Computer Science and Application 计算机科学与应用, 2021, 11(7), 2008-2018
Published Online July 2021 in Hans. http://www.hanspub.org/journal/csa
https://doi.org/10.12677/csa.2021.117205

Hans汉斯

一种基于深度强化学习的资源调度方法

赵飞鸿，孙立峰

清华大学计算机科学与技术系，北京
Email: zfh18@mails.tsinghua.edu.cn, sunlf@mail.tsinghua.edu.cn

收稿日期：2021年6月23日；录用日期：2021年7月21日；发布日期：2021年7月28日

摘　要

启发式调度算法作为云数据中心资源调度的常用方法能够给出一种可行的资源调度策略，但针对特定集群环境设计合适的启发式算法需要较专业的先验知识，而使用通用的启发式方法会对集群资源造成极大浪费。对Kubernetes集群资源调度过程进行建模，设计基于深度强化学习的资源调度方法，对于不同的集群环境和优化目标，使用学习的方法从历史数据中学习得到相应的调度策略，优化集群资源利用情况。实验结果表明，使用该方法比现有启发式方法在任务带权周转时间上平均减少7.5%，最高减少13.5%。

关键词

云计算，资源调度，深度强化学习

A Resource Scheduling Method Based on Deep Reinforcement Learning

Feihong Zhao, Lifeng Sun

Department of Computer Science and Technology, Tsinghua University, Beijing
Email: zfh18@mails.tsinghua.edu.cn, sunlf@mail.tsinghua.edu.cn

Received: Jun. 23rd, 2021; accepted: Jul. 21st, 2021; published: Jul. 28th, 2021

Abstract

As a common method of resource scheduling in cloud data centers, heuristic scheduling algorithm can give a feasible resource scheduling strategy. However, designing a suitable heuristic algorithm for a specific cluster environment requires professional prior knowledge, or using general heuristic methods will cause a great waste of cluster resources. We design a resource scheduling method on Kubernetes system based on deep reinforcement learning. For different cluster environments and optimization goals, using learning methods to obtain corresponding scheduling strategies

文章引用: 赵飞鸿, 孙立峰. 一种基于深度强化学习的资源调度方法[J]. 计算机科学与应用, 2021, 11(7): 2008-2018.
DOI: 10.12677/csa.2021.117205

from historical data can optimize cluster resource utilization. Experimental results show that the use of this method reduces the slowdown of tasks by an average of 7.5% and the highest reduction of 13.5% compared with the existing heuristic methods.

Keywords

Cloud Computing, Resource Scheduling, Deep Reinforcement Learning

1. 引言

随着互联网技术的快速发展，云计算技术逐步走入人们的生活。云计算通过互联网将遍布世界各地数据中心的各种 IT 资源(计算、存储等)提供给用户使用，较大规模的数据中心目前已有数十万台服务器[1]，而对如此大规模数据中心的资源进行有效管理是学术界和工业界的一大难题。

在学术界和工业界，资源调度问题通常认为是 NP 难问题[2]。资源调度主要面临以下几个方面的挑战：首先是调度的任务是多样的；其次各种集群管理系统的资源调度过程是一个十分复杂的流程，通常很难设计出适用于所有调度场景的通用启发式调度算法；最后集群环境通常异构，对于不同集群和不同优化目标需要对启发式方法中的众多参数进行配置，这往往需要专业人员进行特定分析[3]。

本文利用深度强化学习适于解决序列决策问题的优势，将深度强化学习方法用到解决数据中心的资源调度问题中，同时通过设计合理奖励函数来优化资源调度目标。深度强化学习方法可以根据不同集群环境和不同优化目标让智能体自主学习对应的策略，解决了传统启发式方法很难设计以及参数难调的难题，是启发式调度算法的良好替代方案。

2. 相关工作

目前，学术界和工业界对数据中心的资源调度问题进行了广泛研究。随着人工智能领域的发展，使用深度强化学习解决序列决策问题的应用越来越多。文献[4]提出了一种使用强化学习方法解决资源调度问题的方法。文献[5]中提出的 DeepRM 模型将资源调度系统的状态信息建模成图像形式，将获取的图像信息输入到深度神经网络中的卷积神经网络中，通过卷积神经网络对图像信息进行快速特征提取，然后使用强化学习的方法对神经网络中的参数进行迭代更新，形成最终的策略。文献[6]提出的 Decima 模型基于 Spark 集群环境建模了一个事件驱动的仿真环境，使用图神经网络对等待调度的任务进行特征提取，通过使用可扩展的图神经网络可以处理任意形状和大小的具有依赖关系的任务。文献[7]使用阿里巴巴采集的真实集群的日志作为数据集，生成相应的任务流，同时将集群建模成各个机器，考虑了集群中机器的碎片效应。文献[8]使用两层深度强化学习模型来共同完成对集群资源和能耗的管理工作。其中一个深度强化学习模型负责分析集群状态，对任务进行合理调度，而另外一个深度强化学习模型则分析集群内机器的状态，对机器的能耗进行建模，考虑是否对机器进行关机开机处理，进行合理能耗管理。

