

一种新的以服务质量为导向的 Spark 作业调度器

何玉林^{1,2}, 莫沛恒^{1,2}, Philippe Fournier-Viger², 黄哲学^{1,2}

1. 人工智能与数字经济广东省实验室(深圳), 广东 深圳 518107;
2. 深圳大学计算机与软件学院, 广东 深圳 518060

摘要

Spark 大数据计算框架被广泛用于处理和分析爆发式增长的大数据。云端能够提供按需和按量付费的计算资源来满足用户的请求。当前,许多组织将大数据计算集群部署在云端上开展大数据计算任务,其需要高效地处理 Spark 作业调度问题以满足各种用户对 QoS 的要求,如降低使用资源的花费和缩短作业的响应时间。而现有研究大多未能统一考虑多用户要求,忽略了 Spark 集群环境和工作负载的特性,导致资源浪费和用户对 QoS 的要求得不到满足等。为此,通过对部署在云端的 Spark 集群作业调度问题进行建模,设计了一种新的基于 DRL 技术的 Spark 作业调度器来满足多个 QoS 要求。搭建了 DRL 集群仿真环境,用于对作业调度器的核心 DRL Agent 进行训练。在调度环境中实现了基于绝对深度 Q 值网络、基于近端策略优化与广义优势估计联合的训练方法,使 DRL Agent 可以自适应地学习不同类型作业,以及动态、突发的集群环境特征,实现对 Spark 作业的合理调度,以降低集群总使用成本、缩短作业的平均响应时间。在基准套件上对 DRL Agent 测试的结果表明,与其他现有的 Spark 作业调度解决方案相比,本文设计的 DRL Agent 作业调度器在集群总使用成本、作业平均响应时间以及 QoS 达成率上具有显著的优越性,证明了其有效性。

关键词

大数据计算; 服务质量; Spark 作业调度器; 云环境; 深度强化学习

中图分类号: TP391.9

文献标志码: A

doi: 10.11959/j.issn.2096-0271.2025048

A novel quality of service-oriented Spark job scheduler

HE Yulin^{1,2}, MO Peiheng^{1,2}, Fournier-Viger Philippe², HUANG Zhexue^{1,2}

1. Guangdong Laboratory of Artificial Intelligence and Digital Economy (SZ), Shenzhen 518107, China
2. College of Computer Science & Software Engineering, Shenzhen University, Shenzhen 518060

Abstract

Spark is a widely-used big data computing framework to process and analyze the explosive-growing data. The cloud can provide on-demand and pay-as-you-go computing resources to satisfy the users' requirements. Currently, many organizations have deployed big data computing clusters on the cloud. These clusters are required to efficiently handle the Spark job scheduling problem so as to meet the QoS requirements of various users, such as reducing the cost of resource usage and shortening the job response time. However, most of the existing methods don't consider the requirements of multiple users together, and fail to take into account the characteristics of Spark cluster environments

and workloads. To address the above-mentioned challenge, a new Spark job scheduler based on DRL technology was designed to adapt to multiple QoS requirements by modeling the job scheduling problem of Spark clusters deployed in the cloud. A DRL cluster simulation environment was built to train the core DRL Agent of job scheduler. In the scheduling environment, training methods based on absolute deep Q-network and a combination of proximal policy optimization and generalized advantage estimation were implemented, enabling DRL agent to adaptively learn the characteristics of different types of jobs as well as the characteristics of dynamic and bursty cluster environments. This enables rational scheduling of Spark jobs to reduce the total usage cost of the cluster and shorten the average response time of jobs. Testing results of DRL Agent on the benchmark suite show that compared with other existing Spark job scheduling solutions, the newly designed DRL Agent job scheduler in this paper has significant advantages in terms of total cluster usage cost, average job response time and QoS achievement rate, which confirming the feasibility and effectiveness of the job scheduler designed in this paper.

Key words

big data computing, quality of service, Spark job scheduler, cloud environment, deep reinforcement learning

0 引言

随着现代通信技术和互联网的快速发展,行业数据呈现爆发式增长的趋势。在应对大数据处理任务时,大数据计算框架如 MapReduce^[1]和 Spark^[2]被广泛使用。这些框架可以部署在本地大数据计算集群中,也可以部署在公有云上。近年来,云端提供的服务在不断发展,越来越多的公司使用云端提供的服务来满足其不断增长的计算资源需求^[3]。Amazon Web Services、Google Cloud Platform、Microsoft Azure 和 Alibaba Cloud 等云服务提供商提供按需、按量付费的计算资源来满足众多用户的需求。此外,云端资源比本地物理资源更易于管理和部署,许多企业正朝着在云上部署大数据分析集群的方向发展^[4]。服务水平协议(service level agreement, SLA)是用户和服务提供者之间商定的服务条款,包括用户的各种服务质量(quality of service, QoS)要求。Spark 是一种通用、快速的大规模数据计算框架^[5],因其基于内存计算的

快速性和可扩展性在工业界和学术界得到广泛应用^[6]。Spark 作为一个通用框架,支持多种计算方式,如批处理、交互式、迭代和流式计算^[7],其对于具有不同需求的用户来说是非常有用的。Spark 使用了弹性分布式数据集(resilient distributed dataset, RDD)^[8]的数据结构,可以通过内存来存储中间结果,从而大大加快数据的处理速度^[9]。

Spark 有不同层次的调度策略,包括应用程序之间的调度和应用程序内部的调度。Spark 的应用程序是用户提交的作业,本文重点关注应用程序之间的调度。默认的 Spark 调度程序提供了两种在 Spark 集群中创建 Executor (Spark 应用中的计算节点,负责执行具体的任务)的方法: Round Robin 和 Round Robin Consolidate。这两种方法可以均衡地使用 Spark 集群资源,避免了共存 Executor 之间的干扰,提高了工作负载的性能。然而,Spark 集群中的作业到达是随机的,其意味着下一个到达 Spark 集群的作业类型和作业大小是未知的。因此算法必须在没有任何先验知识的情况下调度 Spark 作业,以降低集群的总使用成本,并满足作业的

