。当智能体处于一个未知的环境中时，增加 α 有利于智能体的探索。相反，当智能体已经熟悉环境时，应该减少 α，以便智能体能够选择最佳策略。为了保持探索和利用之间的平衡，有必要实现温度系数的自动调整，SAC 构建了一个约束优化问题，以满足最小熵约束，同时使预期奖励最大化，表述为式(42)：

$$\begin{aligned}&\max_{\pi_0,\ldots,\pi_T}\mathbb{E}_{\rho_\pi}\left[\sum_{t=0}^{T}r(s_t,a_t)\right]\\&\text{s.t. }\mathbb{E}_{(s_t,a_t)\sim\rho_\pi}\left[-\log\pi_t(a_t\,|\,s_t)\right]\geq\mathcal{H}_0,\forall t\end{aligned} \tag{42}$$

通过使用拉格朗日对偶性，原问题被转换为对偶问题，通过式(43)可以得到 α 在时隙 t 的损失函数 $J(\alpha)$。

$$J(\alpha)=\mathbb{E}_{a_t\sim\pi_t}\left[-\alpha\log\pi_t(a_ts_t)-\alpha\mathcal{H}_0\right] \tag{43}$$

其中 $\mathcal{H}_0$ 是预定义的最小策略熵阈值。α 可以通过获取(43)的梯度来更新，可以表示为式(44)：

$$\alpha_{t+1}=\alpha_t-\lambda\nabla_\alpha J(\alpha) \tag{44}$$

提出的算法 SACSTR 步骤显示在算法 1 中，它的训练框架显示在图 2 中。

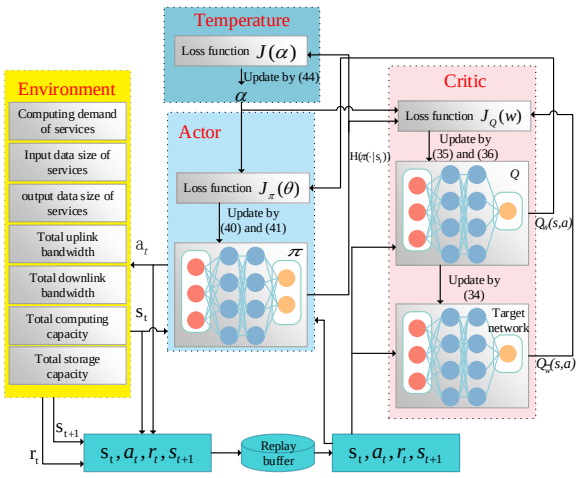

图 2　SACSTR 的训练架构

Fig. 2　Training structure of SACSTR

算法 1　基于 SAC 的服务缓存，任务卸载，资源分配的算法(SACSTR)

输入：$s_t=\{w(t),z^U(t),z^D(t),R,B^U,B^D,f^E\}$

输出：$a_t=\{b^U(t),b^D(t),ao(t),f(t)\}$

a) 分别初始化网络参数 $\theta,w_1,w_2,\overline{w_1},\overline{w_2}$ 和经验回放池 $\mathcal{D}\leftarrow\varnothing$

b) 复制两个策略网络参数 w_1,w_2 给目标网络 $\overline{w_1},\overline{w_2}$

c) for each epoch do

d)　根据状态 s_t 由策略网络选择动作 a_t

e)　执行 a_t，获取下一个状态 s_{t+1} 和奖励 $r(t)$

f)　存储经验 $\mathcal{D}\leftarrow(s_t,a_t,r(t),s_{t+1})$

g)end for

h)for each update step do

i)　　从 $\mathcal{D}$ 中随机抽样一批次经验 $(s_t,a_t,r(t),s_{t+1})\sim\mathcal{D}$

j)　　通过(35)和(36)来更新 Q 网络参数 w_1,w_2

k)　　通过(40)和(41)来更新策略网络参数 θ

l)　　通过(44)来更新温度系数超参数 α

m)　　通过移动平均(34)来更新目标 Q 网络的参数 $\overline{w_1},\overline{w_2}$

n)end for

2.2.4　算法复杂度分析

对于已经训练到达收敛状态的强化学习算法，其通常以一次正向传播的非线性变换的次数作为计算复杂度。因此，提出的算法的计算复杂度可以表示为 $O\left(\sum_{l=1}^{L-1}q_l q_{l+1}\right)$，其中，$L$ 为算法策略网络的神经网络层数，q_l 为第 l 隐藏层的神经元个数。