本文基于 Kubernetes 集群设计了离线仿真环境，针对该环境进行强化学习建模，设计相应的状态、动作和奖励函数，使用深度强化学习方法对模型进行训练对比，验证了使用深度强化学习方法解决资源调度问题的可行性。

3. 系统模型

3.1. 资源调度系统模型

集群系统由 m 个独立的主机(节点)组成，其中每个主机的资源配置可能是异构的。对于每个主机的资源考虑 CPU 资源和内存资源，在调度过程中，每个集群中的待调度任务包含对 CPU 资源和内存资源的需求情况以及任务的实际执行时间，默认任务的资源需求和实际执行时间都已知。在进行资源调度过程中，每个主机均可以作为任务调度的选择目标，当调度的任务资源需求可以被所调度主机满足时，调度有效。

3.2. 仿真环境设计

本文基于 Kubernetes 的资源调度过程设计了离线仿真环境 Deepk8s。Deepk8s 仿真环境使用 Python 语言进行编写，其整体结构如图 1 所示。仿真环境主要包含四个部分：任务生成器、集群机器、集群监视器和资源调度器。

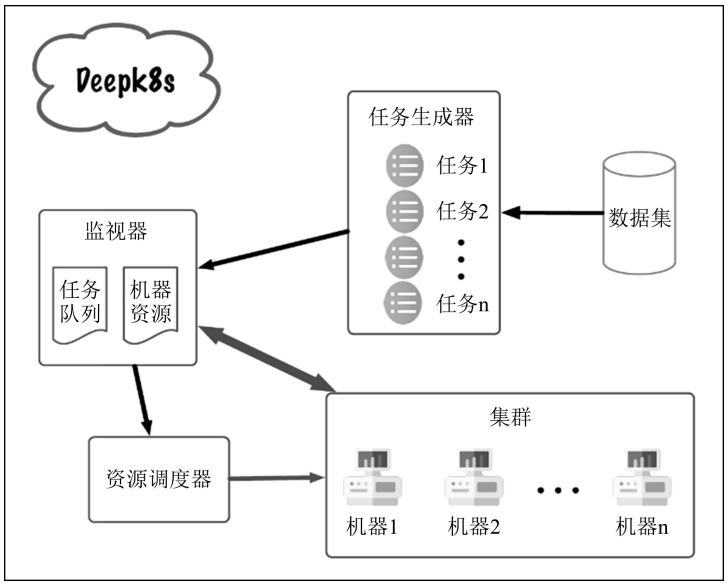


Figure 1. The overall structure of the Deepk8s simulation environment
图 1. Deepk8s 仿真环境整体结构

任务生成器的功能为从数据集中读取数据，然后根据读取到的数据在指定时刻生成相应的任务并提交到集群中进行调度和运行。任务生成器将读取到的任务信息按照任务到达的先后顺序以队列的形式进行存储，当仿真环境的时间运行到任务队列队首任务的到达时刻时，将队首的任务进行出队操作并将任务提交到集群中。集群机器是对真实场景中的主机的抽象，在仿真环境中需要将集群的资源总数和已分配资源数进行表示，根据这些信息可以知道集群的运行状态。集群监视器主要负责在调度过程中获取机器运行情况和集群任务信息，将这些信息反馈给资源调度器。资源调度器则是根据集群信息和任务信息来决定对哪个任务进行调度以及将该任务调度到哪个机器节点进行运行。

Deepk8s 仿真环境是基于事件驱动的环境，在仿真过程中需要调度的事件有以下三个：一次调度结束、新任务到达以及运行任务完成。在调度器刚进行一次调度之后，可以开始下一次调度，这样保证了集群中任务可以被快速调度以及可以将集群资源快速利用起来。当集群中没有任务或者集群中资源数量

不足时，通常需要等待新任务到达或者运行任务完成这两个事件的到来。新任务可能是资源需求较小的任务，因此在新任务到达之后需要考虑是否可以进行一次有效调度；而运行任务完成之后会对集群资源进行释放，因此需要考虑是否可以将之前积压的任务进行调度。