响应时间要求。使用可靠的数学方程对具有复杂状态的 Spark 集群资源进行动态优化建模通常是非常困难的。同时,对复杂的 Spark 集群进行精确调度也非常耗时。随着深度强化学习 (deep reinforcement learning, DRL) 的发展,现有研究已经证明 DRL 可以在没有先验知识的情况下对复杂问题进行优化。DRL 的核心是训练与环境交互的主体 (Agent),并根据环境提供的奖励自主学习经验。DRL 的学习过程是马尔可夫决策过程,其中,Agent 根据其状态采取不同的操作,并因响应好或坏的操作而获得正向或负向奖励。Agent 的目标是学习最佳策略,以在不同状态下选择最佳操作。Agent 在与环境交互的过程中不断尝试获取经验,从而学习到最优策略。为此,本文提出了一种智能作业调度程序,其可以学习实时数据,并针对事务工作负载进行优化决策。

目前,已有研究者对 Spark 集群中的任务调度问题进行研究^[10-20],但已有研究只对单个 QoS 要求进行优化,不能同时满足多个 QoS 要求。对于 Spark 集群中的大数据计算作业调度问题而言,重要的 QoS 要求是降低使用资源的花费和缩短作业的响应时间。然而,当 Spark 集群部署在云端时,使用的计算资源由于工作负载的变化,很难在降低资源使用成本的同时,以不影响用户作业响应时间限制的方式调度作业。此外,当具有不同作业类型的作业同时在集群中运行时,Spark 默认调度器无法捕获广泛的集群资源和工作负载的动态变化特性,不能有效地实现调度目标。

为解决上述 Spark 作业调度过程中存在的问题,本文提出一种新的基于 DRL 的 Spark 调度器,以同时满足多个 QoS 要求,并降低资源使用成本,满足作业响应时间要求。Spark 调度器的主体能够充分学习不断

变化的环境特征,做出更好的调度决策。本文在调度环境中实现了基于绝对深度 Q 值网络 (categorical deep Q network, C-DQN)^[21] 的 Agent、基于近端策略优化 (proximal policy optimization, PPO)^[22] 与广义优势估计 (generalized advantage estimation, GAE) 联合 (简称 PPO+GAE) 的 Agent。本文的主要工作是将 C-DQN 和 PPO+GAE 应用于 Spark 作业调度中,并将 C-DQN、PPO+GAE 与 Spark 作业调度环境相结合。在处理 Spark 作业调度问题的过程中,人们需要将环境的特征转换成模型可用的状态表示,并将 Agent 的行为映射到具体的任务分配和资源利用中,以获得环境的奖励。本文将 C-DQN、PPO+GAE 与 Spark 作业调度环境相结合,通过训练可以有效感知环境的 Agent,使 C-DQN、PPO+GAE 能够很好地解决 Spark 作业调度问题。本文使用真实的工作负载跟踪对所提出的基于 DRL 的 Spark 作业调度器的可行性和有效性进行详细的验证,实验结果表明,与其他现有解决方案相比,本文提出的基于 DRL Agent 的作业调度器在成本优化和降低响应时间方面具有显著的优越性。

1 相关工作

1.1 基于 QoS 要求优化的作业调度

基于 QoS 要求优化的作业调度代表性研究成果总结如下。Denninnart 等^[23]旨在通过修剪较低可能满足截止日期的作业来提高系统鲁棒性和整体服务质量,开发了一种新的任务修剪机制,该机制可以添加到现有的作业映射启发式方法中且不改变它们。Syed 等^[24]为了实现成本最小化的主要目标,将一种新的遗传算法模型混合火烈鸟搜索的遗传

算法,更好地实现任务调度。Zhu等^[25]提出了一个新的调度和执行框架 TOPOSCH,以确定分布式的长期运行应用程序的QoS优先级,同时平衡批处理作业的性能,并通过收集空闲资源来保持较高的集群利用率。TOPOSCH跨微服务跟踪每个用户请求的占用空间,并根据端到端延迟图,使用关键路径分析来识别具有高QoS违规风险的微服务。Neetu等^[26]提出了一种基于蚁群优化算法和神经网络的多目标任务调度方法。蚁群优化算法的优势是其能够提高任务处理效率、缩短平均访问时间,并实现更均衡的作业分配。Taghinezhad-niar等^[27]提出了两种成本和能源感知的作业流调度,它们考虑了任务执行时间的不确定性。这两种解决方案都利用容器来提高资源利用率,降低能耗、资源成本,减少作业流截止日期违规,以满足相应的QoS要求。上述研究未关注Spark作业的特性,未考虑Spark作业调度的QoS需求,同时未使用DRL方法来解决Spark作业调度问题。

1.2 基于启发式和元启发式的Spark作业调度

近年来,通过优化不同QoS目标来构建Spark作业调度器的工作相继被提出。Fu等^[28]提出了一种使用二分图建模的本地感知任务调度算法,以解决Spark中map和reduce任务生成的跨节点/机架数据传输引起的性能下降问题。该算法通过考虑全局最优性,可以生成调度map任务的最优解,并减少具有数据局部性特征的任务,从而提高Spark作业调度的性能。Li等^[29]提出基于动态电压与频率调节的频率感知和节能策略,面向运行在YARN上的Spark工作,提升处理能效,设计实施YARN、Spark两层节能YARN层依据最小能效比求最佳CPU频率,Spark层构建

