http://htkz.cn
引用格式：董柏顺，高晗，罗汝斌，等. 基于强化学习任务调度的仿真算法平台设计与实现[J]. 航天控制，2025，43(3)：93-101.
（DONG Baishun，GAO Han，LUO Rubin，et al. Design and implementation of a simulation algorithm platform based on reinforcement learning task scheduling[J]. Aerospace Control，2025，43(3)：93-101.）

基于强化学习任务调度的仿真算法平台设计与实现

董柏顺，高　晗，罗汝斌，郭大庆
北京宇航系统工程研究所，北京 100076

摘　要　随着信息化技术的快速发展和产品复杂性的显著增加，实物实验的成本不断攀升，仿真技术成为系统设计优化的关键工具。然而，传统仿真算法在调度过程中存在计算资源消耗大、全局优化难及适应性差等问题。针对这些问题，本文提出了一套基于深度强化学习的通用仿真算法调度平台方案，包括基于SAC的任务清单分发算法和基于DQN的计算任务调度算法。SAC算法通过智能决策将任务分发至计算节点，优化任务执行效率；DQN算法则通过经验分类方法提升计算节点的自主调度能力，提高资源利用率。实验结果表明，所提出的算法在任务完成时间和资源利用率方面均优于传统算法，验证了其有效性和先进性。

关键词　强化学习；任务调度；容器技术；仿真算法调度平台
中图分类号：TP315　　**文献标识码**：A　　**文章编号**：1006-3242(2025)03-0093-09

Design and implementation of a simulation algorithm platform based on reinforcement learning task scheduling

DONG Baishun, GAO Han, LUO Rubin, GUO Daqing
Beijing Institute of Astronautical Systems Engineering, Beijing 100076, China

Abstract　With the rapid development of information technology and the significant increasement in product complexity, the cost of physical experiments is still on the rise, so that simulation technologies are recognized as critical tool for system design optimization. However, traditional simulation algorithms are challenged by high computational resource consumption, difficulty in global optimization and poor adaptability. Regarding these issues, a general simulation algorithm scheduling platform is proposed and based on deep reinforcement learning, which consists of a task list distribution algorithm based on SAC and a computational task scheduling algorithm based on DQN. Tasks are intelligently distributed to computing nodes through the SAC algorithm to optimize task execution efficiency, and but the autonomous scheduling capability of computing nodes is enhanced by the DQN algorithm through experience classifi-

收稿日期：2025-03-17
作者简介：董柏顺（1996-），男，工程师，硕士，主要研究方向为体系仿真评估与软件工程；郭大庆（1980-），男，高级工程师，硕士，主要研究方向为军事仿真建模与虚实结合仿真，本文通信作者。

cation, thereby improving resource utilization. Experimental results demonstrate that the proposed algorithms achieve superior performance by comparing with traditional methods in terms of both task completion time and resource utilization, which validates their effectiveness and advancement.

Key words Reinforcement learning; Task scheduling; Container technology; Simulation algorithm scheduling platform

0 引 言

随着信息化技术的快速发展和产品复杂性的显著增加,实物实验的成本不断攀升,能够辅助构建高度匹配模型并进行有效的系统行为预测与分析的仿真技术,成为系统设计优化的关键工具[1]。然而,面对复杂仿真算法的大量开发,算法的集成管理和部署成本居高不下。仿真算法调度平台根据不同算法任务的需求和优先级等多种限制因素,将控制中心的任务清单通过智能决策分发至各计算节点,然后计算节点再针对单个节点进行计算任务调度,为用户提供底层通用化的专业算法成果,从而降低开发和部署成本[2]。

目前,优化算法[3]、决策分析[4]、模拟和数据驱动方法[5]、遗传算法[6]和灰狼算法[7]等传统算法被广泛应用于解决这些任务调度问题,但是存在一定缺陷和不足,例如:优化算法需要精确的数学模型和参数;决策分析依赖于专家经验和主观判断;模拟和数据驱动方法高度依赖历史数据的质量和完整性;遗传算法运行时间过长、容易陷入局部最优解;灰狼算法随机性较大,容易导致结果的稳定性和可靠性不足。

随着深度强化学习技术[8]的迅速进展,其高效的学习能力和卓越的自适应性在多个研究领域已展现出显著的成效。深度学习具备强大的感知能力,而强化学习则通过与环境的持续交互实现决策的不断优化,两者的结合为仿真算法调度问题提供了有效的解决方案。深度强化学习相比于传统算法更具有自主学习能力和适应性,通过选择动作与环境进行交互获得反馈,以最小化误差和代价函数为目标从而实现更加高效的决策制定。

针对当前复杂仿真算法调度平台设计的需求和现有技术的不足,提出了一套基于强化学习的通用仿真算法调度平台方案,通过基于SAC的任务清单分发算法和基于DQN的计算任务调度算法,实现了通用仿真算法调度平台中的多任务调度。