4. 算法设计

本文使用深度强化学习方法来解决资源调度问题。设计深度强化学习模型时首先需要对三个主要要素进行设计，分别是状态、动作和奖励函数。状态是指对智能体所处环境的表示，是智能体对外界环境的感知，动作是智能体在所处环境下所做出的决策，根据该决策环境将进行相应变化，奖励则是对智能体所做决策优劣的反馈。

4.1. 状态设计

Deepk8s 仿真环境中的状态包括集群机器状态和集群中待调度任务状态，如图 2 所示，图中状态空间分为 3 个部分，顶部为机器状态，图中包括 3 台机器的状态，图中所有机器 CPU 资源有 8 个，内存资源有 10 个。其中机器 1 已经分配的 CPU 资源为 5 个，剩余 CPU 资源为 3 个，已分配内存资源 8 个，剩余内存资源 2 个。中间部分为集群中待调度任务状态，其中在任务状态中多了一个维度，即纵轴表示时间。图中纵轴则表示任务的实际执行时间，横轴同样为任务对资源的需求。例如图中任务 1 的实际执行时间为 2 个单位时间，其中对 CPU 资源的需求为 4 个，而内存资源的需求为 1 个。就图中状态而言，可以将任务 1 调度到机器 2 和机器 3 上，而不能调度到机器 1，因为机器 1 的剩余 CPU 资源不能满足任务 1 的 CPU 资源需求。图中底部为积压队列状态，设计积压队列状态的目的是固定待调度任务状态的大小。在不同调度时刻，我们无法得知系统中待调度任务的数量，而神经网络的输入需要固定维度的向量，因此这里固定待调度任务长度为 n，图中 $n = 3$。对于待调度任务数量超出 n 的情况，将超出的任务按照任务到达先后顺序存储在积压队列中。积压队列状态仅包含积压队列的长度以及积压队列队首任务的基本信息。

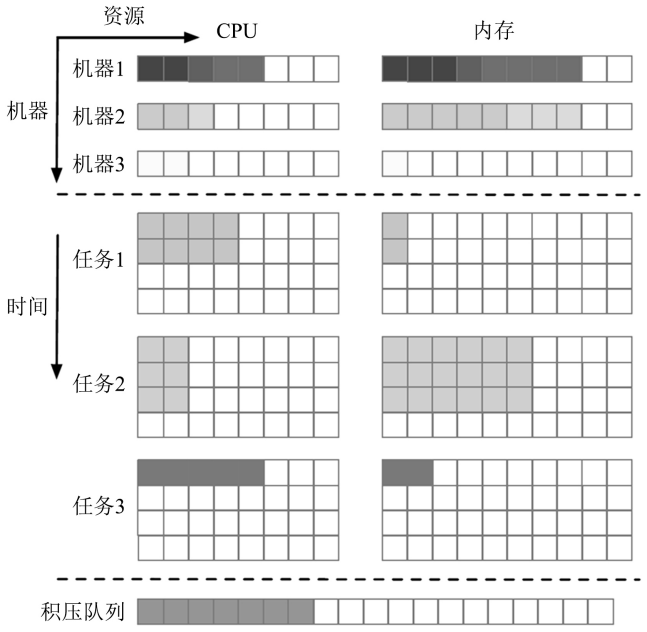


Figure 2. State space design of Deepk8s
图 2. Deepk8s 状态空间设计

4.2. 动作设计

动作是智能体所做的决策，智能体根据当前环境的状态给出对应的动作。本文设计的调度系统中，智能体需要从 k 个待调度任务中选择任意数量任务调度到 m 个机器中的一个，其动作空间大小为 $m \times 2^k$。这个空间大小与任务数量成指数关系，为了减小动作空间的大小，本文忽略系统每次调度的计算开销，即认为在同一时刻可以进行多次调度。每次调度从 n 个待调度任务中选择其中一个进行调度，而在每次调度结束之后可以立即进行下一次调度，这样在同一时刻一样可以进行多次调度动作，采用这种设计方法其动作空间大小变为 $m \times n$，极大减小了动作空间。

除此之外在上述动作中补充动作 ϕ 用来表示在当前调度时刻不做任何调度或者调度决策不合理。当动作为 ϕ 时，仿真环境执行 t 个单位时间之后进入下次调度决策过程。而两次调度的间隔 t 取 3.2 节中几种事件时间的较小值，同时为了控制调度间隔 t 的大小，人为规定 t 的最大值取 16，这样做可以控制两次调度之间的时间间隔不至于太大。