调度法动态调节CPU频率。刘汪根等^[30]通过数据拓扑感知能力确保了计算与数据的局部性,节省了网络I/O开销;优化了针对点对点大数据读取的资源调度,有效降低了网络风暴带来的影响;采用网络和磁盘隔离技术以及可抢占策略,保证服务等级协议的实现。Li等^[31]提出了一种基于作业分类的Spark作业调度策略,以解决原生Spark作业调度算法中的资源浪费问题。该策略通过对不同的作业进行分类来更高效地调度资源,缩短作业完成时间。Liu等^[32]通过预测实时变化的资源需求,动态调整Spark作业Executor的大小,从而有效提高集群利用率。Islam等^[33]提出了一种用于在期限约束下降低总集群使用成本的作业调度算法,该算法能够有效降低混合云环境下Spark集群的总使用成本。Zade等^[34]设计了一种基于模糊改进红狐优化和博弈论的高效任务调度算法。Zhang等^[35]考虑云环境中任务调度的不确定性,设计了区间多目标调度优化模型来模拟真实云环境中的任务调度,并采用区间多目标优化算法调度云环境中的任务。大多数基于元启发式和启发式的Spark作业调度程序仅针对一个QoS要求进行优化,没有考虑在多个QoS要求的情况下进行Spark作业的调度。基于启发式的方法构建的模型严重依赖过去的的数据,而这些数据有时会由于集群环境的各种变化而失效,难以结合工作负载和集群的变化调整或修改基于启发式的作业调度方法。因此,学术界和工业界逐渐关注设计基于DRL的Spark作业调度方法,以更加有效和可扩展地处理Spark作业调度问题。

1.3 基于DRL的Spark作业调度

尽管DRL在Spark作业调度中的应用

相对较新，但是使用 DRL 来解决 Spark 作业调度问题已成为该领域研究的重点。Islam 等^[36]使用 DRL 解决部署在公有云上的 Spark 集群中作业的调度问题。他们实现了基于深度 Q 学习的作业调度器、基于 REINFORCE 的作业调度器两个基于 DRL 的作业调度器，以缩短 Spark 集群中作业的平均执行时间、降低集群的总使用成本。Guo 等^[37]提出了一种名为 Deep RM Plus (deep resource management plus) 的解决方案来处理云资源的调度问题。Deep RM Plus 使用卷积神经网络捕获环境特征，并使用模仿学习来减少强化学习 Agent 的训练时间。Deep RM Plus 可以缩短作业的平均加权周转时间和平均周期时间。Mao 等^[38]提出了一种基于具有策略梯度的强化学习调度算法，其使用强化学习和基于改造的有向无环图 (directed acyclic graph, DAG) 来构建特殊的神经网络，以学习特定于工作负载的调度算法，并以此来自动捕获作业的特征，从而缩短作业的完成时间。Zhao 等^[39]提出了一种基于 DRL 的智能云任务调度器，使最优调度决策可以直接从经验中学习，而不需要任何先验知识。该调度器将任务调度建模为具有约束的动态优化问题，然后采用深度确定性策略梯度网络来找到最佳任务分配解决方案，同时满足性能要求和成本约束。Mohammad 等^[40]提出了一种称为并行强化学习的调度算法。该算法分为两个阶段：第一阶段使用元调度器将任务分配给主机；第二阶段使用强化学习来指导 Agent 的活动，并使用并行策略帮助 Agent 在尽可能短的时间内搜索最优的问题解决方案。该算法在云环境下的任务调度问题中，有效缩短了任务等待时间，降低了能耗，提高了资源利用率，增加了任务执行的安全性。Pradhan 等^[41]针对复杂、高维的问题，尝

试将 DRL 与并行粒子群优化相结合，以加快运算速度，缩短训练神经网络的处理时间。他们提出了一种有效的调度算法——深度强化学习并行粒子群优化算法，旨在解决负载均衡问题，并进行各种参数的优化，具有更高的问题求解精度。

综上所述，现有研究大多针对单一的 QoS 要求进行优化，且没有考虑 Spark 集群中环境的复杂性和 Spark 作业调度中 Executor 细粒度级别的放置。真实 Spark 集群的环境通常很复杂，对其建模很困难。使用 DRL 方法可以有效地在复杂环境中解决困难的 Spark 作业调度问题。经过训练的 DRL Agent 可以将同一作业的 Executor 放置在不同的虚拟机 (virtual machine, VM) 中，并保证在所需资源上启动作业的所有 Executor，还可以通过定价模型处理不同大小的作业 Executor 和不同大小的 VM 实例。本文提出的 DRL Agent 可以学习作业的固有特性和环境的变化特征，以找到适当的放置策略，进一步对 Spark 作业调度方案进行改进。

2 问题表述

在 Spark 集群中，用户可以提交一个或多个作业，并为其指定资源需求，包括所需的 CPU 核数、内存大小、预期使用成本和最晚完成时间等。作业可以是任何类型，并且可以随时到达，即作业的到达是随机的，无法预测下一步会到达什么类型的作业以及所需的资源量。调度器根据 Spark 调度算法处理每个作业，并选择合适的作业节点来创建 Executor。创建 Executor 时，调度器必须满足集群的资源约束。调度器的目标是减少整个集群的总体使用成本和缩短作业的响应时间。**表 1** 给出了建

模 Spark 作业调度问题所需的数学符号定义。

在 Spark 集群中, 每个 VM 可以拥有任意的资源容量 (即 CPU 和内存)。当作业被提交到集群时, 调度器可以在一个或多个 VM 中创建 Executor, 并且必须遵循 VM 的资源容量限制和当前作业的资源需求限制。用户在两个维度上提交 Executor 的资源需求: CPU 内核和内存。因此, 可以将每个作业的 Executor 视为多维框, 在调度过程中需要将其放置到特定的 VM 中。CPU 和内存的资源需求和容量约束定义如下。

$$\sum_{j \in [1, M]} e_{\text{cpu}}^{ij} \times n \leq \text{vm}_{\text{cpu}}^i \quad \forall i \in [1, N] \quad (1)$$

$$\sum_{j \in [1, M]} e_{\text{mem}}^{ij} \times n \leq \text{vm}_{\text{mem}}^i \quad \forall i \in [1, N] \quad (2)$$

调度结束时集群的总使用成本定义如下。

$$\text{Cost}_{\text{total}} = \sum_{i \in [1, N]} \text{vm}_p^i \times \text{vm}_T^i \quad (3)$$

作业的平均响应时间 (average response time, ART) 定义如下。

$$\text{ART} = \sum_{j \in [1, M]} (\text{job}_{\text{fin}}^j - \text{job}_{\text{arr}}^j) / M \quad (4)$$