1 问题描述

以仿真算法的集成管理和调度为研究背景,仿真任务执行之前,会被汇总到控制中心等待统一调度,当控制中心接收到一批待执行的仿真算法任务时,它必须根据各个任务所需要的资源和优先级等要素信息,对任务进行全局调度,旨在优化任务执行效率和最大化系统资源的利用率。在这个过程中,所面临的挑战是如何在动态变化的环境中,在考虑到任务的优先级和执行所需的计算资源的前提下,实现计算任务与计算资源之间的高效匹配,同时,由于整个调度场景的复杂性以及优化目标多样,直接对整个问题进行建模和求解最优方案,往往导致算法难以收敛。本文提出将整个场景分为两个阶段进行求解:首先,根据CPU 负载、内存占用情况等节点计算资源要素以及执行时间和任务重要程度等任务决策要素,对计算任务和计算资源进行全局分配调度;然后,根据整体的分配情况,各计算节点根据自身资源状况,对接收的任务进行自主调度。通过这种分阶段的方法,形成一套完整的仿真算法调度方案,从而提高仿真任务的执行效率和计算资源的利用率。

2 基于SAC的任务分发算法

本章将重点介绍计算任务分发阶段的算法内容。详细介绍了强化学习中状态空间、动作空间、奖励函数、决策网络模型以及算法训练过程等重要内容。

2.1 马尔可夫决策建模

2.1.1 状态空间

在基于SAC(Soft actor critic)[9]的集中式任务分发算法中,由于将整个控制中心当作一个智能体进行训练,因此观察到的状态空间应该包括全局所有计算节点资源状态、运行情况以及计算任务等状态的空间。那么通过上文问题定义可以将整个状态

空间定义为一个元组,包含两个部分,即 $O^t = \{O_c^t, O_p^t\}$。其中:O_c^t 表示所有计算任务的执行资源需求状态信息,O_p^t 表示所有计算节点持有计算资源的相关状态信息。

对于每个计算任务来说,其具有的属性包括:计算任务的标识位,所需计算资源分布属性,任务优先级属性,可表示为

$$\begin{cases} c_i^t = \left\{x_i, \left(r_i^t[1], r_i^t[2], \cdots, r_i^t[Q]\right), v_i^t\right\} \\ O_c^t = \left\{c_1^t, c_2^t, \cdots, c_N^t\right\} \end{cases} \tag{1}$$

对于每个计算节点来说,其具有的属性包括:计算节点的主机编号,持有的计算资源分布属性,剩余的计算资源分布属性,执行计算任务的成功率属性,可表示为:

$$\begin{cases} p_i^t = \left\{x_i^t, \left(r_i^t[1], r_i^t[2], \cdots, r_i^t[Q]\right), \right. \\ \qquad \left. \left(z_i^t[1], z_i^t[2], \cdots, z_i^t[Q]\right), s_i^t\right\} \\ O_p^t = \{p_1^t, p_2^t, \cdots, p_N^t\} \end{cases} \tag{2}$$

2.1.2　动作空间

仿真算法调度平台中的控制中心可选择的方案有:将计算任务分配给某个合适的计算节点或者当某个任务消耗资源量过大时对任务采取不分配处理。而系统中共有 n 台能够提供计算资源的计算节点,因此总共有 n+1 种选择,因此对于控制中心来说其动作空间是一个大小为 n+1 的离散动作空间,其表示为

$$\begin{cases} A_t = \left\{A_{v_1}, \cdots, A_{v_i}, \cdots, A_{v_n}\right\} \\ A_{v_i} = \left\{d^t, w^t \middle| d^t \in n, w^t \in [0,1]\right\} \end{cases} \tag{3}$$

式中:A_t 表示在时间步 t 的动作,d^t 是该时间步决定的节点分配,每个 d^t 取值为 0~n,而 w^t 是一个二进制决策,表示任务是分配给节点还是选择不分配。

2.1.3　奖励函数

本算法的优化目标为最小化任务完成时间,奖励机制由两个部分组成,第一部分是任务完成奖励 η_t,如式(4)所示,v_i^t 设定为一个固定的正值,表示每个节点执行完子任务清单的时间惩罚,取值为 [-10,0],整个任务清单中任务获得奖励之和组成这一步中任务完成的奖励 η_t:

$$\eta_t = \sum_{i=1}^{M} v_i^t \tag{4}$$

第二部分的奖励是资源利用率的奖励 γ_t,是每个节点资源利用率的奖励总和,如果控制中心能够合理地利用计算节点的资源,避免资源空闲或过载的情况可以给予正奖励,如下所示:

$$\gamma_t = \sum_{i=1}^{M} k_i^t \tag{5}$$

式中:k_i^t 用百分比的形式表示资源利用率,因此取值范围是[0,10]。当计算节点的负载越接近其最大负载时获得的奖励越高,从而鼓励控制中心将任务分配给负载较高的节点,以更好地利用计算资源。