3　仿真结果与分析

这一节将分析环境中不同参数对算法的影响，并与其他三种基线算法进行比较，以验证提出算法的有效性。

3.1　仿真参数设置

本文假设在 MSs-MEC 和 MCC 结合的边缘计算网络中有一个 ES，它包含 $N=40$ 个移动用户。此外，ES 的存储容量为 $R=300G$，其计算能力为 $31GHz$。系统中考虑了 $K=10$ 种服务，服务的存储空间需求服从 $U[40,80]M$ 的均匀分布，每个时隙时不同服务的被访问概率服从泽夫定律。对于每个用户任务，这里假设输入数据大小和计算结果服从到达率 $\lambda\in[8,12,18,24]$Mb 的泊松分布，每个任务的计算需求是任务输入数据大小的比率，值为 330 $cycles/byte$ [15]。同样，系统中的总上传带宽和下载带宽为 $B^U=B^D=20MHz$，无线信道的上行和下行频谱效率为 $\eta_i^u=\eta_i^d=3bit/s/Hz$。算法的学习率 lr 和折扣系数 γ 分别为 3e-4 和 0.99，其中有 5 个网络，包括 1 个策略网络和 4 个 Q 网络，其中两个网络用于计算 Q 值，用于减少动作值的过度估计[25]，其他网络负责计算目标 Q 值。每个神经网络包含 4 层，包括输入层、输出层和 2 个隐藏层，隐藏层的神经元数量为 256。此外，SACSTR 的奖励函数中的系数 K_{left_R}、K_{left_U}、K_{left_D} 和 $K_{left_{resource}}$ 分别为 7、8、8 和 7，实验中的默认参数见表 1。

表 1　实验参数

Tab. 1　Experimental parameters

参数	参数值
用户个数 N	40
服务类型个数 K	10
ES 总的存储容量 R	300G
ES 总的计算能力 f^E	31GHZ
总的上传与下载带宽 B^U、B^D	20MHz
上传和下载的无线信道频谱效率 η_i^u、η_i^d	3 $bit/s/Hz$
ES 与 CC 间的有线链接传输速率 b^{ec}	$8Mb/s$
CC 的计算速率 f^C	5GHz
每个 epoch 的步数	100
经验回放池大小	10^6
Batch size	128
折扣因子 γ	0.9
软更新系数 λ	0.001
学习率	0.0003
优化器	Adam

3.2　算法收敛性能分析

为了评估超参数对所提算法性能的影响，本文验证了 Adam 优化器的学习率对算法收敛性和稳定性的影响。在实验中，学习率分别被设定为 3e-3、3e-4、3e-5 和 3e-6。从图 3 可以看出，当学习率为 3e-4 时，曲线在 3k 个 epoch 后收敛到最优值，并在达到收敛后保持稳定状态。然而，当学习率为 3e-5 或 3e-6 时，需要花费 6k 或 7k 个 epoch 才能达到收敛状态。此外，当学习率为 3e-3 时，曲线变得非常不稳定，很难收敛到稳定状态。由此可以得出结论，曲线的最优值与学习率的大小不成正比，当算法中的亚当优化器的学习率太小时，算法需要更多的训练回合来达到收敛状态，当学习率过大时，曲线不能收敛到一个较好的值，甚至会导致训练不稳定。因为学习率是神经网络的权重沿梯度方向的更新步骤的大小，当学习率太小时，神经网络需要更多的学习回合来达到损失函数的最小值。相反，学习率越大，意味着权重变化越大，可能会跳过最优值而陷入局部最优情况，或者一路发散，使整个训练过程不稳定。