得到多 QoS 要求的优化目标如下。

$$\alpha \times \text{Cost}_{\text{total}} + (1 - \alpha) \times \text{ART} \quad (5)$$

$$\text{QoS}^j = \begin{cases} 1, & \text{if } \text{Cost}_{\text{total}}^j < \text{Cost}_{\text{tar}}^j \text{ and } \text{job}_{\text{fin}}^j < \text{job}_{\text{ddl}}^j \\ 0, & \text{其他} \end{cases} \quad \forall j \in [1, M] \quad (6)$$

$$\text{Cost}_{\text{total}}^j = \text{Cost}_{\text{total}} / M \quad (7)$$

本文通过统计满足 QoS 要求的作业个数, 将其与总作业数进行比较, 得出两者之间的比值, 并作为 QoS 达成率。如果作业在期望的 QoS 要求内完成, 则认为作业调度是满足用户要求的。此外, 为了提高

表1 符号定义

符号	含义
N	云环境下的 VM 数量
M	提交到 Spark 集群的作业总数
n	当前 VM 上放置的执行器数量
e_{cpu}^{ij}	第 j 个作业在第 i 台 VM 上放置执行器所需的 CPU 核数
e_{mem}^{ij}	第 j 个作业在第 i 个 VM 上放置执行器所需的内存大小
vm_{cpu}^i	第 i 个 VM 的 CPU 核数
vm_{mem}^i	第 i 个 VM 的内存大小
vm_p^i	第 i 个 VM 的单价
vm_T^i	第 i 个 VM 的使用时间
$\text{job}_{\text{arr}}^j$	第 j 个作业的到达时间
$\text{job}_{\text{fin}}^j$	第 j 个作业的完成时间
$\text{job}_{T_{\text{max}}}^j$	第 j 个作业的最大执行时间
$\text{job}_{T_{\text{min}}}^j$	第 j 个作业的最小执行时间
$\text{Cost}_{\text{tar}}^j$	第 j 个作业的预期使用成本
$\text{job}_{\text{ddl}}^j$	第 j 个作业的期限
QoS^j	第 j 个作业是否完成 QoS 要求

其中, $\alpha \in [0, 1]$, 可以通过设置 α 值来调整对不同 QoS 目标的关注, 以应对不同的调度场景。当 α 的值设置为 1 时, 调度器更关心集群的总使用成本; 当 α 的值设置为 0 时, 调度器更关心集群的平均响应时间; 当 α 的值设置为 0.5 时, 调度器同时关心集群的总使用成本和平均响应时间。当满足以下的条件时, 则说明 QoS 要求被满足。

作业的 QoS 达成率, 本文在降低总体使用成本的情况下, 还需尽量缩短作业的响应时间。

上述优化问题是一个非确定性多项式硬问题。为解决此问题, 调度程序需要在做出任何调度决策之前都先了解作业的完

成时间。这使调度程序的设计极其困难,因为不同的作业配置文件和系统参数对作业调度的影响极大。此外,如果作业数、Executor 数和总集群大小不断变化,以最优方式解决该问题可能变得不可行。尽管基于启发式的算法具有很高的可扩展性,但它们不能泛化多个目标,也不能捕捉 Spark 集群和工作负载的固有特征改进优化目标。因此,本文使用 DRL 技术来解决这个难题。本文的工作重点是应用程序间调度,其中,调度程序的目标是在适当的 VM 中分配资源,以最小化集群的总体使用成本、缩短作业响应时间。在处理应用程序间的调度决策时,本文通过考虑作业完成时间的变化,从更高的层次解决数据依赖性和环境变化引起的调度性能问题。

3 基于 DRL 的 Spark 作业调度器

3.1 DRL 作业调度器框架

图 1 所示为基于 DRL 的作业调度器框架,其由 DRL Agent 和 Spark 集群环境组

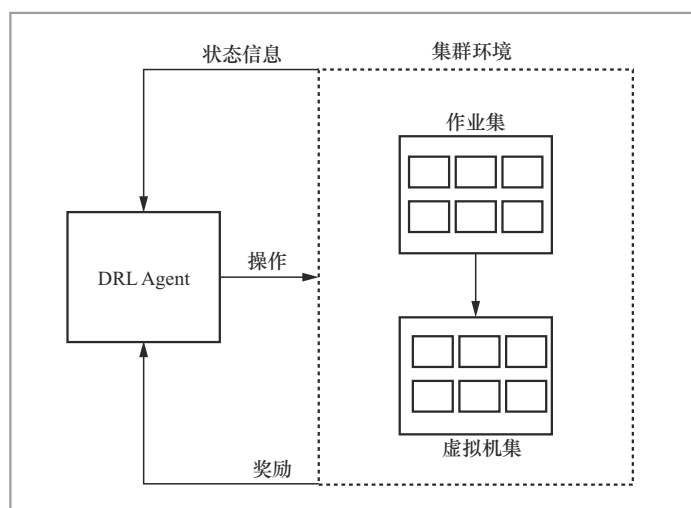


图1 基于DRL的作业调度器框架

成。基于DRL的Agent通过观察环境的状态放置所提交作业的Executor,放置后环境向Agent反馈下一个时间步的状态和当前时间步的奖励。

(1) Agent

Agent充当调度程序,负责调度集群中的作业。在一般的DRL框架中,Agent通过感知环境状态来选择执行的动作,并从环境中接收相应的奖励和下一个可观察的状态。Agent通过观察Spark集群中每个VM的状态和作业到达信息来调度任务,并根据环境反馈不断优化决策。

(2) 环境

环境由Spark集群中的各个VM和提交到集群的作业组成,其将当前状态传递给Agent,并在Agent执行操作后,反馈下一个时间步的状态奖励给Agent。

(3) 状态空间

状态信息包括Agent感知的环境信息以及Agent自身动作带来的变化。状态由Spark集群中所有虚拟机的CPU资源、内存资源以及当前提交到集群的作业信息组成。Agent接收的状态信息可以表示为一个一维向量 $[vm_{cpu}^1, vm_{mem}^1, vm_{cpu}^2, vm_{mem}^2, \dots, vm_{cpu}^N, vm_{mem}^N, job_{id}, job_{tar}, job_{ddl}, e_{cpu}, e_{mem}, e_{num}]$ 。其中, $vm_{cpu}^1, vm_{mem}^1, \dots, vm_{cpu}^N, vm_{mem}^N$ 表示Spark集群环境中所有N台VM的CPU和内存资源, $job_{id}, job_{ddl}, e_{cpu}, e_{mem}, e_{num}$ 表示当前时间点提交到Spark集群的作业ID,作业的期限,每个Executor所需的CPU、内存资源,以及Executor的数量。因此,状态空间的生长仅取决于集群的大小,而不受作业总数的影响。