4.3. 奖励函数设计

奖励函数是智能体用于评价动作好坏的标准，因此设计的奖励函数需要与优化目标息息相关。对于单个任务而言，比较关注任务本身的完成时间。本文设计相应的奖励函数来优化任务的平均带权周转时间，任务的带权周转时间为任务完成时间(任务完成时刻 − 任务提交时刻)除以任务的实际执行时间。

因为每个任务的实际执行时间不同，因此他们对于在系统中的等待时间容忍也不同。对于长任务来说由于会占用较久系统资源，因此能够接受较长时间的等待，而对于短任务来说如果等待太长时间会大大降低这类任务的效率。因此考虑使用任务的带权周转时间作为优化目标，可以同时兼顾长任务和短任务。对于优化任务带权周转时间的奖励函数设计如下：

$$r = -\sum_{j \in J} \frac{t}{d_j} \tag{1}$$

式中的 t 为两次调度之间的时间间隔，d_j 为任务 j 的实际执行时间。将所有 t/d_j 加起来得到的就是所有任务的带权周转时间之和。强化学习中通常是最大化奖励的，因此为了最小化任务的带权周转时间在奖励函数中加入一个负号即可。使用该奖励函数可以优化任务的带权周转时间，提高集群的资源利用率。

4.4. 深度强化学习模型结构设计

深度强化学习方法通过智能体和环境的不断交互来优化智能体的策略，深度强化学习模型结构如图 3 所示。智能体从环境中获取当前时刻的系统状态，用向量对系统状态进行表示，将获取到的状态向量输入到智能体的神经网络中，网络输出该状态对应的动作决策到环境中，此时环境状态会发生变化，同时会得到一个关于状态和动作的奖励反馈给智能体，智能体根据奖励可以知道所做决策正确与否，从而不断优化神经网络。

图 3 所示的模型结构中智能体使用的网络结构为经典的 Actor-Critic 框架。Actor-Critic 框架包含 Actor 网络和 Critic 网络，其中 Actor 网络和 Critic 网络的输入都是环境的状态，Actor 网络输出智能体的动作，Critic 网络输出状态的值函数，用于评价动作的好坏。因为 Actor 网络和 Critic 网络都是从状态中提取集群信息，因此两个网络前两层结构相同。实验时将前两层网络的参数进行共享，这样有效减少了模型的参数个数，加速网络的优化收敛，同时其性能基本不会下降。

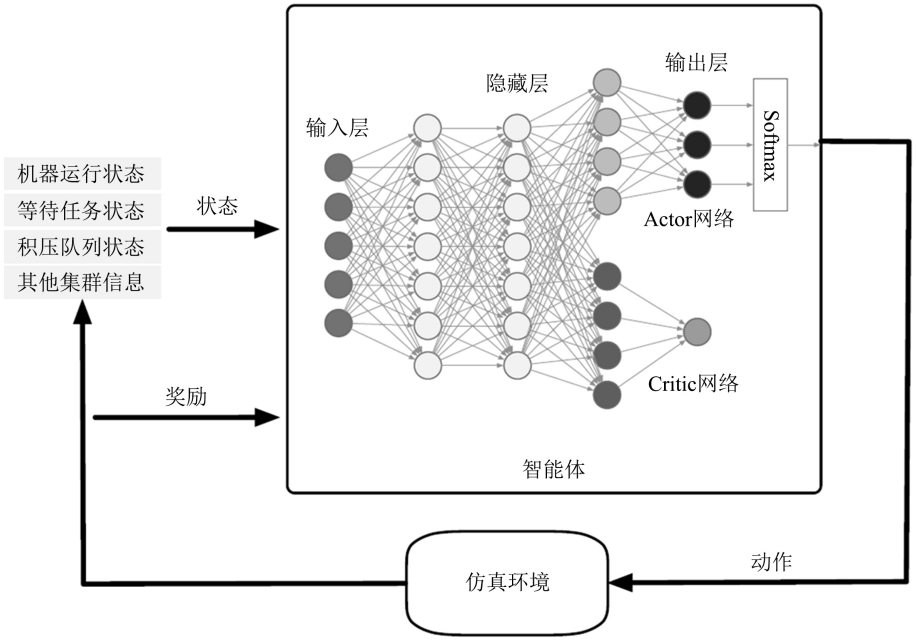


Figure 3. Deep reinforcement learning model structure
图 3. 深度强化学习模型结构

4.5. 算法实现

本文对深度强化学习中的 A3C 算法和 PPO 算法进行实现并对比，然后使用 Dagger 方法实现模仿学习来模仿短任务优先调度算法。

4.5.1. A3C 算法

A3C 算法[9]是众多 Actor-Critic 算法中常用的一种，在 Actor-Critic 框架下，A3C 算法通过将 Actor-Critic 网络放到多个线程中进行同步训练，有效提高了模型的训练速度。同时由于每个线程在训练过程中是独立的，减弱了事件的相关性，使得模型更容易收敛。Actor 网络更新时使用的公式如下：