综上所述,时间步 t 内的最终奖励可以表示为下式,其中 β_1 和 β_2 为超参数,用于平衡各个奖励目标之间的权重:

$$r_t = \beta_1 n_t + \beta_2 y_t \tag{6}$$

2.2　Actor-Critic 网络设计

该算法的神经网络结构主要有两个,分别是 Q-Critic 网络和 Actor 策略网络,整体结构如图 1 所示。

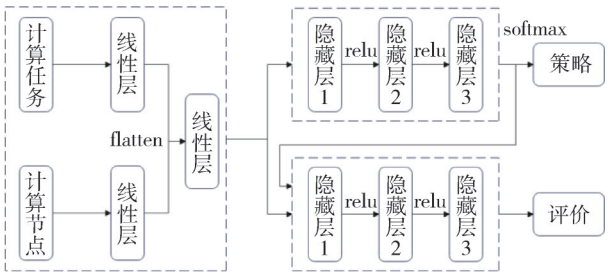

图 1　神经网络结构图

在该网络结构中,Q-Critic 网络和 Actor 网络共用一个状态特征提取层。该层将每个计算任务的属性和每个计算节点的性能指标通过线性变化获得基础编码,再将状态编码展开形成一个 1460 维的状态向量,全面捕捉当前环境的状态。

Actor 网络是核心决策者,接收这个状态向量作为输入,并选择合适的动作 a_t,Actor 网络使用两层隐藏层的全连接神经网络处理信息,每层的节点数为 256 个,这样的设计既可以把握复杂的输入特征,同时也能保持模型的计算效率。Actor 网络的输出大小对应所有任务到节点的映射,对于 m 个计算任务和 n 个计算节点的问题,输出大小应该为 $m \cdot n$。在任务清单分发算法中,特征向量是一个 1460 维向量,动作是一个 4000 维向量,因此 Actor 网络输入层节点为 1460,输出层节点为 4000。Q-Critic 网络作为评价者,输入需要综合状态特征向量和 Actor 网络提供的任务分配方案,Q-Critic 网络同样采用两层隐藏层的设计,每层的节点数为 256 个,以估算在当

前状态下进行特定任务分配的Q值。

在任务清单分发算法中，网络的输入为一个5460维的向量，输出为一个评估Q值。这两个网络使用ReLU激活函数来激励隐藏层的每个神经元，使其能够学习和表达非线性关系。在Actor网络的输出层中应用softmax函数输出一个归一化的概率分布，指导任务的最终分配决策。

2.3 SAC算法训练

基于SAC的任务分发算法训练过程中，首先初始化策略网络、Q网络和目标Q网络的参数设置。为了提升算法的探索能力和避免陷入局部最优，算法中引入了熵正则化项[10]，并使用自适应的温度参数平衡探索与利用的关系。在样本收集阶段，算法根据当前策略与环境的交互收集经验样本，并将这些样本存储至经验回放缓冲区中。训练过程中，算法从缓冲区中随机抽取样本进行重要性采样，并利用这些样本更新策略网络和Q网络，同时可以动态调整学习率以优化训练效果。策略迁移允许算法在已学习到的策略网络上探索新的状态空间并继续训练。训练将持续进行，直到满足预设的停止条件或达到一定的训练轮次。

3 基于DQN的计算任务调度算法

3.1 马尔可夫决策建模

3.1.1 状态空间

本节定义的算法把单个计算主机作为智能体进行训练，考虑状态的可观测性、纬度及粒度等因素，将该场景下状态空间定义为元组$O^t=\{O_c^t,\ O_p^t\}$，其中，O_c^t表示所有计算任务的属性特征，O_p^t表示当前计算节点的属性特征，通过这个元组定义全局资源状态，为智能体提供能够理解和表示其所处环境的方式。

每个计算任务包含的具体属性特征包括：计算任务编号，所需计算资源数量特征，任务优先级属性，计算任务种类和当前执行状态，如式(7)所示，其中$t_i^t\in\{0,1\}$，0表示计算任务为CPU密集型，1表示计算任务为IO密集型：

$$\begin{cases} c_i^t=\left\{x_i^t,\ (r_{\text{cpu}}^t,\ r_{\text{mem}}^t),\ v_i^t,\ t_i^t,\ s_i^t\right\} \\ O_c^t=\left\{c_1^t,\ c_2^t,\ \cdots,\ c_N^t\right\} \end{cases} \tag{7}$$