(4) Episode

一个Episode指Agent与环境的一次完整交互过程,其从第一个作业到达Spark集群开始调度,直到所有作业都被调度完成。在这个过程中,Agent会

与环境交互并获取奖励信息。如果 Agent 执行了负面操作, Episode 可能会提前终止, 环境只在提交作业时向 Agent 发送状态信息。Agent 接收环境信息后, 执行 Executor 的放置动作或者等待放置动作。Agent 执行动作后, 环境会更新状态, 并将更新后的状态信息发送给 Agent。

(5) 动作空间

动作空间是 Agent 可以执行的所有动作。其由等待动作和可以放置的 VM 数量组成。当 Spark 集群中没有足够的资源来放置当前作业的 Executor 时, Agent 会选择等待先前调度的作业执行完成。此外, 为了优化特定目标, Agent 有时不会立即将作业调度到 Spark 集群中, 而是等待某些特定的 VM 变为空闲状态后再进行调度。因此, 在一个包含 N 个 VM 的集群中, 动作空间包括 $N+1$ 种可能的离散动作。其中, Action0 表示 Agent 选择等待特定 VM 空闲后再进行调度, Action1 到 Action N 表示 Agent 选择当前 Spark 集群中的某个 VM 来为作业创建 Executor。

(6) 奖励

Agent 在采取行动后立即获得奖励, 并根据奖励优化其策略。根据行动的效果, Agent 可能收到正向或负向的奖励。正向奖励鼓励 Agent 继续采取有效行动, 从而优化整个 Episode 中的累积奖励。相反, 负向奖励会促使 Agent 避免不良行为。在每个 Episode 中, 为了最大化累积奖励, Agent 必须综合考虑当前奖励和未来的累积奖励。当前奖励指在 Agent 成功或失败地放置一个 Executor 时, 环境给予 Agent 的奖励。当 Agent 成功放置 Executor 时, 会获得少量正向奖励。如果 Agent 放置 Executor 失败, 将会获得大量负向奖励, 并导致 Episode 提前结束。此外, 为了避

免 Agent 过度执行等待操作以规避失败, 当 Agent 执行等待操作时, 环境会给予 Agent 一小部分负向奖励。当一个 Episode 成功结束时, 环境会根据式 (14) 计算, 并给予 Agent 一个 Episode 奖励。

当 Agent 执行非常差的操作, 且所有 VM 保持开启状态直至整个调度事件完成时, 由于调度策略选择不佳, 提交到 Spark 集群的所有作业需要最长时间才能完成。在这种情况下, 集群的最大成本可以定义为作业执行时间与虚拟机价格的乘积。

$$\text{Cost}_{\max} = \sum_{j \in [1, M]} \text{job}_{T_{\max}}^j \times \sum_{i \in [1, N]} \text{vm}_p^i \quad (8)$$

当 Agent 选择一个非常好的策略, 并且所有 VM 保持开启状态直至整个调度事件完成时, 由于良好的调度策略, 提交到 Spark 集群的所有作业将在最短时间内完成。在这种情况下, 集群的最小成本的定义类似于最大成本的定义, 即作业的最小执行时间与虚拟机的价格的乘积。

$$\text{Cost}_{\min} = \sum_{j \in [1, M]} \text{job}_{T_{\min}}^j \times \sum_{i \in [1, N]} \text{vm}_p^i \quad (9)$$

在理想情况下, 所有作业的 Executor 都很好地被放置, 调度器可以获得一个作业的最短平均作业响应时间, 定义如下。

$$\text{ART}_{T_{\min}} = \left(\sum_{j \in [1, M]} \text{job}_{T_{\min}}^j \right) / M \quad (10)$$

类似地, 如果所有作业放置的 Executor 都很差, 调度器会得到一个作业的最大平均作业响应时间。

$$\text{ART}_{T_{\max}} = \left(\sum_{j \in [1, M]} \text{job}_{T_{\max}}^j \right) / M \quad (11)$$

对集群的总使用成本和作业的平均响应时间使用最小-最大归一化方法, 可以将集群的总使用成本和作业的平均响应时间标准化为 $[0, 1]$ 。

$$\text{Cost}_{\text{norm}} = \frac{\text{Cost}_{\text{total}} - \text{Cost}_{\text{min}}}{\text{Cost}_{\text{max}} - \text{Cost}_{\text{min}}} \quad (12)$$

$$\text{ART}_{\text{norm}} = \frac{\text{ART}_{\text{total}} - \text{ART}_{\text{min}}}{\text{ART}_{\text{max}} - \text{ART}_{\text{min}}} \quad (13)$$

R_{fixed} 是一个固定的 Episode 奖励, 其将随 Agent 在每个 Episode 中表现的变化而变化, 以找寻最大化的目标函数。实际上的目标是使集群的总使用成本和作业的平均响应时间尽可能小, 但在 DRL Agent 的训练过程中, 则是使奖励尽可能大。因此, Episode 奖励 R_{epi} 定义如下。

$$R_{\text{epi}} = R_{\text{fixed}} \times [(1 - \text{Cost}_{\text{norm}}) \times \alpha + (1 - \text{ART}_{\text{norm}}) \times (1 - \alpha)] \quad (14)$$

式 (14) 所示的 Episode 奖励是关于式 (5)、式 (12)、式 (13) 的组合。其中, α 的取值为 [0, 1]。通过调整式 (14) 中 α 的值, 可以控制 Agent 在不同优化目标上的重要性。如果选择 α 为 0, 则训练 Agent 只关注缩短作业的平均响应时间; 如果选择 α 为 1, 则训练 Agent 只关注降低集群总体资源使用成本。如果选择 α 介于 (0, 1), 则训练 Agent 同时关注集群的总使用成本和平均响应时间。

