DOI：10. 13471/j. cnki. acta. snus. ZR20240181

面向DAG任务的分布式智能计算卸载和服务缓存联合优化*

李云[1,2]，南子煜[1]，姚枝秀[1]，夏士超[1,2]，鲜永菊[1]

1. 重庆邮电大学通信与信息工程学院，重庆 400065
2. 重庆邮电大学软件工程学院，重庆 400065

摘　要：建立了一种有向无环图（DAG，directed acyclic graph）任务卸载和资源优化问题，旨在应用最大可容忍时延等约束实现系统能耗最小化。考虑到网络中计算请求高度动态、完整的系统状态信息难以获取等因素，最后使用多智能体深度确定性策略梯度（MADDPG，multi-agent deep deterministic policy gradient）算法来探寻最优的策略。相比于现有的任务卸载算法，MADDPG算法能够降低14.2%至40.8%的系统平均能耗，并且本地缓存命中率提高3.7%至4.1%。

关键词：移动边缘计算；多智能体深度强化学习；计算卸载；资源分配；服务缓存

中图分类号：TN92　　**文献标志码：**A　　**文章编号：**2097－0137（2025）01－0071－12

Joint optimization of distributed intelligent computation offloading and service caching for DAG tasks

LI Yun[1,2], *NAN Ziyu*[1], *YAO Zhixiu*[1], *XIA Shichao*[1,2], *XIAN Yongju*[1]

1. *School of Communication and Information Engineering, Chongqing University of Posts and Telecommunications, Chongqing 400065, China*
2. *School of Software Engineering, Chongqing University of Posts and Telecommunications, Chongqing 400065, China*

Abstract: A directed acyclic graph (DAG) was developed for task offloading and resource optimization, aiming to minimize system energy consumption under constraints such as maximum tolerable delay. Considering that computing requests are highly dynamic in the network and it is difficult to obtain complete system state information, the multi-agent deep deterministic policy gradient (MADDPG) algorithm is used to explore the optimal strategy. Compared to existing task offloading algorithms, the MADDPG algorithm can reduce the average system power consumption by 14.2% to 40.8%, and improve the local cache hit rate by 3.7% to 4.1%.

Key words: mobile edge computing; multi-agent deep reinforcement learning; computation offloading; resource allocation; service caching

MEC实质上是对云计算模型在网络边缘侧的自然延伸和优化（Wang et al.，2020；Khan et al.，2020）。通过将计算任务、数据存储等功能下沉至网络边缘，即在靠近数据源的位置部署边缘服务

* **收稿日期：**2024－05－31　　**录用日期：**2024－08－06　　**网络首发日期：**2024－09－29

基金项目：国家自然科学基金（62071077，62301099）；中国博士后科学基金（2023MD734137）；重庆市自然科学基金（2022NSCQ-LZX0191）

作者简介：李云（1974年生），男；**研究方向：**无线移动通信；E-mail：liyun@cqupt.edu.cn

全文阅读
ZR20240181

器，边缘计算能够显著降低数据传输延迟，提供更快速的服务响应(Kong et al.，2022；Kar et al.，2023)。具体而言，相比于传输数据到达云服务器，边缘计算只需将任务卸载至基站附近的边缘服务器执行，从而大幅缩短任务执行时延(Qiu et al.，2020；Luo et al.，2021)。

随着边缘计算应用场景日益多样化和复杂化，任务计算卸载面临依赖性和服务缓存问题的挑战。实际上，任务之间的依赖性和服务缓存都会影响应用的卸载效率。Lv et al.(2022)针对多基站MEC场景，基于时间消耗表提出了一种新的启发式算法，能够预测卸载决策的影响；Sahni et al.(2021)研究了协同边缘计算中DAG任务的卸载问题，优化任务的平均完成时延，并提出了一种基于DAG任务卸载的启发式算法。Gao et al.(2022)将联合服务缓存和任务卸载问题建模为云边协同的离散优化问题，提出了一种混合启发式算法，将尽可能多的任务卸载往云服务器处理，边缘端用来执行时延敏感的任务。Zhou et al.(2023)研究了MEC中的任务卸载与服务缓存联合优化问题，将原优化问题分为等效的主问题和子问题，提出了一种包含梯度下降分配算法和基于博弈论的算法；Shen et al.(2022)研究了DAG任务卸载和服务缓存问题，并将该问题拆分为资源分配和卸载与缓存决策两个子问题，分别使用低复杂度二分法和启发式算法来求解。Su et al.(2024)在考虑了服务缓存和计算能力约束的条件下，设计了一种基于优先级和路径优先的启发式算法，旨在降低整体的服务延迟。

上述研究多倾向于使用传统的启发式优化方法应对任务卸载和资源分配的挑战，难以在有限的计算资源和时间内收敛至最优解；其次，此类方法常常难以摆脱“维数灾难”的困境(Zhou et al.，2023)。传统的优化策略还存在一个显著缺陷，即它们高度依赖于精确且静态的网络模型描述，这严重限制了模型的灵活性与扩展性，使之难以适应移动边缘计算MEC系统中固有的动态变化特性。深度强化学习(DRL)作为一种前沿的智能优化手段脱颖而出。DRL通过融合深度学习在复杂环境中的高效特征提取能力与强化学习在不确定性条件下的决策优化能力，展现出了强大的潜力，尤其擅长于应对MEC系统环境下高度动态和不确定的资源管理与任务卸载挑战。Su et al.(2024)基于递归神经网络中的序列到序列网络，设计了一种DRL卸载算法，能够保证DAG应用的可靠卸载。Liu et al.(2024)提出了一种图神经网络DRL方案，使用双重深度Q学习网络来提取DAG特征，确保了任务的成功执行，最后任务完成时间得到了显著降低。Guo et al.(2024)在动态的车联网环境下，提出了一种基于DDPG的DAG任务卸载方法，该方法能够很好的适应动态环境的变化，并降低了时延和能耗。

目前，研究工作大都采用集中式DRL算法，但在未来网络架构高度密集、网络中计算请求高度动态以及计算和存储资源部署日益去中心化的边缘计算场景下，采用集中式计算卸载和资源分配算法将会面临完整的系统状态信息难以获取、算法复杂性大幅攀升、计算开销激增及处理效率骤降等问题，尤其是在处理高维度和大规模优化问题时，性能表现欠佳。探索并设计出能适应这种复杂环境的、灵活性强的分布式计算任务卸载与资源管理策略，显得尤为关键与迫切。因此，本文研究了面向DAG任务的分布式智能计算卸载和服务缓存联合优化的问题，旨在任务处理时延约束条件下最小化系统的能耗。

1 系统模型

本文的系统模型如图1所示，该模型包含云服务器、边缘服务器和用户设备。这三部分具有不同的数据处理能力与通信资源。系统中有多个边缘服务器，边缘服务器一般为基站(BS)，其集合定义为$\mathcal{N}=\{1,2,\cdots,n,\cdots,N\}$，$|\mathcal{N}|=N$表示系统中BSs的数量。假设每个BS具有$\rho$个核心，可以并行计算$\rho$个任务(Wang et al.，2019)。在BS n覆盖下的用户设备(UE)的集合定义为

$$\mathcal{M}_n=\{\mathrm{UE}\,1_n,\mathrm{UE}\,2_n,\cdots,\mathrm{UE}\,m_n,\cdots,\mathrm{UE}\,M_n\},$$

其中$|\mathcal{M}_n|=M$表示BS n覆盖下的UEs的数量。

此外，假设MEC系统在离散时隙下运行，定义T为时隙的数量，时隙集合表示为$\mathcal{T}=\{1,2,\cdots,t,\cdots,T\}$。系统中有$K$个类型的应用，集合为$\mathcal{K}=\{1,2,\cdots,k,\cdots,K\}$，每个用户在每个时隙必然会生成一个集合里面的应用。

1.1 服务缓存模型

依赖型应用由若干任务组成，每个任务需要前驱任务计算结果才能执行，本文用DAG表示任务之间的依赖性。应用中的任务只能卸载到缓存了特定服务的服务器执行。定义$\mathcal{V}=$

