基于DDPG强化学习的多集群算力资源调度算法

胡亚辉[1]，王越嶙[1]，张宸康[1]，洪雨琛[1]，范鹏飞[2]，宋俊平[2]，周旭[2]，鲍震[3]
（1.中国矿业大学（北京）人工智能学院，北京 100086；2.中国科学院计算机网络信息中心，北京 100083；3.国能智深控制技术有限公司，北京 102211）

摘 要：针对多集群算力资源调度平台普遍存在的算法缺失、GPU资源碎片化等挑战，以最大化资源平均利用率和集群负载均衡为目标，将多集群算力资源调度问题建模为马尔可夫决策过程，提出了一种专家经验增强的双池深度确定性策略梯度（EADE-DDPG）算法。在DDPG网络架构中集成了自注意力机制，以动态感知并提取多集群、多维度资源状态的关键信息；设计了结合历史专家经验与实时在线交互的双经验数据缓存池，构建了混合强化学习范式，显著提升了算法的收敛速度、稳定性和样本效率。此外，在动作空间中新增延迟部署动作，能更灵活地应对资源瞬时不足的实际场景。基于阿里巴巴真实数据集的实验结果表明，在多集群GPU算力资源利用率上表现最优，任务部署数量最大提升了14.83%，资源碎片率最高降低了14.50%，GPU负载方差最高降低了89.01%，显著优化了多集群资源调度效率与均衡性。

关键词：多集群；算力资源；资源调度；深度强化学习；深度确定性策略梯度
中图分类号：TN393.0
文献标志码：A
DOI：10.11959/j.issn.1000–436x.2025195

Multi-cluster computing power resource scheduling algorithm based on DDPG reinforcement learning

HU Yahui[1], WANG Yuelin[1], ZHANG Chenkang[1], HONG Yuchen[1], FAN Pengfei[2], SONG Junping[2], ZHOU Xu[2], BAO Zhen[3]
1. School of Artificial Intelligence, China University of Mining and Technology-Beijing, Beijing 100086, China
2. Computer Network information Center, Chinese Academy of Sciences, Beijing 100083, China
3. CHN Energy Zhi Shen Control Technology Co., Ltd., Beijing 102211, China

Abstract: Addressing algorithm deficiencies and severe resource fragmentation, particularly regarding heterogeneous resources like GPU, in multi-cluster scheduling, the problem was modeled as a Markov decision process. An expert augmented dual experience deep deterministic policy gradient (EADE-DDPG) algorithm was proposed. EADE-DDPG integrated a self-attention mechanism for dynamic state extraction and emploied a dual experience buffer (combining expert/offline and real-time/online knowledge) to ensure high sample efficiency and stability. The model also featured an expanded delayed deployment action for handling resource scarcity. Validation using a real-world Alibaba dataset shows EADE-DDPG yields optimal GPU resource utilization, boosting maximum task deployment by 14.83%, cutting resource fragmentation by up to 14.50%, and decreasing cross-cluster GPU load variance by up to 89.01%, thus significantly improving scheduling efficiency and load balancing.

Keywords: multi-cluster, computing power resource, resource scheduling, deep reinforcement learning, DDPG

收稿日期：2025–07–01；修回日期：2025–10–15
通信作者：胡亚辉，huyahui@cumtb.edu.cn
基金项目：国家重点研发计划基金资助项目（No.2024YFB2908700）；高校基本科研业务费专项资金资助项目（No.2025ZKPYZN02）；国家能源集团科技环保有限公司开放课题基金资助项目（No.YZ-2025-101）
Foundation Items: The National Key Research and Development Program of China (No.2024YFB2908700), The Fundamental Research Funds for the Central Universities (No.2025ZKPYZN02), Open Research Topics of CHN Energy Technology & Environment Limited (No.YZ-2025-101)

0 引言

大模型等人工智能技术的不断发展及其在各领域的推广应用，带来算力需求以指数级规律快速增长[1-2]，单台服务器或者单集群已经无法满足大模型训练等特定场合的实际生产要求。伴随着云计算等算力基础设施不断扩充，集群中服务器规模日益庞大，逐渐超出了单集群资源编排调度管理能力的上限。因此，多集群的编排调度管理需求日益迫切。此外，相比于单集群，多集群既可以避免单点故障，又可以利用多云架构进行弹性伸缩，同时还能增强业务隔离。目前，各大厂商也在不断推出、完善自家的多集群调度平台，例如，Kubernetes[3]社区的KubeFed、CNCF开源的项目Liqo、华为云的Karmada以及阿里云的OCM等。

然而，现有的多集群调度平台[4-5]在编排调度方面还存在如下不足：大部分平台考虑算力资源时，仅考虑中央处理器（CPU, central processing unit）和内存之类的硬件算力资源，忽略了图形处理器（GPU, graphics processing unit）资源[6]；多集群调度平台只是预留了编排调度算法的接口，要么调度算法不够完备，要么并未实现具体的调度算法，例如，Karmada多集群调度算法是集群平均分配算法，也就是任务在各集群之间平均分配，仅简单考虑集群剩余资源是否满足需求，不考虑集群之间的负载均衡和利用率等因素；现有多集群调度算法多是基于启发式的方法，在面对系统资源和任务负载的动态变化时，往往缺乏足够的灵活性和适应性，可能需要手动干预或重新配置；现有基于强化学习的方法，通常需要一定的时间进行学习和适应，应对集群高度动态的环境能力不足。

针对上述问题，本文深入研究了多集群算力资源调度问题，在算力资源侧，除了考虑传统的CPU和内存硬件算力资源外，还引入了GPU资源[7]，提出了专家经验增强的专家示教数据深度确定性策略梯度（EADE-DDPG, expert augmented dual experience deep deterministic policy gradient）算法，以提高多集群资源利用率和负载均衡率。其核心创新点在于，在数学建模上对包含GPU在内的多维指标多集群调度数学模型和马尔可夫过程进行了重新建模，在算法设计上不是对现有深度强化算法的简单应用，而是在机制和架构层面进行了根本性重构，以直接解决前文所述的实际部署挑战。本文主要工作如下。

1) 在数学建模上，以最大化GPU等多维度资源利用率为目标，对多集群调度问题进行了重新的数学描述和马尔可夫建模，并新增了延迟部署的动作空间，以符合实际集群资源不足时存在一部分任务处于队列中等待调度的情况。

2) 在算法机制上，EADE-DDPG的独特性在于其多源知识融合策略。传统的深度强化学习（DRL）算法主要依赖于智能体在环境中的在线探索来学习，本文方法则将专家经验作为一种先验知识流，与实时交互产生的在线知识流相结合。这种结合了离线强化学习（来自专家经验）和在线强化学习（来自实时交互）的混合范式，是本文区别于仅依赖于在线探索的传统方法的根本特征。

3) 在网络架构方面，EADE-DDPG引入了自注意力机制，能够动态地感知和权衡多集群、多维度资源状态的相对重要性。这与传统DRL网络通常使用的简单前馈网络架构形成鲜明对比。

4) 基于阿里巴巴真实的多集群数据集进行了实验验证，并与经典的集群调度算法进行了对比分析。实验结果表明，与深度确定性策略梯度（DDPG, deep deterministic policy gradient）[8-9]等基线算法相比，EADE-DDPG算法在多集群GPU算力资源利用率上表现最优，任务部署数量最大提升了14.83%，资源碎片率最高降低了14.50%，跨集群GPU负载方差最高降低了89.01%。

1 相关工作

集群编排调度算法对提升集群资源利用率有着十分重要的作用。如果所有任务都被部署到同一个集群中，会导致该集群资源负载过高，运行效率下降，甚至出现任务排队等问题，同时也造成了其他集群的资源闲置问题。根据算法的技术基础与实现方式，集群编排调度算法可分为两大类：启发式方法和基于强化学习的方法[10]。

1.1 启发式方法

启发式方法根据预先设定的量化指标对不同的调度方案进行打分，并根据得分的高低来选择任务的部署方案。

Santos等[11]提出了一种基于网络感知的容器调度方法，引入了一个可以感知网络状态的微服务编排器，并将其作为Kubernetes默认调度器的补充，

结合应用的网络需求进行资源调配。通过这种方式，能够有效降低容器的网络延迟，实现更高效的任务部署和执行。Menouer等[12]充分考量了云基础设施的实际情况以及用户的多样化需求，提出了Kubernetes容器调度策略（KCSS, Kubernetes container scheduling strategy）方案，通过对集群中CPU、内存等资源状态进行感知，结合用户对于服务的性能要求来选择任务所部署的节点，降低了任务完成时间和功耗。

启发式方法受限于其预先设定的量化指标与固定打分机制，无法适应集群计算资源的动态性以及服务需求的多样性，难以满足复杂环境下的高效调度需求。

1.2　基于强化学习的方法

强化学习能够与环境持续交互，动态感知环境的变化并对动作策略进行调整，因此能够解决传统调度算法存在的适应性不够的问题。

为了使智能体可以根据集群资源状态动态地作出部署策略，Yao等[13]提出了深度集群管理（DeepCM, deep cluster management）算法，使用子图像组合方式表示任务与节点的详细信息，将集群调度问题建模为深度强化学习过程。实验表明，DeepCM算法相较于传统调度算法，显著缩短了任务平均等待时间，提升了调度性能与集群资源整体利用率。