3.2 DRL 作业调度器中的 Agent 训练

为了解决提出的 Spark 作业调度问题, 本文设计了两种基于 DRL 的 Agent 训练方法: 一种是基于 C-DQN 的 Agent, 其采用 Q-Learning 方法; 另一种是基于 PPO+GAE 的 Agent。在 Spark 作业调度环境中, 任务和资源的分配都具有离散性, 因此本文需要使用支持离散化状态和动作空间的 DRL 来解决调度问题。C-DQN 能够有效解决离散状态空间和动作空间的调度问题, 并且在处理大规模状态空间时表现出色; PPO+GAE 适应于集中调度任务的

分布式需求, 具有较强的稳定性和泛化性。对于复杂的调度问题, PPO+GAE 算法表现优异。这两种 Agent 的工作方式有所不同: C-DQN Agent 优化状态动作值, PPO+GAE Agent 直接更新策略。这使它们可以在相似环境下产生相似但不同的结果。在 Spark 作业调度过程中, DRL 环境提供作业规格信息, 类似于实际工作负载的执行情况, 并将 VM 资源的可用性作为状态空间的一部分来使用与更新。

3.2.1 C-DQN Agent

C-DQN 是基于 DQN 架构设计的, 但最后一层的输出不再是 Q 值的均值估计, 而是对值的预测。在实验中, 每个动作都有 N 个神经元来表示对值分布, 这些分布由 N 个分箱来表示, 其范围在 V_{min} 和 V_{max} 之间, 其中, V_{min} 和 V_{max} 分别表示值分布的最小值和最大值。对于每个动作的 N 个神经元输出, 应用 Softmax 函数可以得到一个概率分布, 如式 (15) 所示。

$$p_i(s, a; \theta) = \frac{\exp f_i(s, a; \theta)}{\sum_{j=1}^N \exp f_j(s, a; \theta)} \quad (15)$$

其中, s 是当前状态, a 是动作, θ 是代理网络的参数, f_i 是由神经网络输出的关于分箱 i 的值。

当前对值分布和通过贝尔曼算子计算得到的下一状态的对值分布可能不会精确对齐, 因此需要将值投影回适当的箱中。贝尔曼算子操作和再投影的过程可以总结为式 (16)。

$$(\Phi T Z_{\theta}(s, a))_i = \sum_{j=1}^N \left[1 - \frac{[T z_j]_{V_{\text{min}}}^{V_{\text{max}}} - z_i}{\Delta z} \right] p_j(s', a'; \theta) \quad (16)$$

其中, Φ 是特征提取矩阵, T 是贝尔曼算

子, $Z_\theta(s, a)$ 是在状态 s 下采取动作 a 的期望累计回报, $[Z_j]_{j_{\min}}^{j_{\max}}$ 是值的限制范围, Δz 是一个箱的间隔大小, z_i 是贝尔曼算子集合中的第 i 个算子值。 s' 为执行当前动作 a 后从当前状态 s 转移到的下一个状态, a' 为在下一状态 s' 时可能采取的动作。为了最小化贝尔曼算子作用后的预测分布与实际分布之间的距离, 本文需要设计损失函数并使用梯度下降算法对其进行优化。根据损失函数来更新模型的参数 θ , 以使预测的对值分布尽可能接近目标分布。

$$\mathcal{L}(\theta) = D_{\text{KL}}(\Phi T Z_{\theta'}(s, a) | Z_\theta(s, a)) \quad (17)$$

$$\mathcal{L}(\theta) = - \sum_i (\Phi T Z_{\theta'}(s, a))_i \log p_i(s, a; \theta) \quad (18)$$

其中, $\mathcal{L}(\theta)$ 表示损失函数, D_{KL} 表示Kulback-Leibler散度, θ' 表示参数 θ 根据梯度下降算法更新得到的新参数值。

在基于C-DQN的Spark作业调度器中, 本文使用循环回放缓冲区来存储先前的转换 (s, a, r) , 并从中随机抽取小批量样本来训练深度神经网络。然后, 使用概率 ϵ 来选择任意动作 a , 或者从 $Q_\theta(s, a)$ 中采样一个动作来创建Executor, 并收集奖励 r 。最后, 采样一批数据并利用 $\mathcal{L}(\theta)$ 更新 θ 。进而, 得到如下基于C-DQN Agent的Spark作业调度器构建算法。

算法1 基于C-DQN的Spark作业调度器

输入: 一维向量, 表示集群中每个VM的可用CPU内核数和内存大小, 以及当前正在提交的作业信息;

输出: 创建的Executor放置序列

1. for 每次迭代 do

2. 从环境中获得作业和虚拟机信息和状态 s

3. 以概率 ϵ 选择一个随机动作 a , 否则从 $Q_\theta(s, a)$ 中采样一个动作

4. 利用选择的动作创建Executor

5. 收集奖励 r , 以及新状态 s , 将 s , a , r 存储在回放缓冲区中

6. 从回放缓冲区中采样一批数据, 并更新Agent的网络参数 θ

7. end for

3.2.2 PPO+GAE Agent

PPO Clip是PPO的一种变体, 相比于PPO Penalty有更加广泛和良好的应用效果。其核心思想是通过直接限制策略更新的范围来优化策略, 而不需要随时间变化动态调整惩罚项。在PPO Clip中, 替代优势的定義如下。

$$\mathcal{L}_{\eta_k}^{\text{CLIP}}(\eta) =$$

$$\mathbb{E}_{\tau \sim \pi_{\eta}} \left[\sum_{t=0}^T \left[\min(\rho_t(\pi_{\eta}, \pi_{\eta_k}) A_t^{\pi_{\eta_k}}, \text{clip}(\rho_t(\pi_{\eta}, \pi_{\eta_k}), 1 - \mu, 1 + \mu) A_t^{\pi_{\eta_k}}) \right] \right] \quad (19)$$

$$\rho_t(\eta) = \frac{\pi_{\eta}(a_t | s_t)}{\pi_{\eta_k}(a_t | s_t)} \quad (20)$$