当前计算节点包含的属性如式(8)所示，包括总计算资源的数量分布，已分配计算资源的数量分布，计算节点崩溃概率和当前系统时间：

$$\begin{cases} p_i^t=\left\{\left(r_{\text{cpu}}^t,\ r_{\text{mem}}^t\right),\ \left(u_{\text{cpu}}^t,\ u_{\text{mem}}^t\right),\ l_i^t,\ k_i^t\right\} \\ O_p^t=\left\{p_1^t,\ p_2^t,\ \cdots,\ p_N^t\right\} \end{cases} \tag{8}$$

3.1.2 动作空间

计算节点的动作空间包含3种动作：执行、等待和中断。具体而言，执行动作是选择一个计算任务c_i并在资源允许的情况下执行；等待动作是指当前没有足够资源执行任务或所有任务都在执行中，系统等待可用资源出现；中断动作是指任务c_i超过了系统设定的最长执行时间后视为任务执行失败，选择中断这个任务。动作空间可表述为

$$\begin{aligned} A=\{&a_{\text{exe}}(c_1),\ \cdots,\ a_{\text{exe}}(c_n),\ a_{\text{wait}},\ a_{\text{interrupt}}(c_1),\\ &\cdots,\ a_{\text{interrupt}}(c_m)\} \end{aligned} \tag{9}$$

式中：$c_1,\cdots,c_n$是所有待执行任务，而$c_1,\cdots,c_m$是正在执行的任务集合。根据如上定义，动作空间的大小是可执行任务的数量加上等待动作及可能中断的动作，此处假设任何时刻只执行一个任务，则大小为$|A|=n+1+m$，n表示可以执行的任务数，1表示等待动作，m表示可能需要中断的任务数量。动作选择的规则可以设定如下：

1)检查每个待执行任务的资源需求，若当前资源满足，将其添加到可选动作集；

2)如果所有待执行任务的资源需求都不满足则选择等待动作；

3)检查每个正在执行的任务的执行时间是否超过最大允许执行时间，如果是，将其中断动作添加到可选动作集。

3.1.3 奖励函数

根据状态空间和动作空间的定义，奖励函数r_t为

$$r_t=\beta_1\eta_t-\beta_2\gamma_t+\beta_3\xi_t \tag{10}$$

γ_t表示时间惩罚，与任务执行时间成正比，通过自上一个动作以来的时间增量计算得出。ξ_t表示资源利用率奖励，定义为资源使用量与资源总量的比例，当任务使用的资源量与当前可用资源的比例高时获得正奖励。

式中：β_1，β_2和β_3为超参数，η_t是任务完成的奖励，计算方式如下：

$$\begin{cases} v_i^t=\begin{cases} v_i, & \sum_{j=1}^{J}o_i^j\leqslant 0 \\ 0, & \sum_{j=1}^{J}o_i^j\geqslant 0 \end{cases} \\ \eta_t=\sum_{i=1}^{N}v_i^t \end{cases} \tag{11}$$

式中:v_i^t表示每个计算任务完成之后能够获得的奖励分数,$\sum_{j=1}^{J} o_i^j \leqslant 0$表示对于计算任务$i$来说,所需要的CPU及内存等计算资源之和小于等于0时,说明相应计算节点单元已经提供给计算任务单元所需的计算资源,V_i表示完成计算任务后获得的奖励反馈。$\sum_{j=1}^{J} o_i^j \geqslant 0$表示计算任务执行时还有所需的计算资源未被满足,导致任务未完成,因此奖励为0。

3.2　基于经验分类的经验池设计

在主机进行资源调度的场景中,涉及如何高效分配有限的计算资源以优化完成时间和资源利用率。在该问题中,对决策的即时反馈可能并不总是立即可得,正面经验相比于负面经验可能更为稀缺。因此使用一种经验分类的方法,如图2所示,帮助智能体更加均衡地学习不同类别的经验,无论资源分配成功还是未能达到预期分配,提升了算法的整体学习效率。将经验分为如下几类:

1)资源分配成功,明显提升了系统性能的经验;

2)资源分配没有显著影响系统性能的经验;

3)资源分配导致性能下降的经验。

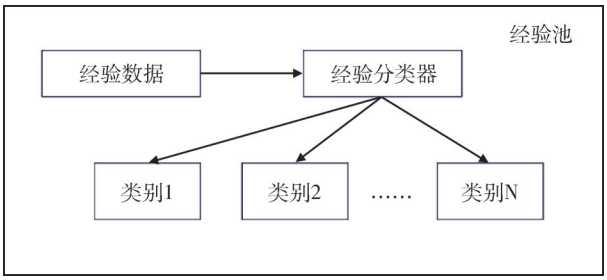

图2　基于经验分类的经验回放池

加入了经验分类方法后的DQN算法在训练神经网络时会抽取不同经验的等量样本混合后训练,提高网络拟合的效率。

3.3　DQN算法训练

在基于DQN的计算任务调度算法中,训练首先初始化了经验回放池D、动作价值函数Q以及目标动作价值函数Q'。算法通过预设经验类别优化学习过程。每个训练周期,算法从环境读取当前状态,并通过贪心策略在每个时间步t选择并执行动作,进而收集奖励和新状态。这些经验元组被存入D并分类标记。算法从D中随机抽取样本,根据是否为终止状态计算目标值,并通过梯度下降法更新评估网络Q以最小化损失。每隔C步,目标动作价值函数参数更新为当前的w,确保训练的稳定性。训练持续至满足停止条件,确保算法学习到高效的任务调度策略。