在高性能计算集群领域，Fan等[14]提出了面向集群调度的深度强化学习（DRAS, deep reinforcement learning for cluster scheduling）算法。该算法同样将深度强化学习应用在调度算法中，通过构建包含资源状态、任务特征和调度历史的状态空间，采用近端策略优化（PPO, proximal policy optimization）算法[15]对奖励函数进行了优化。实验结果显示，DRAS在多个性能指标上均优于传统调度方法，性能提升最高可达50%。

双延迟深度确定性（TD3, twin delayed deep deterministic policy gradient）策略梯度算法[16]虽然在一定程度上解决了DDPG的过估计问题，但它主要是针对离散动作空间的问题进行改进，对于连续动作空间问题，其改进效果并不明显。柔性动作评价（SAC, soft actor critic）[17]算法在探索效率上有优势，但其对奖励函数的敏感度较高，而本文问题中奖励函数较为复杂，SAC在训练过程中容易出现不稳定的情况。PPO在策略更新的稳定性上有较好的表现，但对于一些需要频繁交互和快速响应的环境，其更新速度相对较慢，可能无法及时适应环境的变化。

在Kubernetes调度领域，为了解决调度器对微服务依赖和动态负载适应性不足的问题，Jian等[18]提出了深度强化学习增强的Kubernetes调度器（DRS，deep reinforcement learning enhanced Kubernetes scheduler）。通过将调度问题建模为马尔可夫决策过程，设计了包含节点资源状态、任务优先级等要素的状态空间，以及节点选择动作和资源利用率导向的奖励函数，利用深度强化学习对调度策略进行了优化，并部署了资源采集器实时采集CPU、内存资源信息。与原生调度器相比，DRS大大提高了资源利用率，并降低了集群负载不平衡度[19-20]。对于云边协同下Kubernetes系统中动态任务调度问题，Han等[21]提出了一个基于强化学习的调度框架KaiS。该框架使用了分层调度架构，使用图神经网络对集群信息与任务信息进行建模，利用长短期记忆网络来捕捉资源的使用模式，并通过策略梯度算法在线训练调度模型。实验表明，KaiS框架与传统调度器相比，在任务完成时间、资源利用率和服务质量保障方面均有显著提升。

为了解决跨集群下作业调度效率低、资源分配复杂的问题，Huang等[22]基于深度强化学习的Kubernetes调度器（RLSK, reinforcement learning-based scheduler for Kubernetes），通过设计包含集群状态、任务信息的状态空间，结合基于资源利用率和任务完成时间的奖励函数，训练智能体动态优化调度策略。将RLSK调度器与Kubernetes原生调度器进行集成，成功提高了集群的资源利用率，缩短了任务的完成时长。此外，在云边混合资源调度场景，将软演员–评论家（SAC, soft actor-critic）算法[17]进行了改进，以更有效地生成接近最优的任务迁移决策。Huang等[16]采用双延迟深度确定性策略梯度算法解决车联网中边缘计算与云计算协作的计算卸载和资源分配问题，以提高算法的稳定性和可靠性。

综上，目前基于强化学习的集群调度方法在集群动态资源管理中表现十分优异，能够提升资源利用率。但是现有调度方法普遍仅对CPU、内存等同构资源进行优化调度，未对GPU等异构资源进行

调度优化设计，无法有效支撑神经网络等AI类应用。此外，现有基于强化学习的调度方法主要包括DDPG、TD3、SAC和PPO等，不同方法具有不同的优缺点和适用场景。DDPG结构简单易于构建、计算复杂度低，但稳定性和探索性不如其他算法；TD3虽然在一定程度上解决了DDPG的过估计问题，但它主要是针对离散动作空间的问题进行改进，对于本文的连续动作空间问题，其改进效果并不明显；SAC在探索效率上有优势，但其对奖励函数的敏感度较高，而本文的问题中奖励函数较为复杂，且计算复杂度也偏高；PPO在策略更新的稳定性上有较好的表现，但对于一些需要频繁交互和快速响应的环境，其更新速度相对较慢，可能无法及时适应环境的变化。综上，本文采用计算复杂度较低的DDPG网络，同时通过增加注意力机制改进网络结构提升策略网络和价值网络的性能，并增加专家示教缓存池提升样本效率，最终提升强化学习算法的稳定性和准确性。

2 场景描述及系统建模

多集群场景由多个Kubernetes集群组合而成，每个集群都拥有不同大小的算力资源，其中最为重要的算力资源是CPU、内存以及GPU。随着GPU单卡算力的不断增长，在一整张卡上运行小型计算任务会造成资源的浪费。为了解决这一问题，通常会将一张物理GPU虚拟化为多张虚拟GPU卡。值得注意的是，虚拟GPU卡不能跨卡使用，也就是两个以上有剩余GPU资源的卡，其所能分配的总资源数量并不等于它们所有剩余GPU资源综合。此外，本文同样对GPU进行了虚拟化，并且将GPU进一步细分为了GPU内存与GPU核两种算力资源。

多集群算力资源场景如图1所示。该场景由N个集群组成，第n个集群配备$P^{(n)}$台物理主机，第p（$p\in\left[1,P^{(n)}\right]$）台物理主机上配置了$M_{n,p}$张GPU卡。该多集群算力资源MC可用数学公式描述为

$$\mathrm{MC}=\{C_1,C_2,\cdots,C_n,\cdots C_N\},n\in[1,N] \tag{1}$$

$$C_n=\{\mathrm{MACH}_{n,1},\mathrm{MACH}_{n,2},\cdots,\mathrm{MACH}_{n,P^{(n)}}\} \tag{2}$$

$$\mathrm{MACH}_{n,p}=\{\mathrm{CPU}_{n,p},\mathrm{MEM}_{n,p},\mathrm{GPU}_{n,p}\},p\in\left[1,P^{(n)}\right] \tag{3}$$

$$\mathrm{GPU}_{n,p}=\{\mathrm{GPU}_{n,p,1},\cdots,\mathrm{GPU}_{n,p,m}\cdots,\mathrm{GPU}_{n,p,M}\},m\in\left[1,M_{n,p}\right] \tag{4}$$

$$\mathrm{GPU}_{n,p,m}=\{\mathrm{GPU}_{n,p,m}^{\mathrm{Mem}},\mathrm{GPU}_{n,p,m}^{\mathrm{Core}}\} \tag{5}$$

其中，C_n为集群n所有资源的集合；$\mathrm{MACH}_{n,p}$为第n个集群的第p（$p\in\left[1,P^{(n)}\right]$）台物理主机所有资源集合，$\mathrm{CPU}_{n,p}$与$\mathrm{MEM}_{n,p}$为该机器所拥有CPU的数量与内存的大小；$\mathrm{GPU}_{n,p}$为该机器中GPU资源的集合，$\mathrm{GPU}_{n,p,m}$为该机器上第$m$张GPU卡资源的集合，$\mathrm{GPU}_{n,p,m}^{\mathrm{Mem}}$与$\mathrm{GPU}_{n,p,m}^{\mathrm{Core}}$分别代表该卡显存与计算核心资源的配置数量。

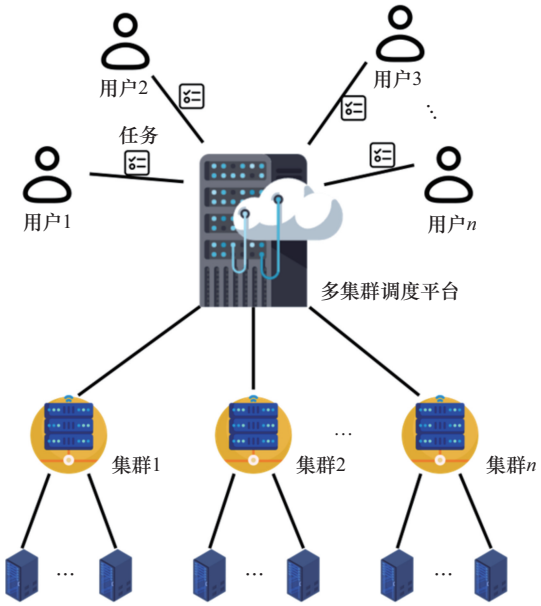


图1 多集群算力资源场景

多集群环境下的任务需求模型同样由CPU、内存以及GPU这3种算力需求资源构成，并且GPU需求同样细分为GPU内存与GPU核两种资源需求。因此，任务需求可以表示为

$$\mathrm{Task}=\{T_1,T_2,\cdots,T_k,\cdots,T_K\},k\in[1,K] \tag{6}$$

$$T_k=\{\mathrm{CPU}_k,\mathrm{MEM}_k,\mathrm{GPU}_k^{\mathrm{Mem}},\mathrm{GPU}_k^{\mathrm{Mem}}\} \tag{7}$$