其中, η 表示当前策略的参数; η_k 表示旧策略的参数; A_t 是对时刻 t 的优势的估计; μ 表示一个小的正值, 用来限制策略更新的幅度, 防止更新过快; π_{η} 表示当前策略; $\min$ 为最小值函数; clip 为截断函数, 当其超出指定的上下限时, 函数将返回对应的上限或下限值; a_t 表示在 t 时刻采取的动作; s_t 表示在 t 时刻所处的状态。由于目标项 $1 - \mu$ 和 $1 + \mu$ 不依赖于 η , 其梯度为0。因此, 超出可信区域的样本将被移除, 以免引入过多的更新。

GAE是一种优化优势函数的方法, 旨在平衡方差和偏差之间的关系。在DRL中, 直接使用每个时间步的即时回报 r_t 作为长期回报的估计会导致较大的方差, 因为每个时间步的回报可能波动较大, 从而

影响策略更新的稳定性,但偏差较小。基于优势函数的方法通过整合多个时间步的信息来估计长期回报,减小方差但可能增加偏差。GAE通过加权平均多个时间步的估计来权衡这两种效应。GAE的形式可以表示为式(21)。

$$\sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+1}^V \quad (21)$$

其中, γ 表示折扣因子, λ 表示权重因子, l 表示未来时间步数的索引, δ_{t+1}^V 表示在时间步 $t+1$ 上基于当前值函数 V 的误差。本文采用倒序计算来估计序列中每个时刻的值,即从时刻 $t+1$ 开始,逐步估计到时刻 t 。

$$\hat{A}_t^{\text{GAE}(\gamma, \lambda)} = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+1}^V = \delta_t^V + \gamma\lambda \hat{A}_{t+1}^{\text{GAE}(\gamma, \lambda)} \quad (22)$$

在基于PPO+GAE的Spark作业调度器中, Agent会在事件开始时观察环境,获取集群中所有VM状态和当前提交作业的信息。每次迭代中, Agent先策略网络运行 T 个时间步长,收集每个时间步的状态、动作和奖励;然后使用GAE对优势进行估计,并更新策略,通过新的策略来选择动作并创建Executor;最后用式(19)更新替代优势,用新的替代优势和学习参数 a 更新策略参数 η 。进而,得到如下基于PPO+GAE Agent的Spark作业调度器构建算法。

算法2 基于PPO+GAE的Spark作业调度器

输入: 一维向量,表示集群中每个VM的可用CPU内核数和内存大小,以及当前正在提交的作业信息

输出: 创建的Executor放置序列

```

1. for 每次迭代 do
2. 从环境中获得作业和虚拟机的信息和状态  $s$ 
3. 运行策略  $\pi_\eta T$  次, 收集  $s_t, a_t, r_t$ 
4. 估计优势  $\hat{A}_t^{\text{GAE}(\gamma, \lambda)} = \delta_t^V + \gamma\lambda \hat{A}_{t+1}^{\text{GAE}(\gamma, \lambda)}$ 
5. 更新策略  $\pi_{old} \leftarrow \pi_\eta$ 
6. 利用策略选择动作并创建Executor
7. 更新  $\mathcal{L}_{\eta_k}^{\text{CLIP}}(\eta)$ 
8.  $\eta \leftarrow \eta - \alpha \mathcal{L}_{\pi_{\eta_k}}^{\text{CLIP}}(\pi_{\eta_k})$ 
9. end fo

```

3.3 训练环境调度流程设计

因为Agent在训练期间需要与环境进行大量交互,为了提高评估效率、评估准确性以及降低外部因素的干扰,本文使用模拟的集群环境来训练DRL Agent。表2总结了在进行调度过程中, Agent选择操作后产生的惩罚和奖励。图2所示为DRL Agent训练过程中细化惩罚机制后的环境

表2 动作-事件-奖励对应情况

序号	动作	事件	奖励值
1	0	一个提交的作业尚未放置完所有的Executor,但是Agent开始等待放置Executor	-100
2	0	当前作业之前的作业的执行者已经被放置,等待当前作业的执行者被放置	-2
3	0	提交的作业队列为空,但Agent正在等待调度	-300
4	1...N	集群没有足够的资源	-300
5	1...N	集群拥有足够的资源,持有当前作业的执行者的虚拟机拥有足够的资源	+2
6	1...N	集群有足够的资源,但当前作业的执行者所在的虚拟机没有足够的资源	-300
7	1...N	所有作业都被成功调度,一个Episode成功完成	10 000