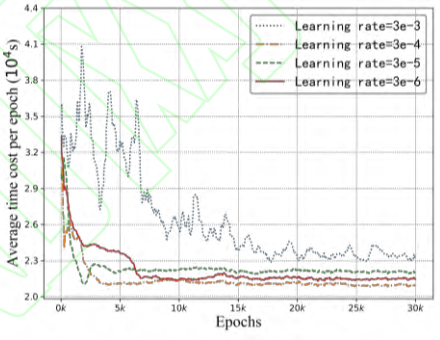

图 3　学习率对收敛性能的影响

Fig. 3　Impact of Learning Rate on Convergence

3.3　对比实验

为了验证提出的算法的优越性和可靠性，本文将提出的算法与其他三种服务缓存、任务卸载和资源分配的基线算法进行比较。

a) 随机方案(Random)。该算法中，系统随机地作出服务缓存、任务卸载和资源分配的决定，前提是算法的资源分配决定必须满足分配总量不超过系统的可分配计算资源。

b) 全云方案(Cloud-Only)。该算法中，所有的任务都被传送到 CC 执行，系统在总带宽的约束下随机分配无线网络带宽资源。

c) 基于 TD3 算法的方案(TD3)。TD3 算法是一种确定性策略的深度强化学习算法，其中 Q 网络目标和 Q 网络都使用两个网络来近似以避免高估 Q 值问题。

本文分别从用户数量、ES 的存储容量、ES 的计算能力、无线网络的频谱效率、CC 的计算速率以及从 ES 到 CC 的传输速率等角度来验证每 epoch 的平均任务延迟。

为了验证不同的用户数对 SACSTR、随机、全云和 TD3 方案的每 epoch 平均延迟的影响，文章设置了用户数以 10 为增量从 20 递增到 40 的实验。图 4 显示，随着用户数量的增加，所有算法的每 epoch 平均延迟都在增加。这是因为任务的处理需要系统中的各种资源，随着用户数量的增加，用户之间对资源的竞争会加剧，在这种情况下，系统分配给用户的资源会减少，从而增加任务的传输和处理延迟。此外，与其他三种基线算法相比，提出的算法 SACSTR 在不同的用户数下实现了最低的任务延时。例如，与随机和全云方案相比，SACSTR 算法的每 epoch 平均延迟平均减少了 72%和 73%，

与同样是基于强化学习算法的 TD3 的方案相比，SACSTR 每轮的平均延迟减少了 42%。主要原因在于对于随机和全云方案，由于必须满足所分配的资源不能超过系统总资源的约束，每次可分配的资源范围相对较小，这导致了较长的任务完成延迟。此外，在不同用户数量的条件下，SACSTR 算法比 TD3 方案更好，这是因为当竞争比较激烈时，基于随机策略的 SACSTR 算法可以尽可能多地探索环境，从而使期望奖励得到显著提高。

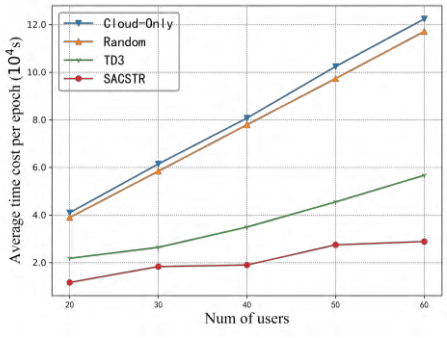

图 4　不同用户数量下每 epoch 的平均时延

Fig. 4　Average time cost per epoch with different numbers of user

图 5 显示，与其他基线算法相比，DOSR 方案在不同的 ES 存储容量下都具有更短的每 epoch 平均延迟，而且 SACSTR、随机和 TD3 方案的每 epoch 平均任务延迟随着 ES 存储容量的增加而减少。然而，全云方案的每 epoch 平均任务延迟不变，因为它将任务卸载到 CC 上执行，与 ES 的存储容量无关。对于 SACSTR 方案，当 ES 的存储容量小于 400GB 时，ES 的缓存空间是影响任务处理延迟的关键因素，因为随着 ES 存储容量的增加，会有更多的任务卸载到 ES。当 ES 的存储容量大于 400GB 时，缓存容量将不再是影响任务处理延迟的最重要因素，即使存储容量增加，任务处理延迟也不会有太大变化。