4　实　验

为了验证算法的有效性和先进性,进行以下3个方面的实验设置:实验环境设置、任务清单分发实验和计算任务调度实验。

4.1　实验环境设置

实验采用Pycharm作为开发工具,实验环境为Python 3.9和PyTorch 1.12.0。具体运行环境和配置如表1所示。

表1　实验环境

配置项	描述
操作系统	Windows 11
开发工具	PyCharm
GPU型号	NVIDIA GeForce RTX 3060
CUDA版本	11.3
语言	Python 3.9
PyTorch版本	1.12.0

4.2　任务清单分发实验

4.2.1　评价指标

在本章节中,我们明确了问题优化的核心目标,即最小化任务完成时间。为了实现这一目标,专注于计算节点的任务完成时间,这是衡量整体计算任务调度效率的关键因素。

4.2.2　收敛性实验

首先对不同学习率下的算法模型收敛性进行实验验证与分析。本章选取了0.03、0.003和0.0003这3个学习率,以比较不同学习率对算法模型回报奖励的影响,如图3所示。

由图3可知,本实验对每种学习率下的算法进行了1000个训练轮次的分析。在学习率为0.03的条件下,算法展示了最快的收敛速度,但未能达到稳定状态。当学习率设置为0.003时,算法在大约200轮后开始收敛,并显示出较高的稳定性。而以0.0003的学习率进行训练时,算法的收敛速度显著减慢,且在500轮次左右仍未表现出稳定趋势。综合考虑,算法的最优表现与学习率的设定并非成正比关系。基于上述观察,确定以0.003的学习率进行后续实验。