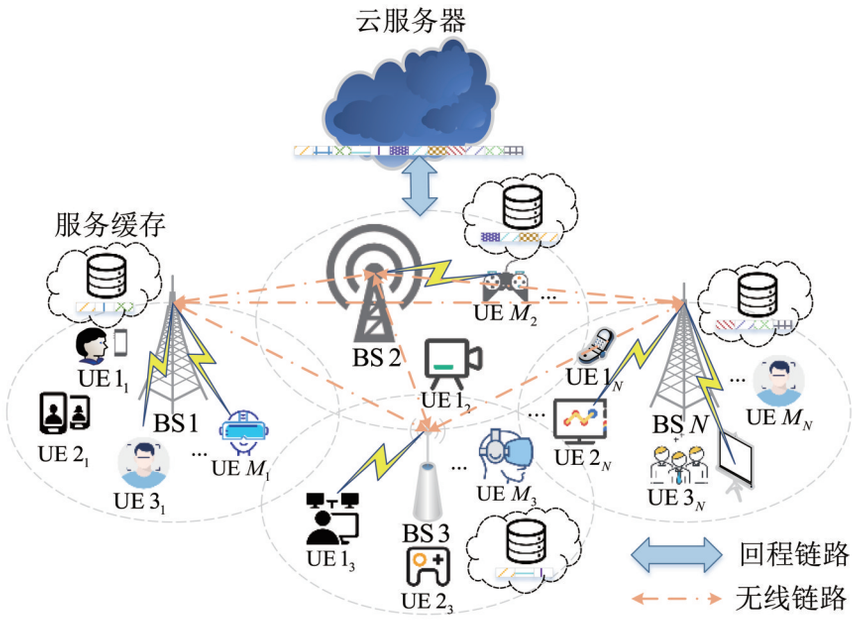

图 1　系统模型

Fig. 1　System model

$\{1, 2, \cdots, v, \cdots, V\}$为网络中的服务类型集合，其中$|\mathcal{V}| = V$表示服务类型的数量。云服务器具有强大的存储与处理能力，可缓存全部服务，但因其与用户设备通信距离较远，导致延迟较大。而边缘服务器虽能进行存储与计算，却受限于存储空间，无法缓存所有服务。如果服务器已缓存了执行任务所需的服务，那么这些任务即可被卸载到该服务器进行处理。

以图 1 为例，假设该应用由 BS 1 覆盖下的 UE m_1产生，系统模型中 BS 1 和 BS 3 缓存了任务τ_1所需的服务（橙色斜线），所以该任务只能卸载到 BS 1 执行。只有 BS 1 缓存了任务τ_2所需服务（蓝色网格线），因此任务τ_2只能卸载到 BS 1 执行。其中，该区域所有边缘服务器均没有缓存任务τ_5所需服务（黑色网格线），因此任务τ_5只能卸载到云服务器执行。

定义c_v表示缓存第v类服务所占用的的缓存空间大小，C_n表示 BS n所能容纳服务的缓存空间大小。$\beta_{n,v}(t) \in \{0, 1\}$作为时隙$t$内 BS n处的第v类服务缓存决策，$\beta_{n,v}(t) = 1$表示第v类服务缓存到 BS n；否则，$\beta_{n,v}(t) = 0$。则 BS n的服务缓存决策为

$$\boldsymbol{\beta}_n(t) = \left\{\beta_{n,1}(t),\ \beta_{n,2}(t),\ \cdots,\ \beta_{n,V}(t)\right\}.$$

由于存储空间有限，$\sum_{v \in \mathcal{V}} \beta_{n,v}(t) c_v \leqslant C_n$表示 BS n缓存的服务大小不能超过 BS n的存储容量。其中，在$t-1$时刻，通过决策$\boldsymbol{\beta}_n(t-1)$缓存的服务并不会立刻产生影响，仅在$t$时刻才可使用（Nath et al.，2020）。

1.2　DAG 应用模型

对于时隙t内，由 UE m_n产生的类型为k的应用 DAG 可以表示为

$$G_{m_n,k}(t) = \left(\mathrm{P}_{m_n,k}(t),\ \mathrm{E}_{m_n,k}(t)\right),$$

其中$\mathrm{P}_{m_n,k}(t) = \{\tau_1(t), \tau_2(t), \cdots, \tau_P(t)\}$为时隙$t$内 UE m_n产生的类型为k的应用。每个节点表示一个任务，并且任务$\tau_p(t)$由 4 个参数表示为

$$\tau_p(t) = \left\{d_p(t),\ v_p(t),\ \varphi_p,\ D_p(t)\right\},$$

其中$d_p(t)$表示任务$\tau_p(t)$数据量的大小，$v_p(t) \in \mathcal{V}$表示执行任务$\tau_p(t)$所需的服务类型，φ_p表示处理 1 bit 任务$\tau_p(t)$所需 CPU 周期数，$D_p(t)$为执行任务$\tau_p(t)$最大可容忍时延，即 DAG 中的相关任务必须在相应的最大可容忍时延范围内完成。

此外，$G_{m_n,k}(t)$的每一条边$\left(\tau_{p-1}(t),\ \tau_p(t)\right) \in \mathrm{E}_{m_n,k}(t)$表示任务$\tau_p(t)$的执行依赖于前一个任务$\tau_{p-1}(t)$的结果。每个任务都只能在其前驱任务全部完成后才能执行。在图 2 中举例说明任务之间的依赖性。例如，任务τ_2、τ_3的执行依赖于任务τ_1执行的结果，而任务τ_4的执行依赖于任务

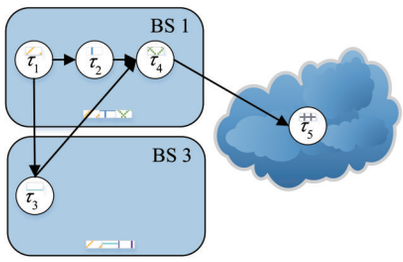

图2 DAG应用实例

Fig. 2 DAG application instance

τ_2、τ_3的执行结果。值得一提的是，任务τ_1是开始任务，任务τ_5是结束任务，任务τ_5完成即进程结束。在实际情况中，还存在一些多结束任务的应用程序，只有当全部结束任务完成时，进程才算结束。由于任务执行所需要的这种依赖数据所占比特数很小，为了简化计算，不考虑依赖数据的传输时延和能耗(Song et al.，2023)。

1.3 应用继承与合并模型

为了提高应用处理效率，本文采用基于应用继承合并的任务管理策略。为了提高应用完成率，允许任务在计算失败后可以被继承到下一个时隙计算。其次，在同一个时隙中，应用之间存在潜在的相似性，这些应用的任务之间可能存在重叠。故允许相同类型的应用可以合并去重后再执行，即去除重复任务节点，相应边上的权重相加，不同的任务成为DAG新的任务节点。图3为基于时隙的应用继承合并的示例图。其中，边上的权重代表所连接的两个任务重复出现的次数。以A、B两种类型的应用为例，来预测最后输出的结果。例如，$t-1$时隙中在时隙结束尚未完成或初次超出可容忍时延的A类应用过渡到时隙t执行，时隙t原有的同类型应用根据任务依赖关系合并成为新的应用(Liu et al.，2023)。那么BS n覆盖下的UEs在时隙t内继承合并后的DAG结果为

$$\mathcal{G}_{\text{total}}(t)=\mathcal{G}_{\text{uncom}}(t-1)+\bigcup_{k,\ m_n} G_{m_n,\ k}(t).$$