其中，Task为任务集合，T_k为第k个任务所申请的资源集合，表示该任务对CPU、内存、GPU内存和GPU核资源的具体需求量。值得注意的是，为了简化数学描述，本文假设任务是实例化以后的任务，也就是每个任务只能在一台物理主机上运行。这种假设也是合理的，对于大模型等需要跨多个物理主机的应用，在调度之前已经进行了任务的实例

化过程，因此对于调度器而言，只需考虑如何满足实例的算力资源需求。

在集群环境下，每种资源的状态一般用两个指标去衡量：总资源和已分配资源。可分配资源为集群中可供工作节点使用的资源量，已分配资源为集群中已分配给任务进行使用的资源量。因此，对于集群n，多维度资源平均利用率$\overline{U}_n$的定义式为

$$\overline{U}_n = \frac{1}{|R|}\sum_{r\in R}\frac{r_o^{(n)}}{r_t^{(n)}} \tag{8}$$

其中，$R=\{\text{CPU},\text{MEM},\text{GPU}^{\text{mem}},\text{GPU}^{\text{core}}\}$为集群资源类型；$r_t^{(n)}$为集群某类资源的总量，$r_o^{(n)}$为该类资源已分配量。

本文的优化目标为最大化多维度资源平均利用率，在提高多个集群整体利用率的基础上，平衡每个集群之间的平均利用率，并且最小化集群内部不同资源类型之间的利用率差异，避免因单个资源缺少造成的资源瓶颈。因此多集群算力资源调度问题可描述为

$$\max\frac{1}{N}\sum_{n=1}^{N}\overline{U}_n \tag{9}$$

$$\text{C1}:\theta_{n,p,k}=\{0,1\},\forall n\in[1,N],p\in\left[1,P^{(n)}\right],k\in[1,K] \tag{10}$$

$$\text{C2}:\sum_{n=1}^{N}\sum_{p=1}^{P^{(n)}}\theta_{n,p,k}\leqslant 1,\forall k\in[1,K] \tag{11}$$

$$\text{C3}:\sum_{k=1}^{K}t_{k,r}\theta_{n,p,k}<r_t^{(n,p)},\forall r\in R,n\in[1,N],p\in\left[1,P^{(n)}\right] \tag{12}$$

其中，$\theta_{n,p,k}$为第k个任务是否在集群n物理主机p上进行部署；$t_{k,r}$为第k个任务所需资源r的大小；$r_t^{(n,p)}$为集群n物理主机p上可分配的总资源。约束条件C1表示$\theta_{n,p,k}$的取值只能为0或者1，若部署，则值为1；否则值为0。在约束条件C2中，当和小于1时，表示对于第k个任务将进入队列，以等待空闲资源进行调度；当和等于1时，表示任务只能部署到一个集群的一台主机中。约束条件C3表示部署到集群n物理主机p上所有任务所占用的资源总和不能超过该类资源的可分配总量。

3　EADE-DDPG调度算法设计

很显然式(9)所示问题是个NP完全问题，因此针对该问题，本文设计了EADE-DDPG算法，融合了自注意力机制与双经验池采样策略，对集群中CPU、内存以及GPU资源进行编排调度，解决多集群场景下对GPU资源调度缺失的问题。

3.1　马尔可夫决策过程建模

EADE-DDPG是基于DDPG改进的深度强化学习算法，因此本文将多集群算力资源调度问题建模为马尔可夫决策过程，主要包含状态空间、动作空间和奖励函数3个部分。

1) 状态空间

状态空间用于描述当前时刻的环境信息，系统根据当前状态空间，能够准确地了解当前的任务需求和集群资源状况，从而做出合理的调度决策。在实际的多集群调度场景中，对于任务的部署集群的选择是一个复杂的决策过程，需要综合考虑任务需求资源、所有集群资源余量以及集群资源可分配资源等多方面的影响。因此状态信息不仅要包含当前任务的需求资源，还应包括其他集群的资源余量，以便算法能够全面了解系统的资源状况，做出更合理的调度决策。因此状态空间可以定义为

$$S=\{R_a,\text{Task}\} \tag{13}$$

其中，$R_a=\left\{r_a^{(n,p)}\middle|\, n\in[1,N],p\in P^{(n)}\right\}$表示此时$N$个集群剩余计算资源的集合，$r_a^{(n,p)}=r_t^{(n,p)}-r_0^{(n,p)}$；Task为任务集合。

2) 动作空间

动作空间描述的是模型根据当前状态所做出决策动作的范围。在本文中，动作代表了需要将任务调度至的目标集群和主机。但在实际生产过程中，不仅要考虑将每个任务分配到合适的集群，还需要考虑系统当前算力资源可能不足，处于不适合部署某个任务的状态。因此，设置了一个等待状态来表示该任务还需在队列中延迟，等待资源空闲再部署。本文的动作空间A定义为

$$A=\{\theta_{n,p,k}|\ \ n\in[1,N],p\in\left[1,P^{(n)}\right],k\in[1,K]\} \tag{14}$$

这种设计使算法在面对复杂的资源状况和任务需求时，能够根据实际情况灵活选择是立即部署任务还是暂时等待，以达到更好的调度效果。

3) 奖励函数

根据式(9)所描述的优化目标，当任务部署到集群n时，正奖励定义为该集群的平均利用率。

同时，考虑到集群选择时可能出现的错误动作，例如，所选集群资源余量不满足限制条件导致无法进行任务部署，或者存在可部署集群但选择不进行部署，这种情况会对整个调度系统的性能产生负面影响。因此，为了引导模型避免此类错误，必须给予负奖励，以引导模型在训练过程中更加谨慎地选择动作，提高调度决策的准确性和有效性。

为了使正负奖励衡量指标一致，在定义奖励时，基于式(9)定义的优化目标，对奖励函数进行变形，以分值作为统一度量标准。部署成功时，奖励函数 Re 可以定义为

$$\text{Socre}_n = \overline{U}_n \tag{15}$$

$$\text{Score} = \frac{\text{Score}_n - \text{minscore}}{\text{maxscore} - \text{minscore}} \tag{16}$$

$$\text{Re} = (\text{Score} - 1) \times \tau \tag{17}$$

其中，Score_n 为集群 n 的得分，$\overline{U}_n$ 为集群 n 的资源利用率，minscore 为所有集群中得分最低的集群，maxscore 为得分最高的集群。在式(16)中，将得分进行归一化，确保后续分析和比较的一致性。在式(17)中，将得分减去 1 以后乘以一个常数 τ，本文中将 τ 设置为 10。当部署失败，将奖励函数 Re_t 设置为−50。

3.2　具体算法设计

EADE-DDPG 算法采用策略－价值（Actor-Critic）的神经网络结构，如图 2 所示。在 DDPG 网络架构的基础上，EADE-DDPG 将 Actor 网络的输入层修改为注意力层，保留了状态空间的维度，从而更有效地学习到对任务所需资源的影响因素。Critic 网络同样在输入层的后面也添加了注意力层，以更好地对所选择动作进行评分。

EADE-DDPG 的神经网络也由主网络和目标网络构成，主网络和目标网络又由 Actor 网络和 Critic 网络构成。主网络基于当前状态 $\boldsymbol{s}_t$ 产生具体的调度策略，由两个深度神经网络组成，主 Actor 网络 $\pi\left(\boldsymbol{s}_t|\boldsymbol{\theta}_\pi\right)$ 和主 Critic 网络 $Q\left(\boldsymbol{s}_t,\boldsymbol{a}_t|\boldsymbol{\theta}_Q\right)$，其中 $\boldsymbol{s}_t$ 和 $\boldsymbol{a}_t$ 分别表示 t 时刻的状态和动作，$\boldsymbol{\theta}_\pi$ 和 $\boldsymbol{\theta}_Q$ 分别表示主 Actor 网络和主 Critic 网络的网络参数。目标网络的结构与主网络相同，但参数不同，因此目标 Actor 网络和 Critic 网络分别用 $\pi'\left(\boldsymbol{s}_t|\boldsymbol{\theta}_\pi^T\right)$ 和 $Q'\left(\boldsymbol{s}_t,\pi'\left(\boldsymbol{s}_t|\boldsymbol{\theta}_\pi^T\right)|\boldsymbol{\theta}_Q^T\right)$ 来表示，$\boldsymbol{\theta}_\pi^T$ 和 $\boldsymbol{\theta}_Q^T$ 分别表示目标 Actor 网络和目标 Critic 网络的网络参数。

在经验缓存池方面，EADE-DDPG 算法将单个经验缓存池改为历史和经验两个经验缓存池。在模型开始训练前，导入历史部署的专家数据。专家数据是由专家根据目前集群的剩余资源情况和任务对资源的需求情况，通过式(15)~式(17)预估将任务部署到不同集群的得分，然后选择一个得分最高的集群进行部署，并将此次部署的数据作为专家示教数据，也就是历史数据。对于模型正常训练的数据，则存储至实时经验缓存池。当实时经验缓存池的样