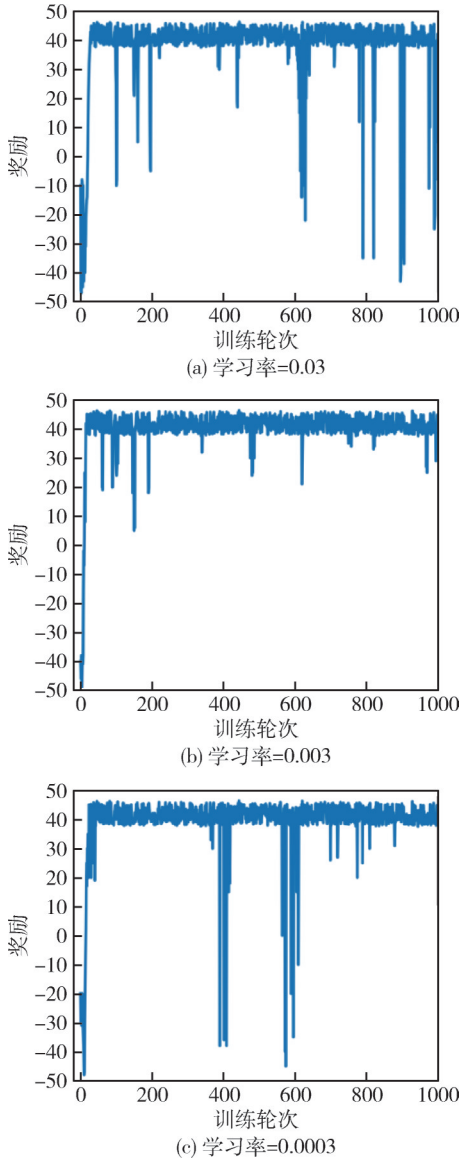

图 3　不同学习率下的训练图

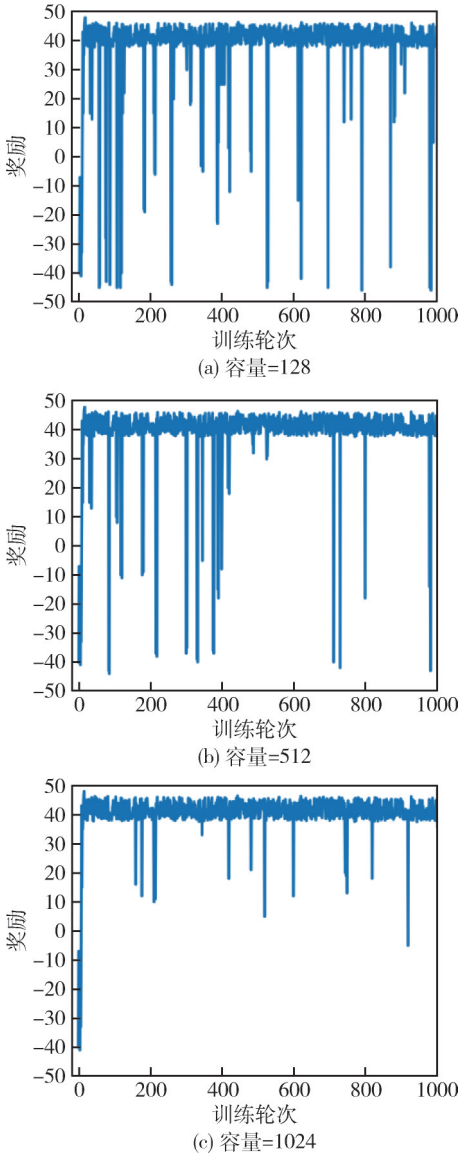

图 4　不同经验池容量下的训练图

实验进一步探讨了不同经验池容量对算法性能的影响。本研究选择了容量分别为 128、512 和 1024 的经验池进行测试，如图 4 所示。实验结果表明，当经验池容量设置为 128 时，算法未能实现收敛。将容量增至 512 时，训练效果有一定提升但收敛性仍然较差。进一步扩大至 1024 容量时，算法成功收敛并且展现了较高的稳定性。

4. 2. 3　对比试验

本章定义的问题优化目标是最小化任务完成时间，因此在本章节的实验中重点关注各计算节点的任务完成时间，并将各节点中任务完成的最长时间定义为整体计算任务调度的完成时间。为了全面评估所提算法的性能，本研究引入随机选择、贪心策略和深度确定性策略梯度（DDPG）3 种基线算法，进行了对比实验。其中随机选择是通过完全随机的策略决定任务分发；贪心策略对所有计算任务的优先级进行排序，在每个时隙，根据贪心策略选择当前优先级最高的计算任务；DDPG 算法是一种深度强化学习技术，采用确定性策略优化控制决策，其状态空间、动作空间和奖励函数的定义与本节所提出的基于 SAC 方法的任务分发算法相同。

图 5 展示了在不同计算任务数量条件下，TD-SAC 算法与其他 3 种算法的性能对比。其中横轴为计算任务规模，纵轴为完成任务的时间消耗（s）。从图中可知，随着计算任务数量的增加，所有算法的总时间消耗呈现出上升趋势。当任务规模较小时，每个主机分到的任务较少，4 种算法的时间消耗区别较小。随着任务规模增加，随机选择方法的时间

消耗出现了明显的增加。与随机选择相比,基于贪心策略的方法表现出了明显的性能提升,但其时间消耗在所有条件下仍高于两种强化学习算法。在任务数量较低的场景中,DDPG 算法的性能与 TD-SAC 算法几乎相同。然而,随着任务数量的增加,TD-SAC 算法的性能表现逐渐超越 DDPG 算法。

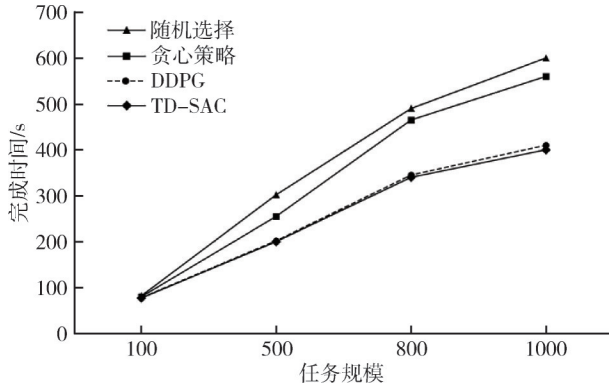

图 5　各算法性能对比

4.3　计算任务调度实验

4.3.1　评价指标

本小节重点介绍在任务清单分发实验中采用的性能评价指标。这些指标由合作单位某航天研究院提供,用以全面评估任务分发系统的性能。任务完成时间作为首要评价指标,直接反映了系统在处理和分发任务时的效率。资源利用率则体现了系统在分配计算资源时的优化程度,是衡量资源分配效果的关键指标。

4.3.2　收敛性实验

本研究对 TS-DQN 算法在不同学习率设置下进行了实验,分别为 0.02、0.002 和 0.0002,如图 6 所示。对每种学习率下的 TS-DQN 算法进行了 1000 个训练轮次。实验结果表明,当学习率为 0.002 时,算法展示出最快的收敛速度,且在大约 100 轮后即可达到收敛,显示出最佳的效果。相比之下,学习率为 0.02 时,算法的收敛速度较慢,在 600 轮左右才开始表现出收敛迹象,且整体效果较差。而在学习率为 0.0002 的情况下,虽然算法表现较好,但收敛速度略慢,在 200 轮左右开始收敛。综合考虑,学习率为 0.002 的设置下,TS-DQN 算法的性能表现最为优异。