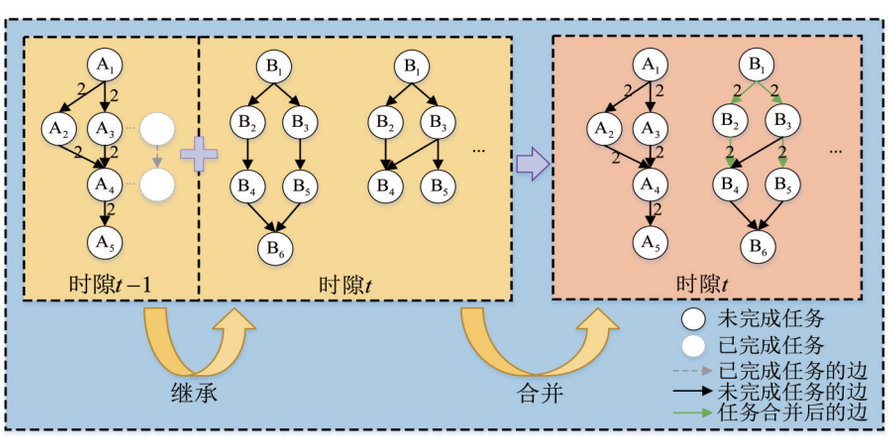

图3 应用继承与合并实例

Fig. 3 Application inheritance and merging instances

1.4 任务计算模型

采用云边协同DAG任务计算模型。由图1可知，BSs与云服务器协同地为UEs提供计算服务。具体而言，每个任务有三种卸载方案：(1) UE m_n的$\tau_p(t)$卸载到本地BS n执行；(2) UE m_n的$\tau_p(t)$卸载到本地BS n后，由BS n转发到相邻BS执行；(3) UE m_n的$\tau_p(t)$卸载到本地BS n后，由BS n转发到云服务器执行。

定义$b_{p,\ m_n}(t)\in\{0\}\cup\mathcal{N}$为UE m_n在时隙t内任务$\tau_p(t)$的卸载决策。其中，$b_{p,\ m_n}(t)=n$，表示任务卸载到本地BS n执行；$b_{p,\ m_n}(t)=n'\left(n'\in\mathcal{M}\{n\}\right)$表示任务卸载到相邻BS n'执行；$b_{p,\ m_n}(t)=0$，表示任务卸载到云服务器进行执行。在时隙t内，UE m_n的任务卸载决策可以表述为

$$\boldsymbol{b}_{m_n}(t)=\left\{b_{1,\ m_n}(t),\ b_{2,\ m_n}(t),\ \cdots,\ b_{P,\ m_n}(t)\right\}.$$

1.4.1 本地基站任务卸载

任务$\tau_p(t)$到BS n的上行数据传输速率$r^{\text{up}}_{m_n,\ n}(t)$为

$$r^{\text{up}}_{m_n,\ n}(t)=B\log_2\left(1+\frac{P_{m_n}(t)h_{m_n,\ n}(t)}{\sigma^2}\right),$$

其中B表示信道带宽，$P_{m_n}(t)$表示BS n覆盖下UE m_n的发射功率，$h_{m_n,\ n}(t)$表示BS n与UE m_n之间的信道增益，σ^2表示噪声功率。

该任务卸载到BS n的传输时间$T^{\text{off}}_{i_n,\ p}(t)$为

$$T^{\text{off}}_{i_n,\,p}(t)=\frac{d_p(t)}{r^{\text{up}}_{m_n,\,n}(t)},$$

则用户的传输能耗 $E^{\text{off}}_{m_n,\,p}(t)=P_{m_n}(t)T^{\text{off}}_{i_n,\,p}(t)$.

如上所述，假定BSs具有ρ个核心，并且可以同时执行ρ个任务。对于任务$\tau_p(t)$上传执行部分会卸载到BS n第i个核心执行，计算时间

$$T^{\text{BS}}_{i_n,\,p}(t)=\frac{d_p(t)\varphi_p}{f_{n_i}},$$

其中f_{n_i}表示BS n第i个核心的CPU频率。

本地基站的计算能耗 $E^{com}_{i_n,\,p}(t)=\kappa_n f^3_{n_i}T^{\text{BS}}_{i_n,\,p}(t)$，$\kappa_n$表示BS n芯片架构相关的有效能量系数。因此，任务$\tau_p(t)$的执行时延和执行能耗为

$$T^{n}_{m_n,\,p}(t)=T^{\text{off}}_{i_n,\,p}(t)+T^{\text{BS}}_{i_n,\,p}(t),$$

$$E^{n}_{m_n,\,p}(t)=E^{\text{off}}_{m_n,\,p}(t)+E^{com}_{i_n,\,p}(t),$$

1.4.2　*邻居基站任务卸载*　如果任务在相邻基站BS n'执行，则需先上传至本地基站，再由本地基站转发至相邻基站。定义BS n与BS n'之间的平均数据传输速率为$\bar{r}_{n,\,n'}$。任务$\tau_p(t)$从BS n卸载到BS n'的传输时间$T^{\text{off}}_{i'_{n'},\,p}(t)$和传输能耗$E^{\text{off}}_{i_{n'},\,p}(t)$为

$$T^{\text{off}}_{i_{n'},\,p}(t)=\frac{d_p(t)}{\bar{r}_{n,\,n'}},\qquad E^{\text{off}}_{i_{n'},\,p}(t)=\bar{p}_{n,\,n'}T^{\text{off}}_{i_{n'},\,p}(t),$$

其中$\bar{p}_{n,\,n'}$为到相邻基站的平均发送功率。该任务在BS n'的第i个核心上的计算时间 $T^{\text{BS}}_{i_{n'},\,p}(t)=\dfrac{d_p(t)\varphi_p}{f_{n'_i}}$，相邻基站的计算能耗 $E^{\text{com}}_{i_{n'},\,p}(t)=\kappa_{m_{n'}}f^3_{n'_i}T^{\text{BS}}_{i_{n'},\,p}(t)$，任务$\tau_p(t)$执行时间和任务在相邻基站的执行能耗为

$$T^{n'}_{m_n,\,p}(t)=T^{\text{off}}_{i_n,\,p}(t)+T^{\text{off}}_{i_{n'},\,p}(t)+T^{\text{BS}}_{i_{n'},\,p}(t),$$

$$E^{n'}_{m_n,\,p}(t)=E^{\text{off}}_{m_n,\,p}(t)+E^{\text{off}}_{i_{n'},\,p}(t)+E^{\text{com}}_{i_{n'},\,p}(t).$$

1.4.3　*云服务器任务卸载*　如果任务在云服务器执行，则需先上传至本地基站，再由本地基站转发至云服务器。定义BS n与云服务器之间的平均数据传输速率为$\bar{r}_{n,\,0}$。任务$\tau_p(t)$从BS n卸载到云服务器的传输时间

$$T^{\text{off}}_{0,\,p}(t)=\frac{d_p(t)}{\bar{r}_{n,\,0}},$$

从BS n卸载到云服务器的传输能耗 $E^{\text{off}}_{0,\,p}(t)=\bar{p}_{n,\,0}T^{\text{off}}_{0,\,p}(t)$，$\bar{p}_{n,\,0}$为BS到云服务器的平均发送功率。任务$\tau_p(t)$在云服务器执行时间和执行能耗为

$$T^{0}_{m_n,\,p}(t)=T^{\text{off}}_{i_n,\,p}(t)+T^{\text{off}}_{0,\,p}(t),$$

$$E^{0}_{m_n,\,p}(t)=E^{\text{off}}_{m_n,\,p}(t)+E^{\text{off}}_{0,\,p}(t).$$

2　问题建模

本文对云边协同DAG任务计算卸载系统能耗进行优化，用户能耗主要为发送任务数据产生的能耗，而基站能耗为向相邻基站转发、向云服务器转发和计算任务数据产生的能耗。对于BS n覆盖下任务$\tau_p(t)$的执行时间

$$T_{m_n,\,p}(t)=\upsilon^{\text{local}}_p(t)T^{n}_{m_n,\,p}(t)+\upsilon^{\text{non-local}}_p(t)T^{n'}_{m_n,\,p}(t)+\upsilon^{\text{cloud}}_p(t)T^{0}_{m_n,\,p}(t),$$

其中$\upsilon^{\text{local}}_p(t)$，$\upsilon^{\text{non-local}}_p(t)$，$\upsilon^{\text{cloud}}_p(t)\in\{0,1\}$是任务$\tau_p(t)$的二元卸载变量。当$b_{p,\,m_n}(t)=n$时，$\upsilon^{\text{local}}_p(t)=1$；否则为0. 当$b_{p,\,m_n}(t)=n'$时，$\upsilon^{\text{non-local}}_p(t)=1$；否则为0. 当$b_{p,\,m_n}(t)=0$时，$\upsilon^{\text{cloud}}_p(t)=1$；否则为0.