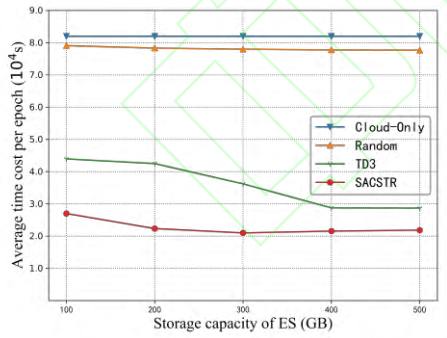

图 5　ES 存储容量对每 epoch 平均时延的影响

Fig. 5　Impact of storage capacity of ES on average time cost per epoch

为了测试了 ES 的计算能力对每 epoch 平均延迟的影响，本文设置 ES 的计算能力从 7 增长到 31MHz，增量为 4MHz。如图 6 显示，ES 的计算能力越大，每 epoch 的平均任务延迟就越低。这是因为较大的计算能力意味着任务将获得更多的计算资源，从而导致任务处理延迟降低。此外，随着 ES 计算能力的增加，随机和全云方案之间的曲线差距变得更大，其原因是随着 ES 计算能力的增加，在 ES 中执行的任务的处理延迟减少，而 ES 计算能力的增加对 CC 中执行的任务的延迟没有影响。同时，SACSTR 方案的曲线变化比 TD3 方案大，这是因为 SACSTR 能在最小化任务时延的同时也追求更高的资源利用率，在面对不同计算能力的 ES 时，可以找到减少任务处理延迟的关键决策。

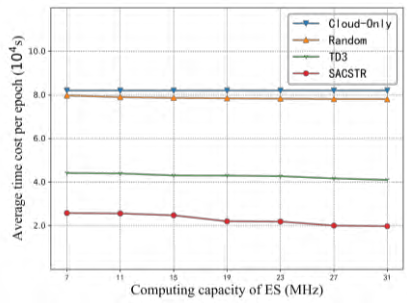

图 6　ES 计算能力对每 epoch 平均时延的影响

Fig. 6　Impact of computing capacity of ES on average time cost per epoch

图 7 验证无线信道的频谱效率从 1.0 增加到 4.0　$b/s/Hz$，增量为 1.0 $b/s/Hz$ 时任务执行时延的变化规律。图中表明，当系统中其他资源不变时，无线网络的频谱效率越高，每 epoch 任务的平均延迟就越小。这是因为无线网络的频谱效率越高，意味着在单位时间内有更多的数据可以传输给 ES，因此任务的传输延迟就越小。从图 7 中可以看出，随着无线网络频谱效率的提高，全云和随机方案的平均任务延迟明显减少。这是因为在这两种算法中分配给任务的带宽资源非常有限，此外，这两种算法的决策不能根据任务的资源需求进行动态调整。相反，图 7 中基于强化学习的方案在不同频谱效率的条件下具有较小的平均任务延迟，而提出的方案由于其较强的随机探索策略，能充分利用有限的无线带宽资源，所以能实现更小的平均任务时延。

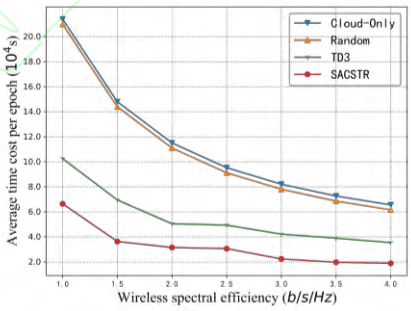

图 7　无线频谱效率对每 epoch 平均时延的影响

Fig. 7　Impact of wireless spectral efficiency on average time cost per epoch

图 8 比较了 CC 的计算速率对每 epoch 平均延迟的影响。由图可知，CC 的计算率越高，每 epoch 的平均任务延迟越短。此外，随着 CC 计算率的提高，每 epoch 平均延迟的变化并不明显，因为当任务被卸载到 CC 时，通信距离是影响任务平均延迟的一个重要因素。提出的算法使用联合的缓存卸载决策同时将无效的动作，能根据任务延迟最小化优先选择把任务卸载到 ES 上。与随机、全云方案和基于 TD3 的方案相比，SACSTR 每 epoch 的任务平均延迟分别减少 71.9%、74%和 49%。