$$d\theta = \Delta_\theta \log \pi\left(a|s;\theta\right)\left(R - V\left(s;\theta\right)\right) + \beta \Delta_\theta H\left(\pi\left(a|s;\theta\right)\right) \tag{2}$$

其中 $\pi\left(a|s;\theta\right)$ 是智能体的策略，R 是交互过程中的累积奖励，而 $V\left(s;\theta\right)$ 是 Critic 网络的输出，除此之外还加上了策略的熵，通过将策略的熵加入损失函数的方法，一定程度上可以防止策略变为确定性策略，增加所有动作遍历到的可能。Critic 网络是对累积奖励的预测，使用均方误差作为损失函数，因此 Critic 网络梯度计算公式如下：

$$d\theta_v = \frac{\partial\left(R - V\left(s;\theta_v\right)\right)^2}{\partial \theta_v} \tag{3}$$

4.5.2. PPO 算法

深度强化学习方法需要智能体和环境不断进行交互，这个过程耗费很大一部分时间，即使使用 A3C 算法并行进行训练，整体训练效率依然不高效，主要原因是这些算法都是同策略的强化学习方法，用来更新策略的数据必须是当前策略和环境进行交互采集得到，而每次完成参数更新之后的数据不可以再次使用，这导致采集的数据利用效率很低。

PPO 算法[10]使用重要性采样原理，将一次采样得到的数据用来多次更新策略，这让模型可以重复

使用以往采样得到的“经验”，大大提升了数据利用率。同时 PPO 算法为了控制每次参数更新的幅度，使用裁剪的方法来控制新旧策略的差异，使策略可以更稳定进行更新。

PPO 算法同样是基于 Actor-Critic 框架的，只是在 Actor 网络的更新上较 A3C 算法有所不同。PPO 算法因使用了重要性采样原理，使得其可以使用一次采样得到的数据进行多次策略参数更新，其 Actor 网络更新的目标函数如下：

$$J=\sum \min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta'}(a|s)},\text{clip}\left(\frac{\pi_\theta(a|s)}{\pi_{\theta'}(a|s)},1-\epsilon,1+\epsilon\right)\right)A^{\pi_{\theta'}(s,a)} \tag{4}$$

4.5.3. 模仿学习

传统深度强化学习方法是通过计算累积奖励来评估策略好坏，然后对策略进行更新以达到预期优化目的。模仿学习则是通过模仿专家做决策的方法来学习很好的决策策略。因为模仿学习不再需要累积奖励来判断动作优劣，因此也就不需要 Critic 网络来预测状态的值函数。而 Actor 网络作为输出策略的网络，只需要对 Actor 网络进行适当修改即可，其更新公式如下：

$$J=\sum \hat{A}\log\pi(a|s;\theta)+\beta H(\pi(a|s;\theta)) \tag{5}$$

其中 $\hat{A}$ 为专家给出的状态 s 下各个动作的概率，相当于专家对状态 s 下各动作的评价。为了应对模仿学习中复合误差问题，使用 Dagger [11]的方式来解决。在智能体和环境进行交互时，根据智能体的策略 $\pi(a|s;\theta)$ 来得到状态 s 下模型执行的动作 a，同时根据专家 $E(s)$ 得到专家在状态 s 下的动作 $\hat{a}=E(s)$。在交互中使用根据模型策略得到的动作 a，同时将状态 s 和专家动作 $\hat{a}$ 作为状态－动作对存储到经验池中。训练时从经验池中随机采样部分样本进行 Actor 网络的更新。

5. 实验

为了验证深度强化学习算法在资源调度问题中的性能，通过仿真实验的手段来分析调度结果，并与以下四种启发式调度算法进行性能比较：先来先服务调度算法、随机调度算法、短任务优先调度算法和 Tetris 调度算法。先来先服务调度算法只需要根据任务达到时间先后顺序维护一个队列，按照队列顺序进行任务调度。随机调度算法则从系统中的待调度任务中随机选择任务进行调度。短任务优先调度算法优先对短任务进行调度，增加一段时间内任务完成数量。Tetris 调度算法融合了公平调度和短任务优先调度，同时考虑资源打包效率，通过一定权重将三者结合起来进行调度决策。