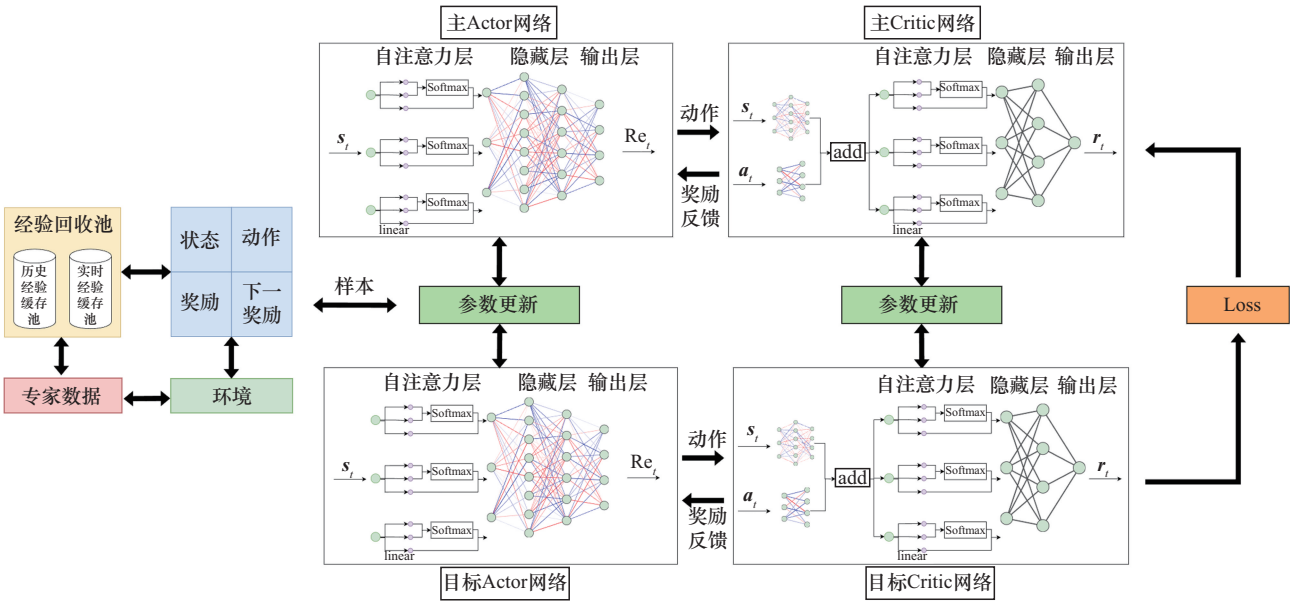


图 2　EADE-DDPG 算法架构

本数量达到最小采样大小时，模型会根据采样策略从历史经验缓存池与实时经验缓存池进行采样与学习。值得注意的是，历史经验缓存池中存储的专家经验数据能够加速模型的学习进度，但同时也可能会导致其获得的奖励值与基于模型推理得到的奖励值存在一定的偏差，最终导致模型在训练阶段出现振荡的现象，因此在每次进行模型训练时，不能仅仅依赖专家经验数据（历史经验缓存池中的数据），而是从模型得到的实时经验缓存池中也取出一部分数据，两部分数据合并得到最终的训练数据。这种采样方式既能充分利用历史经验数据加快收敛，又能利用实时经验数据维持稳定。经验缓存池中存储的经验元组包括当前资源及任务需求状态$\boldsymbol{s}_t$、部署动作决策$\boldsymbol{a}_t$、部署奖励Re_t以及下一状态$\boldsymbol{s}_{t+1}$。

接下来，本文从Actor网络和Critic网络训练及参数更新、目标网络参数更新过程来介绍EADE-DDPG算法。

1) 主Actor网络训练及参数更新

主Actor网络的输入是整合了集群和任务资源信息的一维矩阵，其输出表示系统根据当前状态所做出的任务部署决策，即明确每个任务应分配到的集群或者进行等待。主Actor网络通过学习状态与动作之间的映射关系，能够根据不同的资源状态动态地选择最优的调度动作。

主Actor网络的输出由当前状态空间作为输入来决定，并在主Actor网络输出之后添加了随机噪声，以增大动作空间的探索性。动作输出可描述为

$$\boldsymbol{a}_t = \pi\left(\boldsymbol{s}_t \middle| \boldsymbol{\theta}_\pi\right) + \boldsymbol{n}_0 \tag{18}$$

其中，$\boldsymbol{n}_0$表示随机噪声。主Actor网络采用策略梯度更新网络参数$\boldsymbol{\theta}_\pi$。具体而言，在每步训练迭代中，从两个经验缓存池中按照一定比例批量随机抽取一定数目的经验元组，然后更新网络参数$\boldsymbol{\theta}_\pi$[23]

$$\boldsymbol{\theta}_\pi = \boldsymbol{\theta}_\pi - \frac{1}{N}\sum_n \tilde{N}_a Q(\boldsymbol{s},\boldsymbol{a}|\boldsymbol{\theta}_Q)|_{\boldsymbol{s}=\boldsymbol{s}_n,\boldsymbol{a}=\pi\left(\boldsymbol{s}_n|\boldsymbol{\theta}_\pi\right)} \tilde{N}_{\theta_\pi}\pi\left(\boldsymbol{s}\middle|\boldsymbol{\theta}_\pi\right)|_{\boldsymbol{s}_n} \tag{19}$$

其中，N表示从缓存池中取出的样本数量。

2)主Critic网络训练及参数更新

主Critic网络的输入分为两部分：当前时刻的状态空间和对应的动作空间。它接收主Actor网络输出的动作以及当前的状态信息，对该动作在当前状态下的价值进行评估；输出则是当前动作对于整个多集群调度系统的回报奖励值。主Critic网络通过学习状态–动作对与奖励值之间的关系，能够为Actor网络提供反馈，指导其优化调度策略。

DDPG算法在模型网络层上，Actor网络与Critic网络的网络层由一个输入层、一个自注意力层、一个输出层和两个隐藏层组成。主Critic网络的参数$\boldsymbol{\theta}_Q$通过最小化损失函数进行更新。损失函数定义为

$$L_s(\boldsymbol{\theta}_Q) = \mathbb{E}\left[\left(y_t - Q(\boldsymbol{s}_t,\boldsymbol{a}_t|\boldsymbol{\theta}_Q)\right)^2\right] \tag{20}$$

其中，y_t为奖励目标值，可表示为

$$y_t = \mathrm{Re}_t + \varepsilon Q'(\boldsymbol{s}_{t+1},\pi'(\boldsymbol{s}_{t+1}|\boldsymbol{\theta}_\pi^T)|\boldsymbol{\theta}_Q^T) \tag{21}$$

其中，ε为折扣因子，用于调节未来奖励所占的比重。损失函数$L_s(\boldsymbol{\theta}_Q)$的梯度可表示为

$$\nabla_{\theta_Q} L_s = \mathbb{E}\left[2(y_t - Q(\boldsymbol{s}_t,\boldsymbol{a}_t|\boldsymbol{\theta}_Q))\nabla_{\theta_Q} Q(\boldsymbol{s}_t,\boldsymbol{a}_t|\boldsymbol{\theta}_Q)\right] \tag{22}$$

因此，主Critic网络的参数$\boldsymbol{\theta}_Q$可以采用梯度下降法进行更新

$$\boldsymbol{\theta}_Q = \boldsymbol{\theta}_Q - \frac{\alpha}{N}\sum_{n=1}^{N} 2(y_n - Q(\boldsymbol{s}_n,\boldsymbol{a}_n|\boldsymbol{\theta}_Q))\nabla_{\theta_Q} Q(\boldsymbol{s}_n,\boldsymbol{a}_n|\boldsymbol{\theta}_Q) \tag{23}$$

3) 目标网络参数更新

目标网络的参数更新分为软更新和硬更新两种更新方式。本文的EADE-DDPG算法选取软更新，更新方式为

$$\boldsymbol{\theta}_{Q'} = \varphi\boldsymbol{\theta}_Q + (1-\varphi)\boldsymbol{\theta}_{Q'} \tag{24}$$

$$\boldsymbol{\theta}_{\pi'} = \varphi\boldsymbol{\theta}_\pi + (1-\varphi)\boldsymbol{\theta}_{\pi'} \tag{25}$$

其中，$\varphi \in [0,1]$。

EADE-DDPG算法的模型训练流程如算法1所示，具体流程描述如下。

算法1　基于EADE-DDPG的多集群算力资源调度优化算法

1) 初始化Actor网络和Critic网络的网络参数$\boldsymbol{\theta}_\pi$和$\boldsymbol{\theta}_Q$

2) 复制参数到目标网络的Actor网络和Critic网络，$\boldsymbol{\theta}_{\pi'} \leftarrow \boldsymbol{\theta}_\pi$和$\boldsymbol{\theta}_{Q'} \leftarrow \boldsymbol{\theta}_Q$