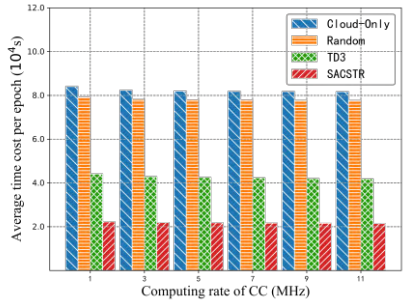

图 8　CC 计算速率对每 epoch 平均时延的影响

Fig. 8　Impact of computing rate of CC on average time cost per epoch

图 9 评估了 ES 和 CC 之间的传输速率对每 epoch 平均延迟的影响。如图所示，ES 和 CC 之间的传输速率越大，每 epoch 的平均任务延迟越短。这是因为传输延迟是任务平均延迟的重要组成部分，较大的传输速率导致 ES 和 CC 之间的传输延迟较小。但是，与 TD3 方案相比，提出的算法的任务平均时延受 ES 到 CC 的传输速率影响小，这是因为提出的算法通过将连续动作输出转为离散动作选择能更恰当地选择缓存和卸载的目的地。与随机、全云和 TD3 方案相比，SACSTR 算法的每 epoch 平均延迟平均分别减少了接 71%、73%和 45%。

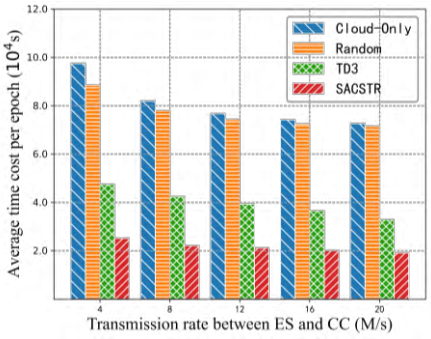

图 9　ES 到 CC 的传输速率对每 epoch 平均时延的影响

Fig. 9　Impact of transmission rate between ES and CC on average time cost per epoch

4　结束语

本文研究了在具有各种服务请求、任务动态生成和有限资源的移动边缘计算系统中的服务缓存、任务卸载和资源分配联合优化问题。首先将该问题建模为 T 时间段内的优化问题。为了使长期平均任务延迟最小化，将优化问题转换为 MDP 问题，然后提出了一个基于 SAC 算法的具有高维度的服务缓存、任务卸载和资源分配方案。该方案可以处理离散和连续的行动空间，在有效改善稳定性的同时，利用多目标奖励函数来提高环境中的资源利用率并加速训练收敛过程。最后，大量的仿真结果显示，与其他三种对比算法相比，提出的算法可以显著降低复杂环境条件下的任务平均延迟。在未来，可将优化问题扩展到边缘节点协作并保证安全来减少任务卸载的延迟或其他费用。

参考文献：

[1] Abbas N, Zhang Yan, Taherkordi A, *et al.* Mobile edge computing: A survey [J]. IEEE Internet of Things Journal, 2017, 5 (1): 450-465.

[2] Shi Shu, Gupta V, Hwuang M, *et al.* Mobile VR on edge cloud: a latency-driven design [C]// Proceedings of the 10th ACM multimedia systems conference. 2019: 222-231.

[3] Zhang Fangzhou. MEC - based latency aware optical character recognition and realtime English translation for smart cities [J]. Internet Technology Letters, 2021, 4 (1): e168. https://doi. org/10. 1002/itl2. 168.

[4] Zhang Jun, Letaief K B. Mobile edge intelligence and computing for the internet of vehicles [J]. Proceedings of the IEEE, 2019, 108 (2): 246-261.

[5] Yang Guisong, Hou Ling, He Xingyu, *et al.* Offloading time optimization via Markov decision process in mobile-edge computing [J]. IEEE Internet of Things Journal, 2020, 8 (4): 2483-2493.

[6] Qiu Xiaoyu, Liu Luobin, Chen Wuhui, *et al.* Online deep reinforcement learning for computation offloading in blockchain-empowered mobile edge computing [J]. IEEE Transactions on Vehicular Technology, 2019, 68 (8): 8050-8062.

[7] Guo Hongzhi, Liu Jiajia. Collaborative computation offloading for multiaccess edge computing over fiber–wireless networks [J]. IEEE Transactions on Vehicular Technology, 2018, 67 (5): 4514-4526.

[8] 方海, 高媛, 赵扬, 等. 卫星边缘计算中任务卸载与资源分配联合优化算法 [J/OL]. 小型微型计算机系统, 1-7 [2022-10-30]. DOI: 10. 20009/j. cnki. 21-1106/TP. 2021-0769. (Fang Hai, Gao Yuan, Zhao Yang, *et al.* Joint Optimization of Task Offloading and Resource Allocation in Satellite Edge Computing [J/OL]. Journal of Chinese Computer Systems, 1-7. [2022-10-30]. DOI: 10. 20009/j. cnki. 21-1106/TP. 2021-0769.)