5.1. 实验设置

本文使用 Python 进行仿真环境编写，建模的机器数量为 5 台，每台的 CPU 资源数量为 10 个，内存资源数量为 64 个，待调度任务数量上限设置为 10 个。除此之外模型训练相关参数设置为学习率为 0.001，网络中可学习的参数使用随机梯度下降法进行优化。网络框架使用 TensorFlow 进行实现。

5.2. 评价指标

5.2.1. 任务完成时间

任务从提交到系统到顺利执行完成所耗费的时间为任务的完成时间，即任务在系统中等待调度的时间加上任务实际在系统中执行的时间。

5.2.2. 任务带权周转时间

任务带权周转时间等于任务的完成时间除以任务的实际执行时间，使用任务带权周转时间作为评价

指标更有利于短任务和长任务的公平。

5.2.3. 任务完工时间

当一系列任务执行结束之后，从第一个任务到达集群到最后一个任务完成的间隔是任务的完工时间，该指标更关注任务整体。

5.3. 实验结果

在 Deepk8s 离线仿真环境中使用相同的任务序列对各种资源调度算法进行测试，各个算法的任务带权周转时间如图 4 所示，其中深度强化学习的结果展示的是 PPO 算法的结果，PPO 算法和 A3C 算法仅在训练速度上有差异，结果上几乎没有差异。图中纵轴为任务的带权周转时间，横轴为五种资源调度算法。其中性能最差的为先来先服务调度算法(FCFS)，因为先来先服务调度算法在调度一个长任务之后可能在很长一段时间没有空闲资源释放，因此导致大批任务阻塞。随机调度算法(Random)能够取到比先来先服务调度算法更好的效果的原因是一定程度上打乱了任务的调度顺序。Tetris 调度算法[12]和短任务优先调度算法(SJF)因为优先对短任务进行调度，从任务带权周转时间的计算公式可以知道使用这种策略可以使任务的平均带权周转时间更小。深度强化学习调度算法因为学习了任务到达的规律，所以能够取得更好效果。

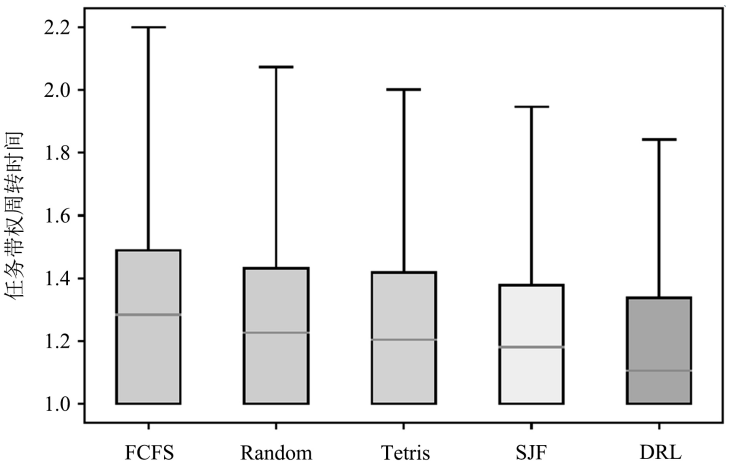
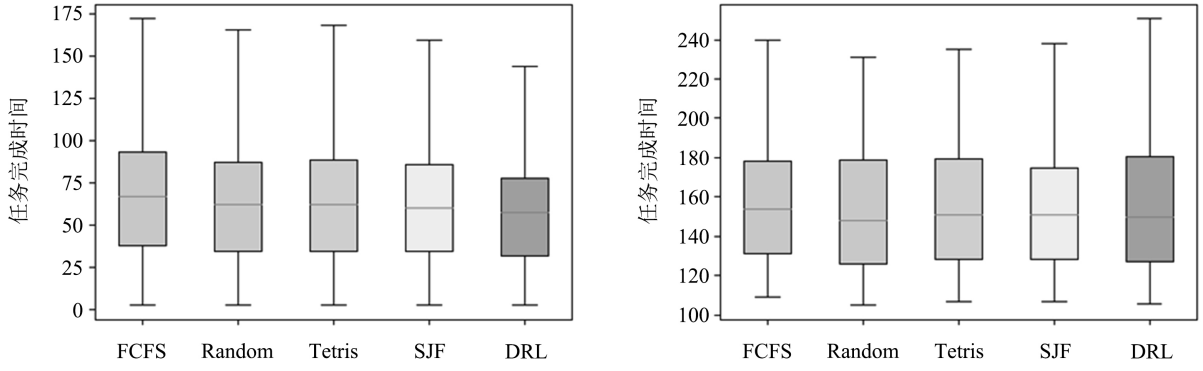