3) 初始化缓存回收池$\mathcal{R}$的大小为V，训练轮数M，每轮迭代次数T，批次样本数N，学习率α，权重参数ε和φ，历史样本采样比例β

4) 历史经验缓存池$\mathcal{R}_h$填充

5) for episode=1→ M

6) do:
7) for t=1→ T do:
8) 获取状态空间$\boldsymbol{s}_t$
9) 根据当前策略选择动作$\boldsymbol{a}_t=\pi\left(\boldsymbol{s}_t|\boldsymbol{\theta}_\pi\right)+\boldsymbol{n}_0$
10) 执行动作获得奖励Re_t和新的状态$\boldsymbol{s}_{t+1}$
11) 将$\left(\boldsymbol{s}_t,\boldsymbol{a}_t,\boldsymbol{r}_t,\boldsymbol{s}_{t+1}\right)$以四元组形式存储在经验缓存池$\mathcal{R}$中
12) 从$\mathcal{R}_h$中随机采样$N\beta$个样本，从$\mathcal{R}$随机采样$N\left(1-\beta\right)$个样本，进行训练
13) 令$y_n=\mathrm{Re}_n+\varepsilon Q'(\boldsymbol{s}_{n+1},\pi'(\boldsymbol{s}_{n+1}|\boldsymbol{\theta}_\pi^T)|\boldsymbol{\theta}_Q^T)$
14) 更新Critic网络：$\boldsymbol{\theta}_Q=\boldsymbol{\theta}_Q-\frac{\alpha}{N}\sum_{n=1}^{N}2(y_n-Q(\boldsymbol{s}_n,\boldsymbol{a}_n|\boldsymbol{\theta}_Q))\nabla_{\boldsymbol{\theta}_Q}Q(\boldsymbol{s}_n,\boldsymbol{a}_n|\boldsymbol{\theta}_Q)$
15) 更新Actor网络：$\boldsymbol{\theta}_\pi=\boldsymbol{\theta}_\pi-\frac{1}{N}\sum_n\nabla_a Q(\boldsymbol{s},\boldsymbol{a}|\boldsymbol{\theta}_Q)|_{\boldsymbol{s}=\boldsymbol{s}_n,\boldsymbol{a}=\pi\left(\boldsymbol{s}_n|\theta_\pi\right)}\nabla_{\theta_\pi}\pi\left(\boldsymbol{s}|\boldsymbol{\theta}_\pi\right)|_{\boldsymbol{s}_n}$
16) 更新Critic网络目标函数：$\boldsymbol{\theta}_{Q'}=\varphi\boldsymbol{\theta}_Q+(1-\varphi)\boldsymbol{\theta}_{Q'}$
17） 更新Actor网络目标函数：$\boldsymbol{\theta}_{\pi'}=\varphi\boldsymbol{\theta}_\pi+(1-\varphi)\boldsymbol{\theta}_{\pi'}$
18） end for
19） end for

历史经验缓存池预填充。模型通过历史部署数据经过专家经验标注得到专家数据，并将其存储至历史经验缓存池中。

Actor决策。模型Actor网络从环境中获取当前状态$\boldsymbol{s}_t\left(\boldsymbol{s}_t\in S\right)$，根据策略得到唯一动作$\boldsymbol{a}_t\left(\boldsymbol{a}_t\in A\right)$。

环境更新。动作执行后，环境根据$\boldsymbol{a}_t$得到新的状态$\boldsymbol{s}_{t+1}$。

Critic反馈。模型Critic网络根据Actor网络生成的动作决策$\boldsymbol{a}_t$以及当前状态$\boldsymbol{s}_t$得到奖励Re_t。

数据存储。模型Actor网络根据当前状态$\boldsymbol{s}_t$、动作决策$\boldsymbol{a}_t$、奖励Re_t以及下一状态$\boldsymbol{s}_{t+1}$构建四元组$(\boldsymbol{s}_t,\boldsymbol{a}_t,\mathrm{Re}_t,\boldsymbol{s}_{t+1})$，并将其存储至实时经验缓存池中。当经验池达到其最大容量时，使用最新的数据替换回收经验池中最旧的数据，以此保证训练数据的时效性和有效性。

模型更新。在训练过程中，一旦经验缓存池达到最小抽样大小，模型便会通过采样策略抽取小批次采样数据开展Actor网络和Critic网络的训练工作，进行网络参数的更新，更新过程可参照DDPG算法。

4 实验与分析

为验证上述多集群调度系统调度算法性能，本节搭建了实验仿真平台，利用真实的集群和算力资源需求数据集，对调度算法进行了性能对比分析，进一步验证调度算法的有效性。

4.1 实验设置与数据集介绍

本文基于Python搭建了多集群环境，并使用Pytorch搭建了多集群环境下的EADE-DDPG模型。由于强化学习算法的超参数会影响模型的性能与稳定性，因此本节将进行多次对比实验，选取表现最好的超参数，得到最优的模型。

实验仿真数据来自阿里云开源数据集，为阿里巴巴一个具有超过6 000个GPU生产的MLaaS集群中收集的。本文实验选取其中64个GPU节点组成16个集群来构造多集群环境，其中每个节点拥有不同数量的CPU、内存以及GPU卡，GPU卡型号选择为P4。任务流从数据集中随机抽取220个任务组成，依次进行部署实验。模型训练的最大周期数为1 000。

4.2 EADE-DDPG超参数选择

在深度强化学习模型的训练中，影响较大的因素有采样比例大小、网络层数大小与网络学习率大小。此外，本文设计的EADE-DDPG中引入了两个经验缓存池，并从中抽取一定比例的样本对模型进行训练，两个缓存池的采样比例也会对模型产生较大的影响。因此本节对双缓存池收敛性进行分析，并选取了经验缓存池数据采样比例、网络参数、学习率与缓存样本采样比例4个超参数进行对比实验，选取其中表现最好的参数值作为模型的最终参数。

1) 双经验缓存池收敛性分析

由于历史经验缓存池和实时经验缓存池对网络收敛速度、收敛效果和稳定性均有影响，因此有必要分析采用双经验缓存池方案以后对强化学习算法收敛性的影响。

图3对比分析了在专家样本采样比例为0、10%以及100%下模型的奖励函数值。其中，0代表不从历史经验缓存池中取数据，也就是完全采用强化学习进行资源调度；10%代表每次模型的训练

数据有10%来源于历史经验缓存池，有90%来源于实时经验缓存池，也就是采用双经验缓存池的方案；100%代表完全从历史经验缓存池中取数据，也就是采用监督学习对强化学习进行训练，然后再进行资源调度。

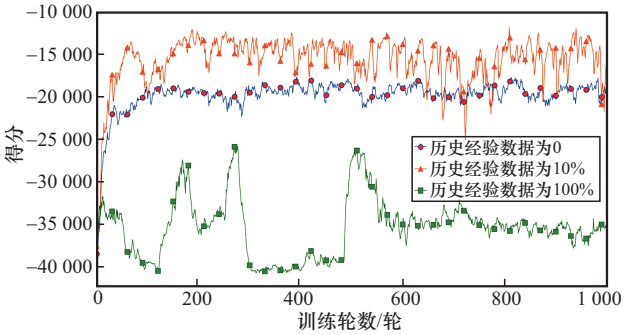


图 3　不同采样比例得分对比

由图3可以看出，在采样比例为10%时，与采样比例为0（完全采用实时经验缓存池）相比，模型能够更快地收敛到更高的值，虽然收敛稳定性比0和100%稍差，但基本在正常范围内波动，而且模型的奖励函数值能收敛到更高的值，表现明显更优异，说明两个缓存池的设计方案确实更有利于模型进行更优的调度，以提升多集群平均资源利用率。采样比例为0（完全采用实时经验缓存池）的奖励函数值比采样比例为100%（完全采用历史经验缓存池）的奖励函数值要高，说明强化学习具有更好的探索能力。综上，采用双缓存池方案，既能够利用历史经验数据加快收敛，又能够利用强化学习探索性达到更好的奖励值。

2) 经验缓存池数据采样比例

由图3的实验结果可知，需要确定每次训练从两个不同的缓存池选择数据的比例，以达到更好的收敛效果。因此，本节也同样对其他采样比例进行了实验，结果如表1所示。由表1可知，历史样本采样比例对模型性能有显著影响。当历史样本采样比例为10%时，最佳得分从−11 742提升至−6 211，表明历史样本的引入提升了模型性能。然而，当历史样本采样比例增加至20%和50%时，最佳得分分别降至−7 765和−8 684，说明过高的历史样本采样比例对模型产生了一定的负面影响。

综上所述，当历史样本采样比例为10%时，模型表现最佳，能够平衡性能提升与稳定性，过高的比例则可能导致性能下降和稳定性变差。

表 1　不同采样比例下的模型性能对比

历史样本采样比例	最佳得分
10%	−6 211
20%	−7 765
50%	−8 684

3) Batch Size选择

Batch Size为64与256的模型得分对比如图4所示。从图4中可以发现，当Batch Size=256时，模型的得分更稳定，且得分稍微高于Batch Size=64的时候。这表明模型采样的Batch Size对模型有较大的影响。