执行任务应用的系统能耗$E_{m_n}(t)$为

$$E_{m_n}(t)=\sum_{p=1}^{P}\Big[\upsilon^{\text{local}}_p(t)E^{n}_{m_n,\,p}(t)+\upsilon^{\text{non-local}}_p(t)E^{n'}_{m_n,\,p}(t)+\upsilon^{\text{cloud}}_p(t)E^{0}_{m_n,\,p}(t)\Big],$$

最后，以执行应用的系统能耗为优化目标建立优化模型：

$$\min_{\beta(t),b(t),P(t),t}\ \lim_{|\mathcal{T}|\to\infty}\frac{1}{|\mathcal{T}|}\sum_{t\in\mathcal{T}}\sum_{n\in\mathcal{N}}\sum_{m\in\mathcal{M}_n}E_{m_n}(t),$$

s.t.

$$\beta_{n,\,v}(t)\in\{0,\ 1\},\tag{1-1}$$

$$b_{p,\,m_n}(t)\in\{0\}\cup N,\tag{1-2}$$

$$T_{m_n,\,p}(t)\leqslant D_p(t),\tag{1-3}$$

$$\sum_{v\in\mathbb{V}}\beta_{n,\,v}(t)c_v\leqslant C_n,\tag{1-4}$$

其中BS n覆盖下的所有应用的卸载决策为$\boldsymbol{b}_n(t)=\{\boldsymbol{b}_{1_n}(t),\boldsymbol{b}_{2_n}(t),\cdots,\boldsymbol{b}_{M_n}(t)\}$，则该系统中基站任务卸载策略为$\boldsymbol{b}(t)=\{\boldsymbol{b}_1(t),\boldsymbol{b}_2(t),\cdots,\boldsymbol{b}_N(t)\}$。(1-1)表示BS n对于v类服务只有一种处理方式，(1-2)表示任务$\tau_p(t)$只有一种卸载方式；由于应用计算失败的情况大都源于应用中某个任务的执行时间过长，本文考虑约束力度更强的任务可容忍时延，(1-3)表示任务执行时间不能超出其可容忍时延；(1-4)表示缓存的服务大小不能超过BS n的缓存容量。由于MEC环境的动态多变和完整的系统状态信息难以获取，本文使用一种基于MADDPG的DAG任务卸载和服务缓存算法来解决上述问题。

3 基于 MADDPG 的 DAG 任务卸载和服务缓存算法

首先，进行任务优先级排序；其次，将优化问题转化为N个智能体的部分可观测马尔可夫决策过程（POMDP，partially observable Markov decision process）；最后，使用MADDPG算法来解决问题。

3.1 任务卸载优先级

考虑到由有向边连接的两个任务在DAG中不能逆序执行，本文首先计算待卸载DAG中各个任务的优先级，并按照此优先级顺序做出任务卸载安排。优先级信息根据DAG拓扑信息和任务容忍时延，而卸载决策按照优先级度量的递增顺序来进行。每个任务的优先级取决于每个任务及其后继任务的容忍时延，即

$$\mathrm{rank}_p(t)=\max_{\tau_{p'}(t)\in \mathrm{Suc}\,c_p}\mathrm{rank}_{p'}(t)+D_p(t),$$

其中$\mathrm{Suc}\,c_p$代表$\tau_p(t)$的所有后继任务的集合。若$\tau_P(t)$为应用的结束任务，则

$$\mathrm{rank}_p(t)=D_P(t),$$

$\mathrm{rank}_p(t)$越高，则表示优先级越高。通过对$G_{m_n,\,k}(t)$上的所有任务基于卸载优先级降序排列，得到其卸载任务序列为

$$Q^G(t)=\left(\tau_1'(t),\tau_2'(t),\cdots,\tau_P'(t)\right),$$

其中$|P|$为$G_{m_n,\,k}(t)$中任务的个数。对$G_{m_n,\,k}(t)$中任务的卸载从$Q^G(t)$的第一个元素$\tau_1'(t)$开始，依次进行卸载决策，直到所有任务均被执行完成。值得注意的是，$Q^G(t)$同时是$G_{m_n,\,k}(t)$的一个拓扑排序，此顺序进行任务卸载确保了任务之间原有的依赖关系。

3.2 问题转化

本文将每个BS抽象为一个智能体，由于智能体本身不具备获取全部环境信息的能力，故而需将优化问题转化为POMDP。POMDP交互过程使用$(\mathcal{S},\ \mathcal{O},\ \mathcal{A},\ \mathcal{R})$表示，其中$\mathcal{S}$表示全局状态空间，$\mathcal{O}$表示本地观测集合，$\mathcal{A}$表示动作空间的集合，$\mathcal{R}$表示奖励的集合。在每个时隙$t$的开始，智能体$n$从环境$s(t)\in\mathcal{S}$获得环境观测信息$o(t)\in\mathcal{O}_n$，从而在现有的策略来选择卸载、缓存和资源分配动作$a(t)\in\mathcal{A}_n$。而环境会根据所有智能体的联合动作$\mathcal{A}(t)=\{a_1(t),\ a_2(t),\ \ldots,\ a_N(t)\}$来转移到下一个状态$s(t+1)\in\mathcal{S}$，智能体会收到环境返回的奖励值$r_n(t)\in\mathcal{R}_n$。

（1）环境状态。环境状态$s(t)=\{\phi(t),\ \beta(t-1)\}$包含MEC系统中所有应用的任务信息和缓存状态信息。$\phi(t)$表示应用的任务信息，包括任务的数据量$d_p(t)$、所需服务类型$v_p(t)$、每比特任务所需CPU周期数和可容忍时延$D_P(t)$。$\beta_n(t-1)$表示当前时隙中服务缓存状态信息，根据$t-1$时隙的服务缓存动作得到。

（2）本地观测。每个智能体n在时隙t的本地观测$o_n(t)=\{\phi_n(t),\ \beta_n(t-1)\}$，包括本地UE生成的任务信息和$t-1$时隙中基站服务缓存状态信息。

（3）动作。给定智能体n的本地观测，智能体n根据当前策略采取动作。该动作应描述智能体的决策信息，包括服务缓存决策、任务卸载决策和发射功率。在时隙t内，智能体n的动作可以描述为$a_n(t)=\{b_n(t),\ \beta_n(t),\ P_n(t)\}$。其中，$b_n(t)$为BS n覆盖下的任务的卸载动作，$\beta_n(t)$为BS n的服务缓存动作，$P_n(t)$为BS n覆盖下的UE的发送功率动作。

（4）奖励。优化目标是在满足最大可容忍时延的范围内最小化系统能耗。即智能体n的奖励

$$r_n(t)=\sum_{m_n\in\mathcal{M}_n}\left[Y_{m_n}(t)-E_{m_n}(t)\right],$$

其中$Y_{m_n}(t)=\sum_{p=1}^{\chi}\xi H\left(D_p(t)-T_{m_n,\,p}(t)\right)$表示任务卸载是否满足最大可容忍时延。由于任务之间的依赖性，一个任务计算失败就会导致整个应用计算失败。因此，使用χ表示首个未满足可容忍时延约束任务的前序任务索引。前序任务是指$Q^G(t)$中位于当前任务之前，且与当前任务相邻，需要先被执行的任务。$H(\cdot)$表示阶跃函数，ξ表示权重系数。当所有任务满足可容忍时延要求时，$Y_{m_n}(t)=M\xi$，否则$Y_{m_n}(t)=\chi\xi$。