Figure 4. Slowdown of tasks
图 4. 任务带权周转时间

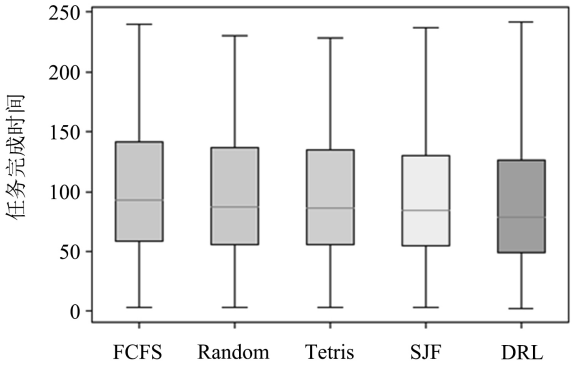
图 5 为任务的平均完成时间。从图 5 中图 5(c)可以看到深度强化学习方法在任务完成时间上相较于其他启发式调度算法优势并不那么明显了，但是整体优势依然存在。图 5(a)和图 5(b)分别展示了短任务和长任务的任务完成时间，对比短任务和长任务的结果可以看出来，深度强化学习算法优先对短任务进行调度，短任务的任务完成时间较其他四种启发式算法低很多，这势必也导致长任务的调度存在一定延时，因此长任务的完成时间反而上升了。

为了更清晰展示实验结果，将所有结果总结到表 1 中。基于深度强化学习的资源调度算法可以将任务的平均带权周转时间降低到 1.325，相较于其他启发式调度算法至少降低了 7.5%，最多降低了 13.5%，在任务平均完成时间上深度强化学习算法同样取得了 94.116 的最优效果，相较于短任务优先调度算法降低了约 0.2%。同时任务完工时间指标各个算法的差别不大，表明深度强化学习算法在降低任务平均带权周转时间的同时，并没有明显延长任务整体的进度。



(a) 执行时间小于100的任务

(b) 执行时间大于100的任务



(c) 所有任务

Figure 5. Average completion time of tasks
图 5. 任务平均完成时间

Table 1. Test results
表 1. 测试结果

算法	带权周转时间	任务完成时间	完工时间
先来先服务	1.531	99.210	680
随机调度	1.463	94.726	645
Tetris 算法	1.470	95.366	**639**
短任务优先	1.433	94.324	646
深度强化学习	**1.325**	**94.116**	643
模仿学习	1.435	94.331	646

图 6 展示了不同深度强化学习方法模型的训练收敛速度。横轴为迭代的次数，纵轴为模型获得的累积奖励。图中横线为启发式算法中表现最优的短任务优先调度算法的累积奖励值，可以看到 PPO 算法和 A3C 算法在收敛之后效果差不多，而 PPO 算法在大概 1000 次迭代之后达到收敛，A3C 算法则相对慢很多。收敛速度最快的是模仿学习，在仅仅迭代约 500 次时就收敛了，但是其收敛结果只和短任务优先算法相同，主要原因是模仿的专家是短任务优先算法，其效果并不是最优的。

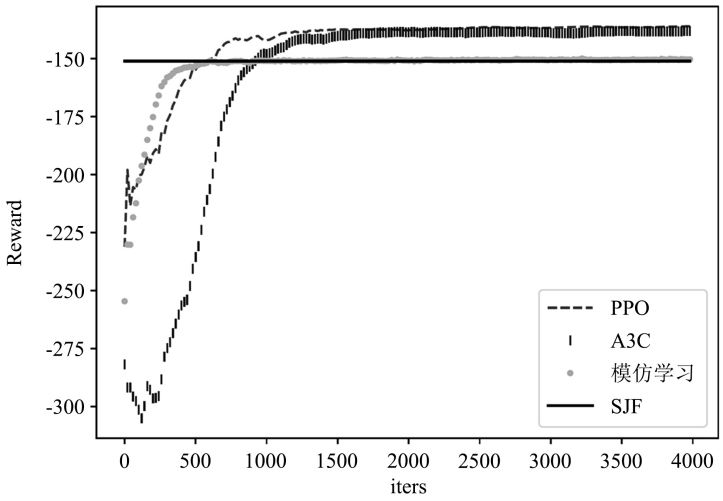


Figure 6. Cumulative rewards of different deep reinforcement learning algorithm models
图 6. 不同深度强化学习算法模型训练中的累积奖励