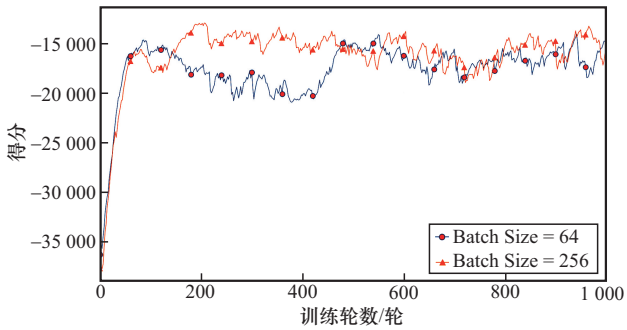


图 4　不同Batch Size得分对比

表2展示了不同Batch Size对算法性能的影响。随着Batch Size的增大，模型的最佳得分先下降后上升，Batch Size=256时最佳得分为−6 211，表明较大的Batch Size有助于提升模型性能，因此本文将Batch Size设置为256。

表 2　不同Batch Size下的模型性能对比

Batch Size	最佳得分
32	−8 459
64	−9 285
128	−8 300
256	−6 211

4) 网络参数选择

图5展示了Actor网络与Critic网络的网络层结构分别为128-64与256-128的得分对比。从图5中可以发现，当网络层结构为128-64时，模型得分更稳定。这表明Actor网络与Critic网络的网络层结构对模型有较大的影响。

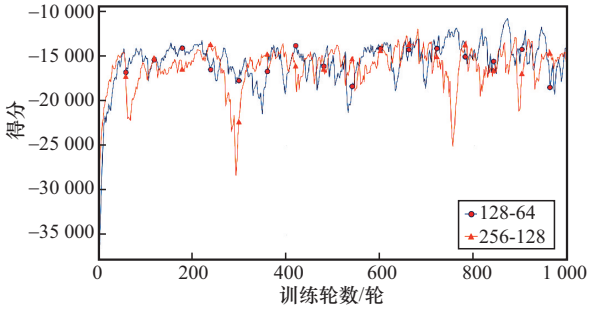


图 5　不同 Layer 得分对比

如表 3 所示，不同网络层结构对算法性能有显著影响。当网络层结构为 128-64 时，模型的最佳得分为−6 934，表现最优。

表 3　不同网络层结构下的模型性能对比

网络层结构	最佳得分
32-32	−7 318
64-64	−8 300
128-64	−6 934
128-128	−8 169

5) 学习率选择

图 6 展示了模型在 Actor 网络与 Critic 网络学习率分别为 0.000 1-0.001 与 0.001-0.000 1 时的得分对比。从图 6 中可以发现，两种参数在最开始时表现十分接近，但是在训练至 500 轮时，学习率为 0.000 1-0.001 的模型出现了得分下滑且波动增大的现象。这表明 Actor 网络与 Critic 网络学习率对模型有较大的影响。

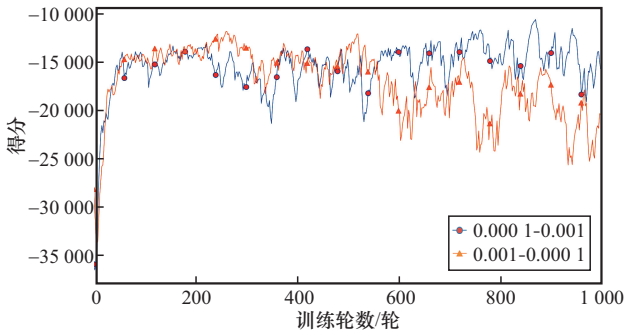


图 6　不同学习率得分对比

表 4 显示了不同学习率对模型性能的影响。当学习率组合为 0.000 1-0.001 时，模型的最佳得分为−6 934，表现最佳。

表 4　不同学习率下的模型性能对比

学习率	最佳得分
0.000 1-0.001	−6 934
0.000 1-0.000 1	−8 614
0.001-0.000 1	−8 702
0.000 1-0.000 5	−8 409

4.3　算法性能对比

1) 实验设置

本节实验的任务设置流与上文保持一致，从阿里云数据集中随机选取批量数据，以先进先出的队列在相同的多集群环境下进行任务部署。根据 4.2 节的超参数选择结果，本节中 EADE-DDPG 超参数设置如表 5 所示。

表 5　模型参数设置

参数	取值
训练周期	1 000
每一周期最大迭代数	220
批次采样数	256
Actor 网络自注意力机制层	16×4+4, 128
Actor 网络两层隐藏层单元个数	128, 64
Critic 网络自注意力机制层	16×4+4+1, 128
Critic 网络两层隐藏层单元个数	128, 64
隐藏层激活函数	ReLU
学习率	0.000 1, 0.001
历史样本采样比例	10%

2) 对比算法

为全面评估多集群调度算法的性能表现，本节选取经典算法进行多维度的指标对比分析，包括 DDPG 算法[8-9]、TD3 算法[16]、SAC 算法[17]、PPO 算法[15]、BestFit 算法[24]、Greedy 算法[25]、Random 算法[9]、Utilization 算法[10]。

DDPG 算法：传统的强化学习算法，基于多集群当前剩余算力资源和任务请求资源，进行任务部署到哪个集群的决策。DDPG 算法只有网络架构和缓存池两个部分与 EADE-DDPG 不同，其他均相同。

TD3 算法：通过采用双重评论家网络，在一定程度上解决了 DDPG 的 Q 值过高估计问题，并通过

延迟策略更新来提高训练的稳定性。

SAC算法：通过采用双重评论家网络，在一定程度上解决了DDPG的Q值过高估计问题，同时通过在奖励函数中引入熵项来鼓励智能体进行更广泛的探索，从而在探索效率上具有显著优势。

PPO算法：使用截断目标函数来限制策略更新的幅度，防止更新过大导致性能崩溃，是一种在策略更新稳定性上表现良好的策略梯度算法。

BestFit算法：将任务分配至资源空闲量最低的集群，是一种经典的启发式算法。

Greedy算法：一种追求速度的算法，在查找到一个满足任务需求的集群时便停止搜索并进行任务的部署。

Random算法：一种随机分配算法，在满足任务需求的集群中随机选择一个集群进行部署。

Utilization算法：一种贴近度算法，倾向于将任务部署至资源余量比例与任务资源需求比例接近的集群上。

3) 性能指标

本节实验中各指标的设计紧密围绕调度系统的核心优化目标（资源利用率最大化与负载均衡），并结合实际工程场景中的关键需求（如实时性、稳定性），形成多维度的量化评估体系。

任务部署成功数量：反映调度器的决策效率与成功率，为所有部署成功的任务以及成功判断不存在可调度集群任务的集合，其定义为

$$N = n_{\text{success}} + n_{\text{suc\_jud}} \tag{26}$$

资源碎片率：量化未分配资源的占比程度，反映调度策略的资源整合能力，避免高碎片率导致后续大规模任务因资源不足而无法部署（资源碎片化现象），其定义为

$$F = \frac{1}{N}\sum_{n=1}^{N}\sum_{r\in R}\frac{r_t^{(n)} - r_o^{(n)}}{r_t^{(n)}} \tag{27}$$

跨集群GPU负载方差：用于评估多集群间的GPU负载均衡性，方差越小说明GPU资源利用越均衡；方差越大可能导致部分集群过载而其他集群GPU闲置，其定义为

$$\sigma_L^2 = \frac{1}{N}\sum_{i=1}^{N}(\overline{U}_n - \frac{1}{N}\sum_{n=1}^{N}\bar{U}_n)^2 \tag{28}$$

4) 实验结果与分析

资源利用率对比如表6所示。从表6可以看出，EADE-DDPG算法在GPU资源利用率上表现优越。无论是GPU内存还是GPU核，其平均利用率均是所有算法中最高的，这表明该算法能够更充分地利用稀缺的GPU资源。更重要的是，相比纯数值计算方法，EADE-DDPG在GPU利用率的标准差上大幅领先，特别是GPU内存的标准差仅为0.025 1，远低于Greedy的0.116 4。标准差的降低是衡量负载均衡性的关键指标，这有力地证明了EADE-DDPG并非简单地将任务堆积在少数集群上，而是智能地将GPU负载均匀地分配到所有集群，有效避免了局部过载和资源闲置。

在GPU资源利用率方面，其他深度强化学习算法的表现不如EADE-DDPG，主要原因可能在于其核心机制的局限性。SAC通过在损失函数中引入信息熵以增加策略探索的随机性，在探索效

表 6　　**各算法集群资源利用率对比**

算法	GPU内存利用率平均值↑	GPU内存利用率标准差↓	GPU核利用率平均值↑	GPU核利用率标准差↓	CPU利用率平均值↑	CPU利用率标准差↓	内存利用率平均值↑	内存利用率标准差↓
EADE-DDPG	**0.953 7**	**0.025 1**	**0.835 0**	**0.095 3**	0.472 4	**0.243 7**	0.516 6	**0.261 7**
DDPG	0.895 9	0.136 6	0.778 4	0.143 7	0.429 8	0.249 9	0.478 9	0.262 9
TD3	0.920 9	0.0.06 4	0.803 5	0.071 0	0.454 7	0.252 2	0.508 1	0.286 1
SAC	0.925 1	0.060 3	0.813 6	0.109 7	**0.498 0**	0.313 2	**0.550 4**	0.334 4
PPO	0.911 9	0.059 2	0.801 0	0.080 0	0.494 3	0.322 3	0.546 6	0.336 6
BestFit	0.914 6	0.089 7	0.808 6	0.093 1	0.463 6	0.290 5	0.520 9	0.317 8
Greedy	0.899 7	0.116 4	0.799 6	0.121 9	0.457 9	0.289 2	0.526 1	0.347 1
Random	0.890 1	0.102 1	0.794 6	0.131 7	0.472 9	0.296 7	0.538 7	0.351 4
Utilization	0.905 0	0.103 4	0.801 9	0.115 3	0.466 8	0.286 0	0.505 9	0.278 3