3.3 MADDPG算法

如图4所示，MADDPG由N个智能体组成，每个智能体能够进行分布执行和集中训练。此外，MADDPG是一种基于actor-critic的算法，actor网络只使用本地观测来分布执行动作；每个智能体n使用所有智能体的状态信息通过集中式学习方式训练actor和critic网络。

在集中训练阶段，智能体n从经验回放池中采样数据，获得环境中所有智能体在该时刻的本地

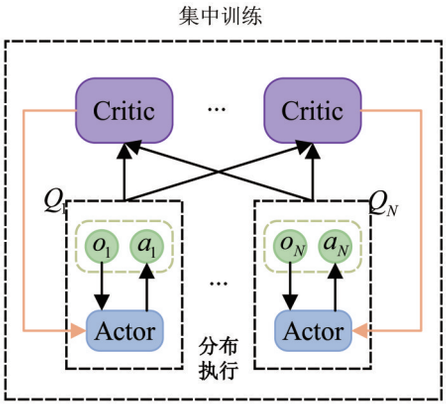

图4　MADDPG算法框架

Fig. 4　The algorithm framework of MADDPG

观测结果$\boldsymbol{o}(t)=\{o_1(t),\ o_2(t),\ \ldots,\ o_N(t)\}$和采取的相关动作$\boldsymbol{a}(t)=\{a_1(t),\ a_2(t),\ \ldots,\ a_N(t)\}$。critic网络会进行训练，输出$Q_n$来评估actor网络的动作（Cui et al.，2023）。

设$\boldsymbol{\pi}=\{\pi_1,\pi_2,\cdots,\pi_N\}$是由$\boldsymbol{\theta}=\{\theta_1,\theta_2,\ldots,\theta_N\}$参数化的连续策略集合。智能体通过从经验回放池随机采样最小批量大小数量的样本U，来最大化actor网络的动作价值函数$Q_n^{\omega_n}(o_n^u,\ \boldsymbol{a}^u)$：

$$\nabla_{\theta_n}\mathcal{J}(\pi_{\theta_n})\approx\frac{1}{U}\sum_{u=1}^{U}\pi_{\theta_n}(o_n^u)\nabla_{a_n}Q_n^{\omega_n}(o_n^u,\boldsymbol{a}^u)\Big|_{a_n=\pi_{\theta_n}(o_n^u)},$$

u为样本序号；重放缓冲区包含$(\boldsymbol{o}(t),\boldsymbol{a}(t),\boldsymbol{r}(t),\boldsymbol{o}'(t))$，$o_n'(t)$表示下一时刻智能体$n$的观测值。

Critic网络通过最小化每个智能体n的损失函数来更新参数ω_n：

$$\mathcal{L}(\omega_n)=\frac{1}{U}\sum_{u=1}^{U}\left(y^u-Q_n^{\omega_n}(o_n^u,\boldsymbol{a}^u)\right),$$

$$y^u=r_n^u+\gamma Q_n^{\omega_n'}(o'^u_n,\boldsymbol{a}'^u)\Big|_{a_n'=\pi'_{\theta_n'}(o'^u_n)},$$

其中$\boldsymbol{a}'$是下一时刻的动作，$\boldsymbol{\pi}'$是下一时刻的目标策略，γ是折扣因子。

分散执行阶段，每个时刻智能体的actor网络根据本地观测$o_n(t)$、当前actor网络状态更新参数θ_n和自身策略π_n做出动作，则相关动作为

$$a_n(t)=\pi_n(o_n(t),\theta_n).$$

为了提高算法的探索能力，在抽样时加入随机噪声N_0后$a_n(t)=\pi_n(o_n(t),\ \theta_n)+N_0$，目标网络的参数通过软更新的方式进行更新，即

$$\theta_n'=\varepsilon_n^a\theta_n+(1-\varepsilon_n^a)\theta_n',$$

$$\omega_n'=\varepsilon_n^c\omega_n+(1-\varepsilon_n^c)\omega_n',$$

其中ε_n^a和ε_n^c分别是actor和critic的软更新率。

4　仿真与分析

选取集中式深度强化学习（CDRL）算法、基于最近最少使用策略的MADDPG算法（MADDPG-LRU）、基于最不经常使用策略的MADDPG算法（MADDPG-LFU）、基于深度确定性策略梯度的算法（DDPG）、基于LRU的深度确定性策略梯度算法（DDPG-LRU）、基于LFU的深度确定性策略梯度算法（DDPG-LFU）共6种对比算法作为参照基准，对MADDPG算法的效能进行验证与评估。

4.1　仿真设置

在本文的云边协同计算场景中，设置有5个BS，每个时隙长度为80 ms。在时隙t开始时刻，假设BS可以获取相邻BS的服务缓存信息；在该场景下，每个BS下均匀覆盖5个用户，假设每个UE每个时隙都能生成一个应用，共有5种应用类型。每个应用共由[3，9]个任务组成，应用中每个任务所需要的服务服从Zipf分布。BS n中任务$\tau_p(t)$请求服务v的概率为

$$\phi_{n,\ v}(t)=\frac{z_{n,\ v}^{-\delta_n}(t)}{\sum_{l\in\mathcal{V}}z_{n,\ l}^{-\delta_n}(t)},$$

其中δ_n表示流行度的偏度参数，$z_{n,\ v}$表示服务v在BS n中的流行度的排名。BS n中的服务流行度可用$\phi_n(t)=\{\phi_{n,\ 1}(t),\ \phi_{n,\ 2}(t),\ \cdots,\ \phi_{n,\ V}(t)\}$来表示，并且BS n覆盖下的每个UE会根据本地BS的$\phi_n(t)$来生成具有特定服务需求的应用（Yao et al.，2022）。本文为了实现基站之间的差异化服务流行度，设置了不同的流行度等级和偏度参数。具体而言，设置5个BS的流行度等级各不相同，并以服务目录为流行度等级随机排序。偏度参数设置为$\delta_1=0.4$，$\delta_2=0.6$，$\delta_3=0.8$，$\delta_4=1.0$，$\delta_5=1.2$。

本文假设每个服务的大小为[15, 25] GB，共有10种服务类型，BS的缓存容量C_n为100 GB。基站的带宽B为30 MHz，基站第i个核心的CPU频率f_{n_i}为2.5 GHz，基站到相邻基站的平均发送功率$\bar{p}_{n,\ n'}$和到云服务器的发送功率$\bar{p}_{n,\ 0}$为1 W，到相邻基站的平均传输速率$\bar{r}_{n,\ n'}$为2 Mbit/s，到云服务器的平均传输速率$\bar{r}_{n,\ 0}$为0.5 Mbit/s；BS覆盖下的UE m_n发送功率p_{m_n}为[5, 33] dBm，无线链路增益

$h_{m_n,n}=10^{-6}$，加性高斯白噪声功率 $\sigma^2=-50$ dBm；任务的数据量 d_p 为 $[10,30]$ kbit，所对应的最大可容忍时延 D_p 为 $[10,30]$ ms，处理任务每 bit 所需 CPU 周期数 φ_p 为 $[600,800]$ cycles/bit。UE m_n 的有效能量系数 $\kappa_{m_n}=1\times10^{-27}$。

actor 网络第一层为全连接层，有 128 个神经单元，激活函数为 ReLU；第二层为全连接层，有 64 个神经单元，激活函数为 ReLU。critic 网络结构与 actor 网络结构相同。其它参数设置如表 1 所示。

表 1 仿真参数设置

Table1 Simulation parameter setting

参数	值
actor 网络学习率	0.000 1
critic 网络学习率	0.001
Replay buffer size	256
Mini-batch size	5×10^5
折扣因子 γ	0.9
软更新率 ε_n^a 和 ε_n^c	0.01