本研究中采用了基于经验分类的 DQN 算法,如图 7 所示,将 TS-DQN 算法与原生 DQN 算法的收敛性能进行了对比。实验结果显示,TS-DQN 算法在

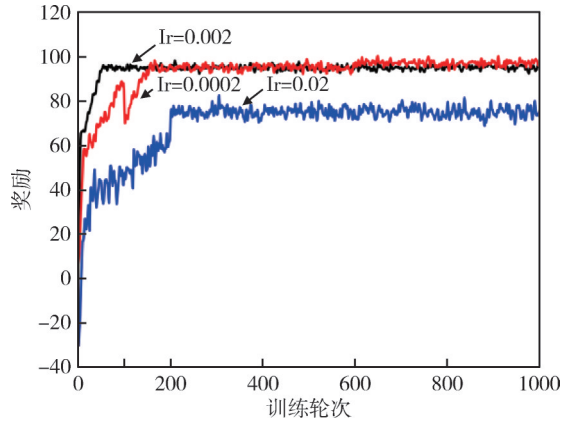

图 6　不同学习率下的收敛表现

100 轮左右达到收敛,而原生 DQN 算法则需 150 轮左右才开始收敛,且收敛效果略差于 TS-DQN 算法。这一结果证明了采用经验分类方法在提升算法收敛速度和效果上的有效性。

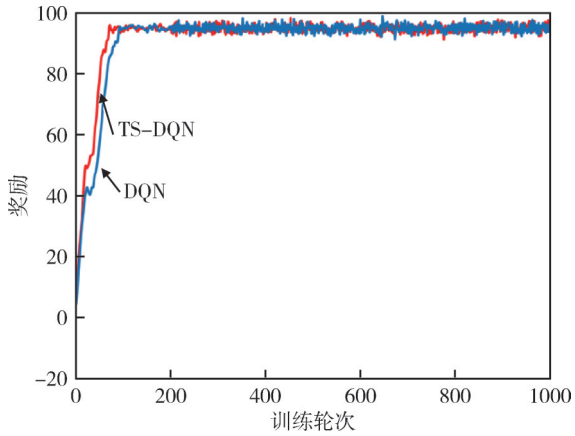

图 7　TS-DQN 算法与 DQN 算法收敛性对比

4.3.3　对比实验

对于计算任务调度问题,其主要评价指标分为任务完成时间和资源利用率两个部分。为了全面评估 TS-DQN 算法的有效性,本章引入以下几种基线算法进行对比实验:顺序选择、原生 DQN 算法和 DDPG。其中顺序选择方法是按照任务到达顺序直接执行,将这个算法的结果作为基本参考指标。原生 DQN 算法和 DDPG 算法都属于强化学习方法范畴,与本章使用方法有相同的建模过程。这几种方法与 TS-DQN 算法的性能对比如下:

1)任务完成时间

图 8 比较了随着任务数量增加时不同算法的计算任务完成时间。其中横轴为任务规模,纵轴为任务完成时间(s)。从图中可以观察到,顺序执行算法

由于采用了任务到达后的顺序处理方式，其时间消耗远高于其他算法。DDPG算法在任务数量较少时表现较好，但随着任务数量的增加，其性能有所下降。这种现象可能由于计算节点不仅要执行计算任务还需负责训练神经网络，随着任务的增加，资源不足以支撑DDPG复杂网络的训练。与此同时，原生DQN算法和本研究提出的TS-DQN算法具有相对简单的网络结构，因此它们对任务数量的增加不敏感，整体表现较好。特别是，TS-DQN算法通过分类处理经验池，相比原生DQN算法显示了更优的性能表现。

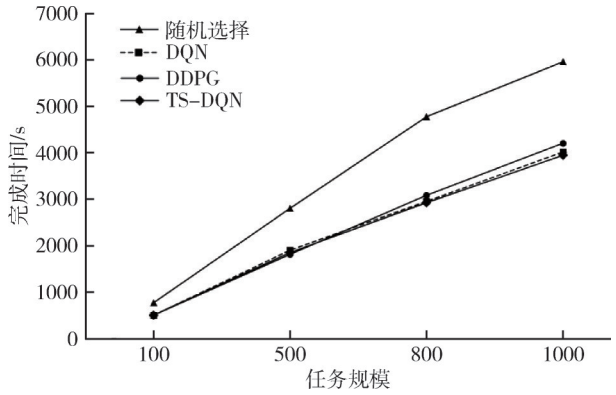

图8 各算法性能对比(任务完成时间)