以上实验结果可以看出，深度强化学习可以应用于资源调度问题中。同时经过精心设计奖励函数，可以达到对相应优化目标的优化。

6. 结语

为了解决云数据中心资源调度问题，提高资源利用率，本文提出一种基于深度强化学习的模型，通过自主学习来得到调度策略。本文设计了 Deepk8s 离线仿真环境来对资源调度过程进行仿真，然后对资源调度过程进行深度强化学习建模，设计其状态、动作和奖励函数，使用阿里巴巴真实集群数据进行仿真实验，对比经典启发式算法中的先来先服务调度算法、随机调度算法、短任务优先调度算法和 Tetris 调度算法，深度强化学习算法在不明显延后任务整体完工时间的基础上，可以大幅降低任务的平均带权周转时间，为启发式资源调度方法提供一种很好的替代方法。

致　谢

感谢导师对本文工作的指导，以及感谢实验室同学的热心帮助。感谢国家重点研发计划项目 2018YFB1003703。

基金项目

国家重点研发计划项目 2018YFB1003703。

参考文献

[1] 凤凰网科技. 字节跳动在美租用数据中心: 数十万台服务器, 能耗达 53 兆瓦[EB/OL]. https://tech.ifeng.com/c/80zDZm54HDc, 2020-10-30.

[2] 叶宇飞. 基于深度强化学习的数据中心资源算法调度研究[D]: [硕士学位论文]. 成都: 电子科技大学, 2019.

[3] Ferguson, A.D., Bodik, P., Kandula, S., *et al*. (2012) Jockey: Guaranteed Job Latency in Data Parallel Clusters. *Proceedings of the 7th ACM European Conference on Computer Systems*, Bern, 10-13 April 2012, 99-112. https://doi.org/10.1145/2168836.2168847

[4] Martinez, J.F. and Ipek, E. (2009) Dynamic Multicore Resource Management: A Machine Learning Approach. *IEEE Micro*, **29**, 8-17. https://doi.org/10.1109/MM.2009.77

[5] Mao, H., Alizadeh, M., Menache, I., *et al.* (2016) Resource Management with Deep Reinforcement Learning. *Proceedings of the* 15*th ACM Workshop on Hot Topics in Networks*, Atlanta, GA, 9-10 November 2016, 50-56. https://doi.org/10.1145/3005745.3005750

[6] Mao, H., Schwarzkopf, M., Venkatakrishnan, S.B., *et al*. (2019) Learning Scheduling Algorithms for Data Processing Clusters. *Proceedings of the ACM Special Interest Group on Data Communication*, Beijing, 19-23 August 2019, 270-288. https://doi.org/10.1145/3341302.3342080

[7] LeCun, Y., Bottou, L., Bengio, Y., *et al.* (1998) Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE*, **86**, 2278-2324. https://doi.org/10.1109/5.726791

[8] Liu, N., Li, Z., Xu, J., *et al.* (2017) A Hierarchical Framework of Cloud Resource Allocation and Power Management Using Deep Reinforcement Learning. 2017 *IEEE* 37*th International Conference on Distributed Computing Systems* (*ICDCS*), Atlanta, 5-8 June 2017, 372-382. https://doi.org/10.1109/ICDCS.2017.123

[9] Mnih, V., Badia, A.P., Mirza, M., *et al*. (2016) Asynchronous Methods for Deep Reinforcement Learning. *International Conference on Machine Learning*, *PMLR*, New York, 20-22 Jun 2016, 1928-1937.

[10] Schulman, J., Wolski, F., Dhariwal, P., *et al.* (2017) Proximal Policy Optimization Algorithms. arXiv:1707.06347 [cs.LG]

[11] Ross, S., Gordon, G. and Bagnell, D. (2011) A Reduction of Imitation Learning and Structured Prediction To No-Regret Online Learning. *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics. JMLR Workshop and Conference Proceedings*, Fort Lauderdale, 11-13 April 2011, 627-635.

[12] Grandl, R., Ananthanarayanan, G., Kandula, S., *et al.* (2014) Multi-Resource Packing for Cluster Schedulers. *ACM SIGCOMM Computer Communication Review*, **44**, 455-466. https://doi.org/10.1145/2740070.2626334