4.2 仿真分析

本节将 MADDPG 算法与 CDRL、MADDPG-LRU、MADDPG-LFU、DDPG、DDPG-LRU、DDPG-LFU 进行仿真比较。其中，平均奖励和缓存命中率为平均每个时隙中系统所获得的平均值。其次，验证了不同算法在不同基站存储空间、不同偏度系数和不同用户数量下的算法性能。

4.2.1 收敛性分析 各个算法的收敛性能如图 5 所示。除 MADDPG、CDRL、DDPG 算法以外的算法均在 20 个训练回合内收敛到最优值，MADDPG、DDPG 算法在 40 个训练回合内趋于稳定，CDRL 算法在 70 个训练回合内趋于稳定。其中，CDRL 算法使用单个智能体对全系统的用户进行集中训练与集中执行，故收敛速度较慢。由于 LRU 与 LFU 同属于页面置换算法，无须进行深度强化学习，故应用 LRU 与 LFU 的 MADDPG 与 DDPG 算法收敛速度较快。其中，MADDPG 算法的奖励收敛于 0.73 左右，CDRL 算法的奖励收敛于 0.75 左右，但 CDRL 算法收敛时间比 MADDPG 算法收敛时间长 30 个训练回合左右，故 MADDPG 算法的综合性能更好。DDPG-LRU 和 DDPG-LFU 算法收敛性最差，奖励收敛于 −6.56 左右；MADDPG-LRU、MADDPG-LFU 算法的奖励仅次于 MADDPG 算法，最终收敛于 0.03 左右；DDPG 算法的奖励收敛于 −5.03 左右。相较于 DDPG 及其衍生算法，MADDPG 及其变种展现出显著的性能优势。根本原因在于 DDPG 算法的训练局限性：仅允许任务卸载到本地基站执行，忽略了与环境中其他相邻基站之间的协同交互。这一局限性使算法在复杂场景下难以做出最优的决策。相反，MADDPG 通过智能体之间的协同交互，使任务不仅能卸载到本地基站执行，还可以卸载到其他基站执行。此特性提高了智能体处理时延敏感型应用的能力。

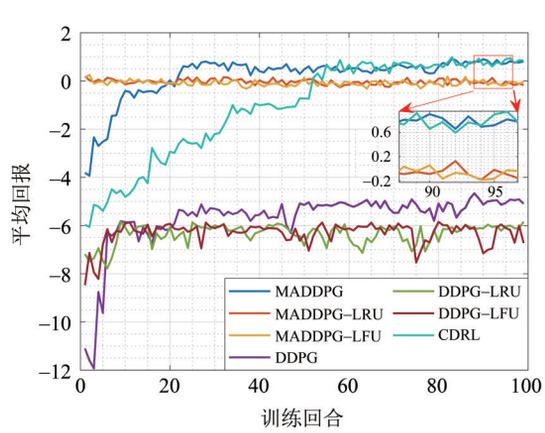

图 5 所用算法的收敛性能

Fig. 5 Convergence performance of the algorithm used

与传统的页面置换算法相比较，DRL 在复杂环境中的信息提取及决策优化表现出显著优势。具体而言，MADDPG 算法体系的能耗相较于其与 LFU 策略结合的变种（MADDPG-LFU）降低了 14.2%，而 DDPG 算法系统能耗对比 DDPG-LFU 算法则下降了 8.5%。从图 6 可以发现：CDRL 算法达到 100% 缓存命中率时，比 MADDPG 算法本地缓存命中率高 0.4%；当 MADDPG 及其变体达到 100% 缓存命中率条件时，分别比 MADDPG-LFU 和 MADDPG-LRU 高出 3.7% 和 4.1%。这是因为传统页面置换算法受限于预设规则，在处理动态变化且复杂多维的内存访问模式时，灵活性和适应性不足。这些算法往往难以在高维度状态空间内有效地识别并利用所有相关信息进行最优决策，导致性能受限。相比之下，DRL 通过深度学习模型，不仅能够有效处理高维输入并提取关键特征，还能够通过强化学习过程动态调整策略，以适应不断变化的环境条件。值得注意的是，现实世界中的系统规模庞大，单个节点难以处理全部的网络请求，并且单节点的故障风险高。因此，集中

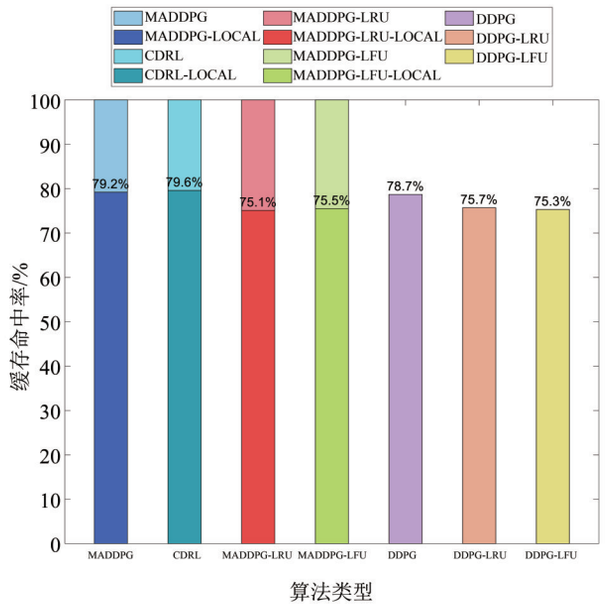

图6　所用任务卸载算法的缓存命中率

Fig. 6　Cache hit rate of the task offloading algorithm used

式算法并不适用于处理网络环境中的计算请求。

4.2.2　性能分析　本节对比了MADDPG、CDRL、MADDPG-LRU、MADDPG-LFU、DDPG、DDPG-LRU和DDPG-LFU七种算法在不同基站缓存容量、不同偏度系数和不同用户数量条件下的性能表现。其中，不同基站缓存容量下的平均缓存命中率、平均能耗和平均应用计算成功率指标如图7-9所示。

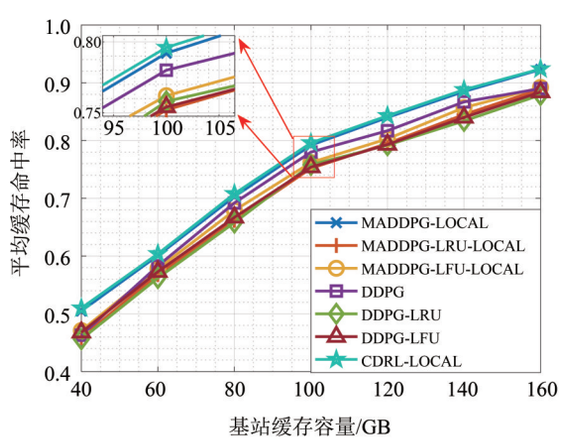

图7　不同基站容量下的平均缓存命中率

Fig. 7　Average cache hit ratio for different base station capacities

随着基站缓存空间的增长，七种算法的平均缓存命中率、平均应用计算成功率均呈现上升趋势，平均能耗呈现下降趋势。具体而言，在达到

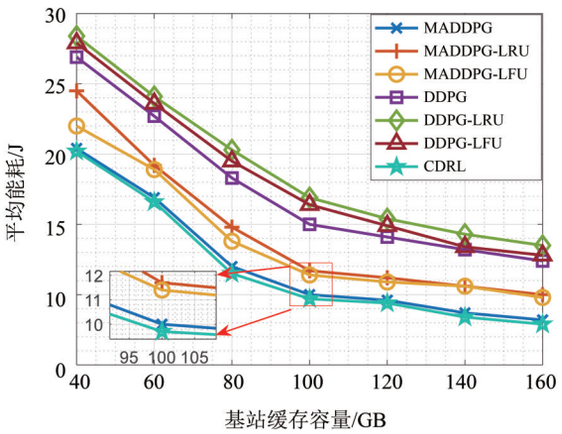

图8　不同基站容量下的平均能耗

Fig.8　Average energy consumption for different base station capacities

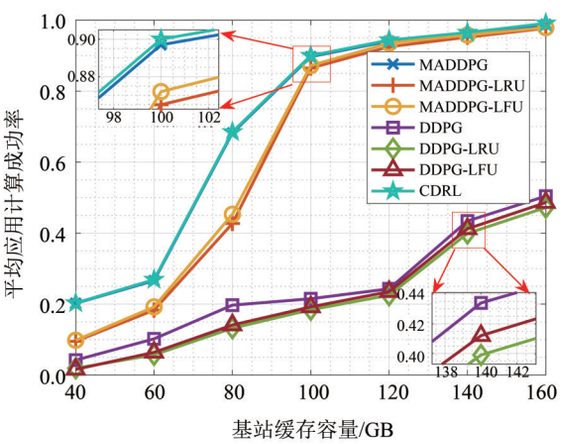

图9　不同基站容量下的平均应用计算成功率

Fig. 9　Average application calculation success rate for different base station capacities