2)资源利用率

图9比较了不同算法在各种计算任务数量下的资源利用率。纵轴为资源利用率，最大的可能值为1，横轴为任务规模。从结果可以看出，顺序处理方法的资源利用率始终维持在较低水平，约为0.2。DDPG算法在任务数量较少时能够维持较高的资源利用率，但随着任务数量的增加，计算节点的资源更多地用于处理计算任务，从而减少了用于DDPG复杂网络结构训练的资源，导致算法性能下降，资源利用率稳定在约0.7的水平。TS-DQN算法通过在经验回放中采用经验分类的方法，使得智能体能更均衡地学习策略，取得了比原生DQN算法更好的效果，且由于网络结构较为简单，在任务数量较多时仍然能有较好的表现。

5 结 论

通过强化学习的方法设计了一套面向复杂仿真场景的通用算法调度平台。研究从实际需求出发，将仿真算法调度问题分解为两大子问题：任务

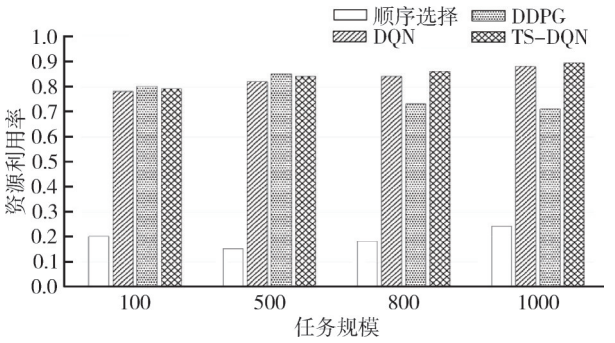

图9 各算法性能对比(资源利用率)

清单分发问题和计算任务调度问题。针对这两个阶段，分别提出了基于SAC的任务清单分发算法和基于DQN的计算任务调度算法。实验数据表明，任务清单分发阶段中，TD-SAC方法相较于传统随机选择算法，在0.003的学习率、1000的任务规模下耗时减少50%；计算任务调度阶段中，基于经验分类的TS-DQN算法使资源利用率达到0.89，较顺序执行方法提升了3倍，同时采用容器化技术解决了算法开发中的跨语言运行冲突问题。该方法在任务执行效率和资源利用率上均表现出显著优势，验证了其有效性和先进性。

参 考 文 献

[1] GUO L, WANG M, RUAN C, et al. A cloud simulation based environment for multi-disciplinary collaborative simulation and optimization[C]//The 21st IEEE International Conference on Computer Supported Cooperative Work in Design, Wellington, New Zealand, April 26-28, 2017.

[2] WANG Z C. Development and formation of simulation science[J]. Acta Simulata Systematica Sinica, 2005.

[3] ALI M A. Design and performance evaluation of an adaptive resource allocation scheme for MIMO-OFDMA systems with fairness constraint[D]. Saudi Arabia: King Fahd University of Petroleum and Minerals (Saudi Arabia), 2011.

[4] HUANG J, YIN Y, ZHAO Y, et al. A game-theoretic resource allocation approach for intercell device-to-device communications in cellular networks[J]. IEEE Transactions on Emerging Topics in Computing, 2014, 4(4): 475-486.

[5] YANG T, ZHANG R, CHENG X, et al. Graph coloring based resource sharing (GCRS) scheme for D2D communications underlaying full-duplex cellular networks [J]. IEEE Transactions on Vehicular Technology,

2017,66(8): 7506-7517.

[6] CHEN H M, CHEN S Y, HSUEH S H, et al. Designing an improved ML task scheduling mechanism on kubernetes[C]//Sixth International Symposium on Computer, Consumer and Control (IS3C), Taichung, June 30-July 2, 2023.

[7] 孙鹏,武君胜,廖梦琛,等. 基于自适应遗传算法的战场资源动态调度模型及算法[J]. 系统工程与电子技术,2018,40(11):2459-2465. (SUN Peng, WU Junsheng, LIAO Mengchen, et al. Battlefield resource dynamic scheduling model and algorithm based on improved self-adaptive genetic algorithm [J]. Systems Engineering and Electronics, 2018, 40(11): 2459-2465.)

[8] SOUALHIA M, KHOMH F, TAHAR S. Task scheduling in big data platforms: a systematic literature review [J]. Journal of Systems and Software, 2017, 134: 170-189.

[9] HAARNOJA T, ZHOU A, ABBEEL P, et al. Soft actor-critic: off-policy maximum entropy deep reinforcement learning with a stochastic actor [C]//The 35th International Conference on Machine Learning, Stockholm, Sweden, July 10-15, 2018.

[10] SHANNON C E. A mathematical theory of communication[J]. ACM SIGMOBILE Mobile Computing and Communications Review, 2001, 5(1): 3-55.