[9] Chen Ying, Liu Zhiyong, Zhang Yongchao, *et al.* Deep reinforcement learning-based dynamic resource management for mobile edge computing in industrial internet of things [J]. IEEE Transactions on Industrial Informatics, 2020, 17 (7): 4925-4934.

[10] Xu Haitao, Huang Wentao, Zhou Yunhui, *et al.* Edge computing resource allocation for unmanned aerial vehicle assisted mobile network with blockchain applications [J]. IEEE Transactions on Wireless Communications, 2021, 20 (5): 3107-3121.

[11] Feng Jie, Yu F R, Pei Qingqi, *et al.* Cooperative computation offloading and resource allocation for blockchain-enabled mobile-edge computing: A deep reinforcement learning approach [J]. IEEE Internet of Things Journal, 2019, 7 (7): 6214-6228.

[12] 李凡, 王超. 基于移动边缘计算的区块链资源分配算法仿真 [J]. 计算机仿真, 2022, 39 (09): 420-424. (Li Fan, Wang Chao. Simulation of Blockchain Resource Allocation Algorithm Based on Mobile Edge Computing [J]. Computer Simulation, 2022, 39 (09): 420-424.)

[13] Xu Jie, Chen Lixing, Zhou Pan. Joint service caching and task offloading for mobile edge computing in dense networks [C]// IEEE INFOCOM 2018-IEEE Conference on Computer Communications. IEEE, 2018: 207-215.

[14] Zhao Gongming, Xu Hongli, Zhao Yangming, *et al.* Offloading tasks with dependency and service caching in mobile edge computing [J]. IEEE Transactions on Parallel and Distributed Systems, 2021, 32 (11): 2777-2792.

[15] Fan Qingyang, Lin Junyu, Feng Guangsheng, *et al.* Joint service caching and computation offloading to maximize system profits in mobile edge-cloud computing [C]// 2020 16th international conference on mobility, sensing and networking (MSN) . IEEE, 2020: 244-251.

[16] Zhong Shijie, Guo Songtao, Yu Hongyan, *et al.* Cooperative service caching and computation offloading in multi-access edge computing [J]. Computer Networks, 2021, 189: 107916.

[17] Zhang Guanglin, Zhang Shun, Zhang Wenqian, *et al.* Joint service caching, computation offloading and resource allocation in mobile edge computing systems [J]. IEEE Transactions on Wireless Communications, 2021, 20 (8): 5288-5300.

[18] Liu Qian, Shi Long, Sun Linlin, *et al.* Path planning for UAV-mounted mobile edge computing with deep reinforcement learning [J]. IEEE Transactions on Vehicular Technology, 2020, 69 (5): 5723-5728.

[19] Liu Yi, Yu Huimin, Xie Shengli, *et al.* Deep reinforcement learning for offloading and resource allocation in vehicle edge computing and networks [J]. IEEE Transactions on Vehicular Technology, 2019, 68 (11): 11158-11168.

[20] Huang Liang, Bi Suzhi, 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.

[21] Wang Zhongyu, Lv Tiejun, Chang Zheng. Computation offloading and resource allocation based on distributed deep learning and software defined mobile edge computing [J]. Computer Networks, 2022, 205: 108732.

[22] Sellami B, Hakiri A, Yahia S B, *et al.* Energy-aware task scheduling and offloading using deep reinforcement learning in SDN-enabled IoT network [J]. Computer Networks, 2022, 210: 108957.

[23] Qu Bin, Bai Yan, Chu Yul, *et al.* Resource allocation for MEC system with multi-users resource competition based on deep reinforcement learning approach [J]. Computer Networks, 2022: 109181.

[24] Mnih V, Kavukcuoglu K, Silver D, *et al.* Playing atari with deep reinforcement learning [J]. arXiv preprint arXiv: 1312. 5602, 2013.

[25] Fujimoto S, Hoof H, Meger D. Addressing function approximation error in actor-critic methods [C]// International conference on machine learning. PMLR, 2018: 1587-1596.