率上有优势，但其探索比较随机，因此可能会收敛到局部最小值点。TD3和PPO虽然在一定程度上解决了DDPG的Q值过高估计问题，相对于DDPG性能也得到了提升。在所有的强化学习算法中，DDPG算法因其相对较弱的稳定性和探索能力，在复杂的GPU调度环境中难以持续地收敛到最优策略，导致其性能出现利用率较低、标准差较高的现象。值得注意的是，BestFit算法的性能反倒比强化学习的PPO和DDPG优越，说明了强化学习的随机探索性在多集群调度问题上可能不如基于人工经验预先定义规则的启发式算法。这进一步证明了EADE-DDPG中的双经验池策略使其能够更快、更稳定地学习并执行更优的调度策略，从而在GPU资源利用率和负载均衡上取得了压倒性的优势。

与基线算法相比，EADE-DDPG算法在CPU和内存的利用率平均值维度上没有大幅提升，原因在于阿里云数据中CPU和内存资源属于比较丰富的资源，GPU属于比较紧缺的资源，GPU利用率总是远高于CPU和内存的利用率，这就导致在以多资源利用率加权最大化为调度目标的算法会优先最大化紧缺资源利用率。这与实际资源利用情况一致，也就是在实际系统分配时，决定任务部署数量的往往是紧缺资源（如表6所示）。在标准差维度上，EADE-DDPG算法均优于基线算法，这表明EADE-DDPG算法在集群的负载方面更加均衡，有效避免了CPU和内存资源的闲置。

任务部署成功数量对比如表7所示。从表7中可以看出，EADE-DDPG算法在任务部署率上面有了较大幅度的提升，与强化学习算法相比，最高与DDPG算法相比提高了12.97%，最低与SAC相比提高了7.18%；与其他启发式算法相比，最高与Greedy算法相比提高了14.83%，最低与Utilization算法相比提高了11.17%。主要原因还是在阿里云的数据集中，GPU属于稀缺资源，因此GPU利用率决定了所能成功部署的任务数。值得注意的是，DDPG算法在表6的资源利用率方面虽然表现不佳，但在表7的任务部署成功数量上表现有所提升，主要是因为本文部署的任务GPU核心分布范围在[114,5 999] ms，DDPG在进行决策时，可能是先部署了对GPU资源要求比较低的任务，后期对GPU资源要求高的大任务进行部署时，因为某些集群上已经形成了资源碎片化，所以大任务无法部署。如图7所示，在集群10上的GPU资源仍然有空闲，但CPU和内存已不足。最终，DDPG的表现是资源利用率偏低，但任务部署成功数量并不是最低。

表7 任务部署成功数量对比

算法	任务部署成功数量/个	提升率
EADE-DDPG	209	—
SAC	195	7.18%
PPO	193	8.29%
TD3	191	9.42%
Utilization	188	11.17%
DDPG	185	12.97%
Random	184	13.59%
BestFit	183	14.20%
Greedy	182	14.83%

资源碎片率对比如表8所示。从表8可以看出，EADE-DDPG算法在资源碎片整合上表现也较为优异，最低与SAC算法相比降低了5.04%，最高与Greedy算法相比降低了14.50%。整体而言，强化学习算法的资源碎片化率要更高一些，这是因为所有的强化学习算法均是以最大化平均资源利用率为优化目标。显而易见的是，在传统的启发式算法中，Utilization的资源碎片率最低，这是因为它的目标也是最大化平均资源利用率。

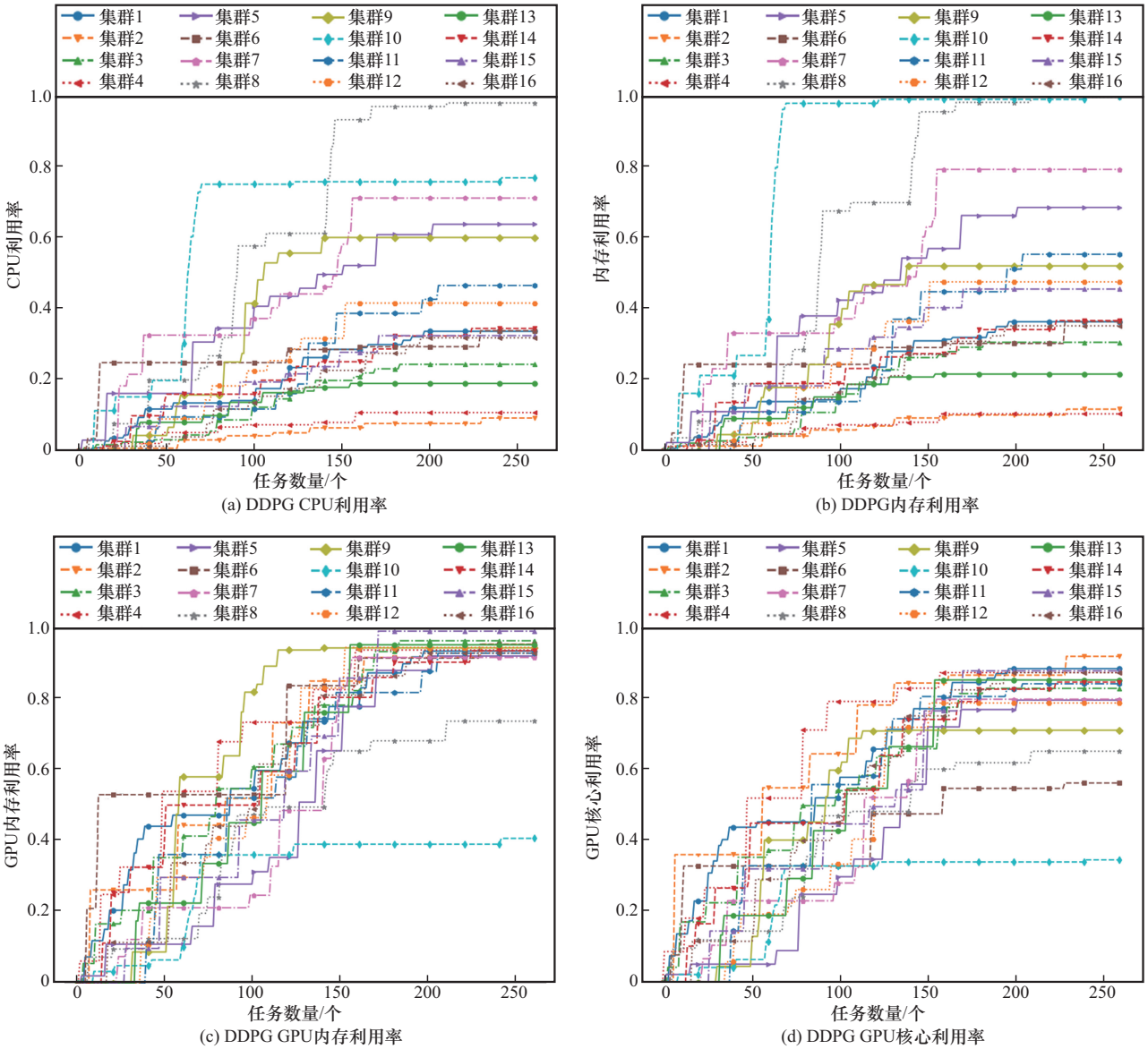
跨集群GPU负载方差对比如表9所示。从表9可以看出，EADE-DDPG算法对GPU负载的利用率也更均衡，其跨集群负载方差仅为0.002 0，最高与DDPG算法相比降低了89.01%，最低与PPO算法相比也降低了34.03%。各种算法在GPU资源利用率上表现越差，证明可能在某个集群上进行决策部署时，形成了GPU的资源碎片，从而导致各个集群之间出现了资源利用率不平衡的现象。例如，如图7所示，DDPG算法正是因为在集群10上部署了过多CPU和内存要求较高、GPU资源要求较低的任务，后续因CPU和内存资源不足，导致该集群也无法部署对CPU、内存和GPU资源同时有要求的任务，因此在该集群上出现了GPU利用率不足的现象。

表 8　资源碎片率对比

算法	资源碎片率	下降率
EADE-DDPG	1.985 3	—
SAC	2.090 7	5.04%
TD3	2.126 3	6.63%
Utilization	2.145 7	7.47%
DDPG	2.146 6	7.51%
BestFit	2.164 0	8.26%
PPO	2.206 4	10.02%
Random	2.255 3	11.97%
Greedy	2.322 0	14.50%