100 GB缓存容量时，MADDPG、CDRL、MADDPG-LRU、MADDPG-LFU算法的缓存命中率均达到了理论最大值100%，体现了在充足缓存资源下，各算法的有效性。CDRL算法的上述指标略高于MADDPG算法，但其收敛周期过长，故综合性能要比MADDPG算法差。其中，除CDRL算法外，MADDPG算法在上述三方面均表现出最优性能，不仅实现了最高的平均缓存命中率和平均应用计算成功率，而且保持了最低的平均能耗。当缓存规模达到100 GB，尽管MADDPG与MADDPG-LRU、MADDPG-LFU在总体缓存命中率上趋同，反映出在缓存资源充裕情况下，基本任务需求都能得到满足，但MADDPG算法在本地缓存命中率上仍保持领先地位，直接减少了对外部基站资源的依赖及相应的能耗开销。因此，即

使在缓存命中率普遍接近饱和的条件下，MADDPG算法通过优化本地缓存利用率，继续维持其相较于衍生算法的性能优势，再次验证了其在高效资源管理和能耗控制方面的卓越能力。

随着偏度系数的增大，用户请求特定服务的概率呈现出递增态势；反之，偏低的偏度系数则意味着服务缓存行为具有更高的随机性。7种算法在不同偏度系数下的性能指标如图10−12所示。将智能体的偏度系数设为一致，具体取值为[0.2，0.4，0.6，0.8，1.0，1.2]。

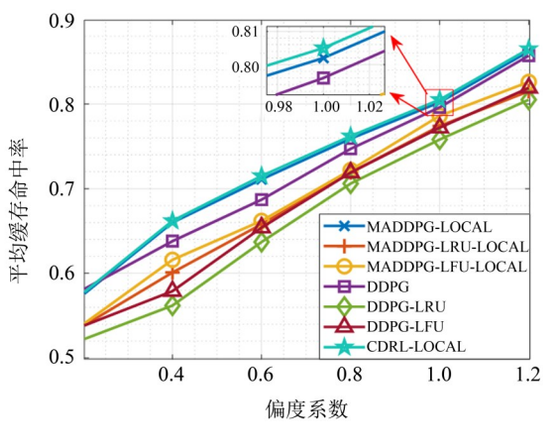

图10 不同偏度系数下的平均缓存命中率

Fig. 10 Average cache hit ratio for different skewness coefficient

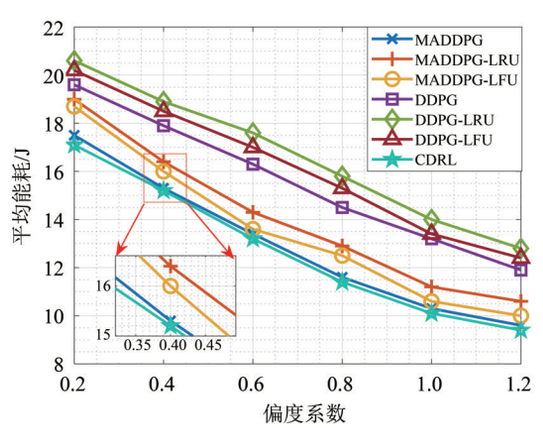

图11 不同偏度系数下的平均能耗

Fig. 11 Average energy consumption for different skewness coefficient

由图10−12可知，随着偏度系数的增长，7种算法的平均缓存命中率、平均应用计算成功率均呈现上升趋势，在平均能耗指标上均呈现下降趋势，这表明算法效率随偏度系数增加得到了优化。其中，CDRL算法与MADDPG算法性能相似，但在不同偏度系数下的收敛速度均慢于MADDPG算法。当偏度系数达到0.8时，MADDPG及其衍生算法的缓存命中率能够达到100%，在平均缓存命中率和平均应用计算成功率上MADDPG算法也展现出超越其他衍生算法的优异表现。更进一步地，在偏度系数达到1.2时，MADDPG算法的本地缓存命中率比DDPG的有1.4%的提升，比MADDPG-LFU的高出3.6%。究其原因，MADDPG算法的独特之处在于其允许所有智能体基于各自的观测值进行联合训练，并允许各智能体间服务缓存状态的差异化信息共享。而DRL能够进行自我学习和持续优化，逐渐逼近最优策略，从而确保了算法性能的持续、稳定提升。

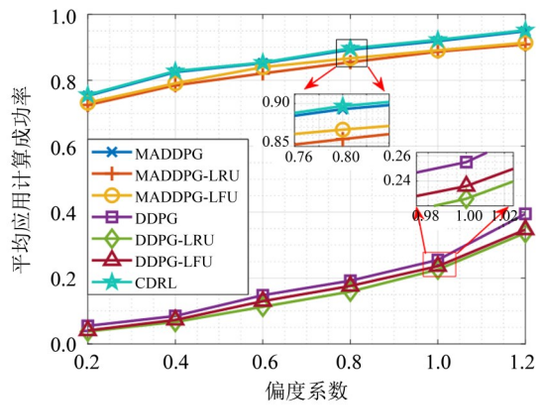

图12 不同偏度系数下的平均应用计算成功率

Fig. 12 Average application calculation success rate for different skewness coefficient

7种算法在不同用户数量下的性能表现如图13−15所示。值得注意的是，尽管用户数量有所变动，但平均缓存命中率与平均应用计算成功率的波动并不显著。原因是在足够的计算资源下，用户数量的增加主要导致系统工作负荷的增长，而非直接影响智能体的动作选择。随着用户数量的增加，系统能耗的增长趋势渐趋平缓。这是由于同一时隙内的应用间存在一定程度的相似性，采用有效的应用合并策略能够有效抑制能耗的线性增长，转而趋向一个稳定水平。

随着用户数量的增加，各算法平均能耗的浮动越来越小，在用户数量达到8个的时候，MADDPG能够稳定在11.6左右；CDRL算法稳定在11.5左右；MADDPG-LRU与MADDPG-LFU分别稳定在13.0与12.7左右。虽然CDRL算法能耗比MADDPG算法高0.1，但其收敛周期最长，综合性能弱于MADDPG算法。相对于除CDRL算法以外的其他算法，MADDPG算法的性能最稳定、平均