表 9　跨集群GPU负载方差对比

算法	跨集群GPU负载方差	下降率
EADE-DDPG	0.002 0	—
PPO	0.003 0	34.03%
TD3	0.003 8	47.91%
SAC	0.005 0	60.76%
BestFit	0.006 7	72.29%
Utilization	0.009 6	79.57%
Random	0.011 1	82.25%
Greedy	0.012 9	84.74%
DDPG	0.017 9	89.01%



(a) DDPG CPU利用率　(b) DDPG内存利用率

(c) DDPG GPU内存利用率　(d) DDPG GPU核心利用率

图 7　DDPG算法中各集群资源占用情况

5 结束语

本文对多集群算力资源调度问题进行了数学建模，目标是在任务算力资源需求及资源调度约束的情况下最大化多集群多维资源的平均利用率。为求解这一问题，本文设计并实现了基于深度强化学习的EADE-DDPG调度算法。该算法加入了对GPU资源信息的调度，且在动作空间选择上设计了延迟调度，更符合物理多集群调度系统的实际情况。在DDPG算法基础上，EADE-DDPG在神经网络架构上增加了注意力机制，能够实现对复杂算力网络状态关键信息的学习，设计了双经验缓存池，进一步加速算法收敛效果。实验结果表明，EADE-DDPG算法相比于DDPG以及常见的量化评估算法，在多集群GPU算力资源利用率上表现最优，在任务部署成功数量上最高提升了14.83%，在资源碎片率上最高降低了14.50%，在跨集群GPU负载方差上最高降低了89.01%。

尽管本文在多集群资源调度和GPU资源管理方面取得了一定成果，但仍有一些可改进和拓展的方向：1)异构集群算力的统一度量和调度，在跨集群环境下，集群的配置总是不尽相同，以GPU为例，虽然目前在集群中对GPU的调度以核心数与显存作为调度单位，但是不同型号GPU的核心计算能力是不一致的，其他的部分硬件算力资源衡量方式也是如此，因此对集群算力进行度量的统一，优化调度逻辑也是本文后续的工作重点；2)调度算法优化，虽然本文设计EADE-DDPG算法实现了对任务的编排调度，但是并未考虑到任务的等待时长，未来可进一步探索更复杂高效的深度强化学习算法，预先对任务进行运行时长的预估，并且将未部署任务等待时长考虑进算法的反馈中，以进一步贴合实际集群调度的情况。

参考文献：

[1] ACHIAM J, ADLER S, AGARWAL S, et al. GPT-4 technical report[J]. arXiv Preprint, arXiv: 230308774, 2023.

[2] DEVLIN J. Bert: pre-training of deep bidirectional transformers for language understanding[J]. arXiv Preprint, arXiv: 181004805, 2018.

[3] KUBERNETES. Production-grade container orchestration[R]. 2023.

[4] HINDMAN B, KONWINSKI A, ZAHARIA M, et al. Mesos: a platform for fine-grained resource sharing in the data center[C]//Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation. Berkeley: USENIX Association, 2011: 295-308.

[5] MESOS A. Program against your datacenter like it's a single pool of resources[R]. 2023.

[6] SHI L, CHEN H, SUN J H, et al. vCUDA: GPU-accelerated high-performance computing in virtual machines[J]. IEEE Transactions on Computers, 2012, 61(6): 804-816.

[7] GU J, SONG S B, LI Y, et al. GaiaGPU: sharing GPUs in container clouds[C]//Proceedings of the 2018 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications (ISPA/IUCC/BDCloud/SocialCom/SustainCom). Piscataway: IEEE Press, 2018: 469-476.

[8] LIU Q Q, ZHANG H X, ZHANG X, et al. Improved DDPG based two-timescale multi-dimensional resource allocation for multi-access edge computing networks[J]. IEEE Transactions on Vehicular Technology, 2024, 73(6): 9153-9158.

[9] PENG H X, SHEN X S. DDPG-based resource management for MEC/UAV-assisted vehicular networks[C]//Proceedings of the 2020 IEEE 92nd Vehicular Technology Conference (VTC2020-Fall). Piscataway: IEEE Press, 2020: 1-6.

[10] SENJAB K, ABBAS S, AHMED N, et al. A survey of Kubernetes scheduling algorithms[J]. Journal of Cloud Computing, 2023, 12 (1): 87.

[11] SANTOS J, WAUTERS T, VOLCKAERT B, et al. Towards network-aware resource provisioning in Kubernetes for fog computing applications[C]//Proceedings of the 2019 IEEE Conference on Network Softwarization (NetSoft). Piscataway: IEEE Press, 2019: 351-359.

[12] MENOUER T. KCSS: Kubernetes container scheduling strategy[J]. The Journal of Supercomputing, 2021, 77(5): 4267-4293.

[13] YAO Z J, CHEN L, ZHANG H. Deep reinforcement learning for job scheduling on cluster[C]//Artificial Neural Networks and Machine Learning – ICANN 2021. Berlin: Springer, 2021: 613-624.

[14] FAN Y P, LI B Y, FAVORITE D, et al. DRAS: deep reinforcement learning for cluster scheduling in high performance computing[J]. IEEE Transactions on Parallel and Distributed Systems, 2022, 33(12): 4903-4917.

[15] AISWARYA S, GEETHA A, RAMESH K. Intelligent resource provisioning and optimization in fog computing using deep reinforcement learning[J]. International Journal of Electronics and Communication Engineering, 2023, 10(8): 85-97.

[16] HUANG J W, WAN J Y, LV B F, et al. Joint computation offloading and resource allocation for edge-cloud collaboration in Internet of vehicles via deep reinforcement learning[J]. IEEE Systems Journal, 2023, 17(2): 2500-2511.

[17] ZHANG F C, JIANG L H, CHEN J. ETPAM: an efficient task pre-assignment and migration algorithm in heterogeneous edge-cloud computing environments[C]//Proceedings of the 2024 27th International Conference on Computer Supported Cooperative Work in Design (CSCWD). Piscataway: IEEE Press, 2024: 2400-2405.

[18] JIAN Z L, XIE X S, FANG Y Z, et al. DRS: a deep reinforcement learning enhanced Kubernetes scheduler for microservice-based system[J]. Software: Practice and Experience, 2024, 54(10): 2102-2126.

[19] Prometheus: from metrics to insight[R]. 2023.

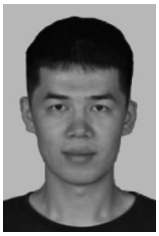
[20] GRAFANA. Grafana OSS and enterprise[R]. 2024.

[21] HAN Y W, SHEN S H, WANG X F, et al. Tailored learning-based scheduling for Kubernetes-oriented edge-cloud system[C]//Proceedings of the IEEE INFOCOM 2021 - IEEE Conference on Computer

Communications. Piscataway: IEEE Press, 2021: 1-10.
[22] HUANG J M, XIAO C M, WU W G. RLSK: a job scheduler for federated Kubernetes clusters based on reinforcement learning[C]//Proceedings of the 2020 IEEE International Conference on Cloud Engineering (IC2E). Piscataway: IEEE Press, 2020: 116-123.
[23] WU J, WANG R, LI R Y, et al. Multi-critic DDPG method and double experience replay[C]//Proceedings of the 2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC). Piscataway: IEEE Press, 2018: 165-171.
[24] JOHNSON D S, DEMERS A, ULLMAN J D, et al. Worst-case performance bounds for simple one-dimensional packing algorithms[J]. SIAM Journal on Computing, 1974, 3(4): 299-325.
[25] AZIZI S, SHOJAFAR M, ABAWAJY J, et al. GRVMP: a greedy randomized algorithm for virtual machine placement in cloud data centers[J]. IEEE Systems Journal, 2021, 15(2): 2571-2582.

[作者简介]

胡亚辉（1982– ），女，河南信阳人，博士，中国矿业大学（北京）副教授、硕士生导师，主要研究方向为数据中心网络优化传输、边缘计算与边缘智能、算力网络、网络流量分析与优化、6G等。

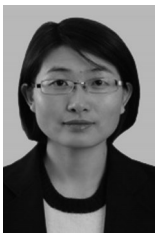


王越嶙（1999– ），男，新疆哈密人，中国矿业大学（北京）硕士生，主要研究方向为云计算与算网融合技术。

张宸康（1996– ），男，河南商丘人，中国矿业大学（北京）硕士生，主要研究方向为云计算与算网融合技术。

洪雨琛（2000– ），男，湖南湘潭人，中国矿业大学（北京）硕士生，主要研究方向为云计算与算网融合技术。

范鹏飞（1982– ），男，内蒙古巴彦淖尔人，中国科学院计算机网络信息中心高级工程师、硕士生导师，主要研究方向为未来网络架构与技术、算网融合技术与应用等。



宋俊平（1985– ），女，河北沧州人，博士，中国科学院计算机网络信息中心助理研究员，主要研究方向为算力网络、网络人工智能等。

周旭（1976– ），男，四川成都人，博士，中国科学院计算机网络信息中心研究员、博士生导师，主要研究方向为计算机网络体系结构、5G移动通信、网络人工智能技术等。

鲍震（1980– ），男，河南临颍人，国能智深控制技术有限公司高级工程师，主要研究方向为工业自动化、工业智能化等。