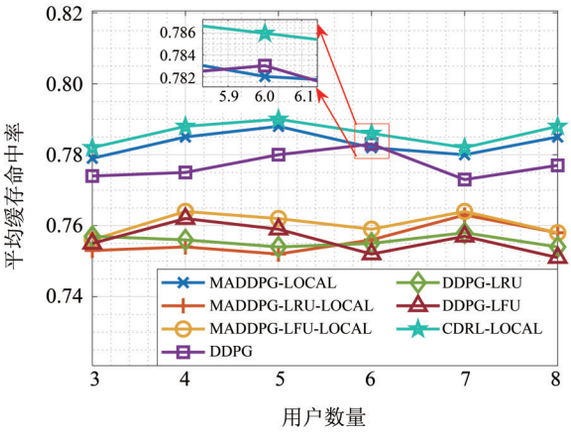

图 13　不同用户数量下的平均缓存命中率

Fig. 13　Average cache hit ratio for different number of users

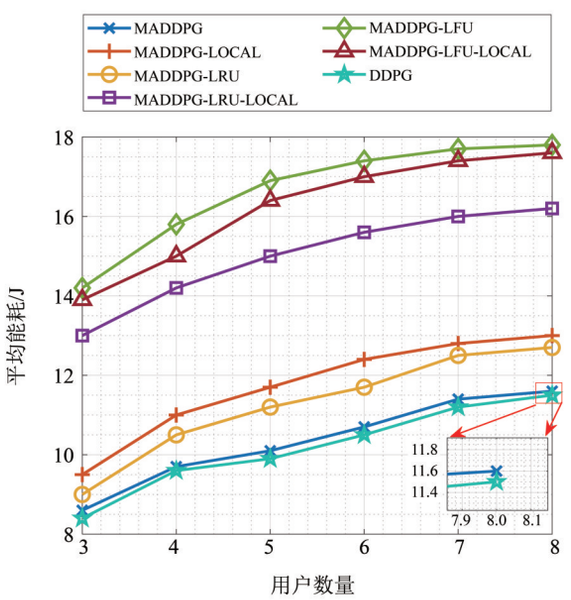

图 14　不同用户数量下的平均能耗

Fig. 14　Average energy consumption for different number of users

能耗也最低。MADDPG算法之所以能够在基站资源充沛的场景下保持优良性能，关键在于其具备对任务协同计算的内在机制。此特性使得智能体在面临卸载决策与缓存决策时，能够充分利用基站资源，制定出更为优化的决策，从而确保系统的高效运行。

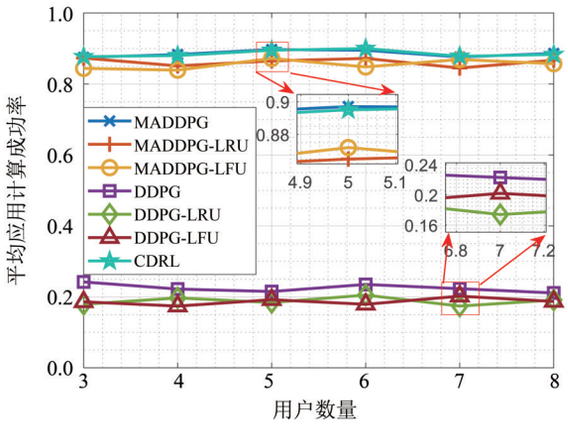

图 15　不同用户数量下的平均应用计算成功率

Fig. 15　Average application calculation success rate for different number of users

5　结　语

本文研究了多基站多用户 MEC 场景下的 DAG 任务计算卸载与资源分配问题，旨在最小化系统能耗。首先，在网络架构高度密集、网络中计算请求高度动态以及计算和存储资源部署日益去中心化的背景下，本文考虑了服务缓存的重要性、任务之间的依赖性和用户资源的局限性，建立了 DAG 任务分布式计算模型；然后，以最小化系统能耗为目标，实现云边协同任务卸载和资源分配优化。考虑到任务的依赖性和相邻时隙之间的相似性，本文改进了应用继承合并策略，提高了应用的计算成功率。最后，使用基于 MADDPG 的 DAG 任务卸载和服务缓存算法进行仿真实验。实验结果从缓存命中率、系统能耗和应用计算成功率三个方面对算法的性能进行了验证。大量实验数据表明，本文所提方案能在有效降低系统能耗的同时，提高缓存命中率与应用计算成功率。

参考文献：

ALE L，KING S A ，ZHANG N，2022. D3PG：dirichlet DDPG for task partitioning and offloading with constrained hybrid action space in mobile-edge computing［J］. IEEE Internet Things J，9（19）：19260-19272.

CUI Q，ZHAO X，NI W，et al，2023. Multi-agent deep reinforcement learning-based interdependent computing for mobile edge computing-assisted robot teams［J］. IEEE Trans Veh Technol，72（5）：6599-6610.

GAO T，TANG Q，LI J，et al，2022. A particle swarm

optimization with lévy flight for service caching and task offloading in edge-cloud computing[J]. IEEE Access, 10: 76636-76647.

GUO Y, MA D, SHE H, et al, 2024.Deep deterministic policy gradient-based intelligent task offloading for vehicular computing with priority experience playback[J]. IEEE Trans Veh Technol, 73(7): 10655-10667.

KAR B, YAHYA W , LIN Y D, et al, 2023.Offloading using traditional optimization and machine learning in federated cloud - edge - fog systems: a survey[J].IEEE Commun Surv Tutor, 25(2): 1199-1226.

KHAN L U, YAQOOB I, TRAN N H, et al, 2020. Edge-computing-enabled smart cities: A comprehensive survey [J]. IEEE Internet Things J, 7(10):10200-10232.

KONG X, WU Y, WANG H, et al, 2022. Edge computing for internet of everything: a survey [J]. IEEE Internet Things J, 9(23): 23472-23485.

LIU S, YU Y, LIAN X, et al, 2023. Dependent task scheduling and offloading for minimizing deadline violation ratio in mobile edge computing networks[J]. IEEE J Sel Areas Commun, 41(2): 538-554.

LIU Z, HUANG L, GAO Z, et al, 2024. GA-DRL: graph neural network-augmented deep deinforcement learning for DAG task scheduling over dynamic vehicular clouds [J]. IEEE Trans Netw Serv Manag, 2024: 1.

LUO Q , HU S , LI C, et al, 2021. Resource scheduling in edge computing: A survey [J]. IEEE Commun Surv Tutor, 23(4): 2131-2165.

LV X, DU H, YE Q, 2022. TBTOA: A DAG-based task offloading scheme for mobile edge computing[C]//IEEE International Conference on Communications. Seoul, Korea:IEEE: 4607-4612.

NATH S, WU J, 2020. Deep reinforcement learning for dynamic computation offloading and resource allocation in cache-assisted mobile edge computing systems [J]. Intell Converged Netw, 1(2): 181-198.

QIU T, CHI J, ZHOU X , et al, 2020. Edge computing in industrial internet of things: Architecture, advances and challenges[J]. IEEE Commun Surv Tutor, 22(4): 2462-2488.

SAHNI Y, CAO J, YANG L, et al, 2021.Multihop offloading of multiple DAG tasks in collaborative edge computing [J]. IEEE Internet Things J, 8(6): 4893-4905.

SHEN Q, HU B J , XIA E, 2022. Dependency-aware task offloading and service caching in vehicular edge computing[J].IEEE Trans Veh Technol, 71(12): 13182-13197.

SONG T, TAN X, REN J, et al, 2023. DRAM: A DRL-based resource allocation scheme for MAR in MEC[J]. Digit Commun Netw, 9(3): 723-733.

SU S, YUAN P, DAI Y, 2024.Reliable computation offloading of DAG applications in internet of vehicles based on deep reinforcement learning[J]. IEEE Trans Veh Technol, 99: 1-13

WANG J, HU J, MIN G, et al, 2019. Georgalas, Computation offloading in multi-access edge computing using a deep sequential model based on reinforcement learning [J]. IEEE Commun Mag, 57(5): 64-69..

WANG X, HAN V, LEUNG V C M, et al, 2020. Convergence of edge computing and deep learning: A comprehensive survey[J].IEEE Commun Surv Tutor, 22(2): 869-904.

YAO Z, LI Y, XIA S, 2022. Attention cooperative task offloading and service caching in edge computing[C]// IEEE Global Communications Conference. Rio de Janeiro, Brazil: IEEE: 5189-5194.

ZHANG Y, FENG B, QUAN W, et al, 2020. Cooperative edge caching: A multi-agent deep learning based approach[J]. IEEE Access, 8: 133212-133224.

ZHOU H, ZHANG Z, LI D, 2023. Joint optimization of computing offloading and service caching in edge computing-based smart grid [J]. IEEE Trans Cloud Comput, 11(2): 1122- 1132.

（责任编辑　王海蓉）