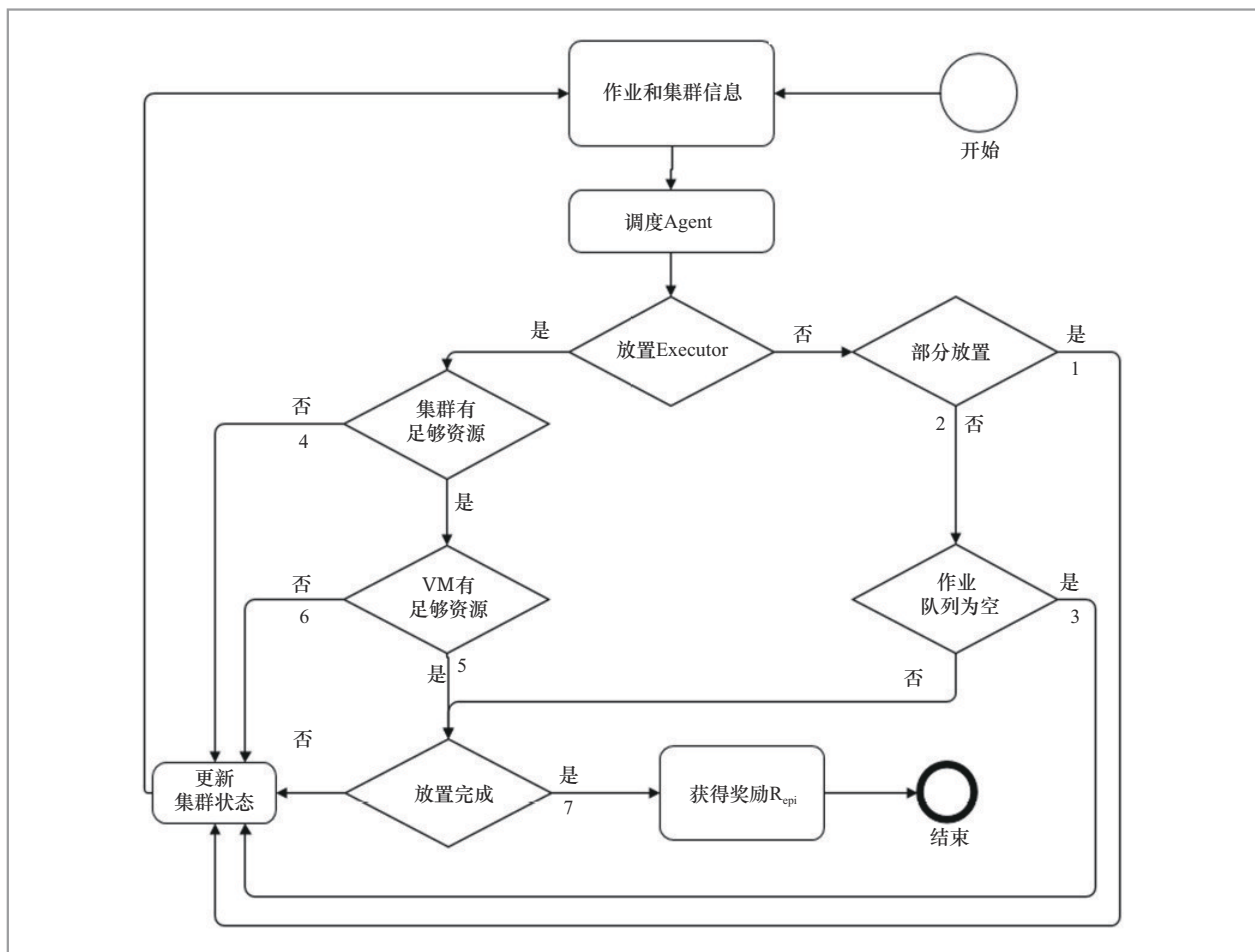


图2 DRL Agent环境工作流程

工作流程。表2中每个惩罚或奖励的序号对应于图2所示的序号。本文使用的模拟环境可以用于训练一个或多个DRL Agent，即实现一个或多个目标，如成本效率、响应时间的最优化。

4 实验与评估

Agent在训练过程中需要与环境频繁交互，而将DRL集成到真实的Spark集群中涉及大量的部署、管理和维护工作，这些工作包括现有系统的集成与协调、监控

模型性能等，超出了本文的主要研究范畴。因此，为了提高评估效率、评估准确性以及降低外部因素的干扰，本文采用模拟的集群环境来训练DRL Agent。这种方式能够有效控制实验条件，减少外部因素的干扰，从而更准确地验证算法的性能。本节将详细描述模拟实验的设置、用于比较的基线调度算法，以及模拟实验的结果分析。

4.1 实验设置

(1) 集群资源

本文选择了具有不同定价模型的VM

实例类型训练和评估 DRL Agent 作业调度器，以优化成本、模拟 Spark 集群部署在公共云上的真实情况。**表 3**总结了集群资源的详细信息。VM 实例的定价以阿里云的开源大数据平台 E-MapReduce（通用型）模型为标准定价。

(2) 工作负载

本文使用 BigDataBench^[42]基准测试套件进行实验，选取了 3 种类型的应用程序：计算密集型的 WordCount、内存密集型的 Sort 和网络密集型的 PageRank。本文使用均匀分布和泊松分布来生成符合**表 4**所示配置需求的作业。

(3) 作业到达时间

本文从 cluster-trace-gpu-v2023 中提取 24 h 的作业到达率（即 QoS 栏中标记为 LS 的数据），用作模拟作业到达时间。作业到达时间为正常（1 h 内 50 个作业到达）和突发（10 min 内 100 个作业到达）。当作业到达时，若 Spark 集群资源不足以进行调度，该作业将被置于等待队列。一旦集群资源充足，调度器将会对其进行调度。

(4) Spark 作业集

本文将工作负载的生成分布和作业到达时间相结合，一共生成了 4 组 Spark 作业集：正常-均匀作业集、正常-泊松作业集、突发-均匀作业集、突发-泊松作业集。针对每个作业集本文都进行了一次单独完整的实验，即从输入一个作业集到运行出实验结果的整个过程。

(5) 训练

本文收集了真实的 Spark 集群中作业的执行信息，并将其用作后续仿真环境的模拟数据。通过仿真环境，本文能够模拟一个高效的 Spark 作业调度过程，从而对本文设计的 DRL Agent 进行训练。在进行调度的过程中，如果作业中的 Executor 放置不当，仿真环境会将作业的执行时间增加 30% 作为惩罚。DRL Agent 的环境工作流程如**图 2**所示。本文对 DRL Agent 的训练环境配置了 TensorFlow 2.11、TF Agent 0.16.0 和 Python 3.7。其中，本文使用 TensorFlow 2.11 提供的模型构建和训练与评估应用程序编程接口（application programming interface, API）；使用 TF Agent 0.16.0 的 C-DQN、PPO+GAE 和基础 API，包括环境交互、策略定义、经验回放、网络架构以及评估工具等；将 Python 3.7 作为运行环境，其提供了执行 TensorFlow 和 TF Agent 代码所需的基础平台。

(6) 超参数

表 5列出了 DRL Agent 的超参数设置以及其他环境参数。**表 5**中的奖励能向 Agent 提供有效的行动奖励，Agent 成功放置 Executor（+2 奖励）、Agent 决定等待（-2 奖励）的情况下，Agent 没有违反任何约束，可以继续进行调度。若在 Agent 选择等待操作时（操作 0）给予的奖励是 0 或大于 0，则可能导致 Agent 无限选择等待以累积正奖励，因此本文对等

表3 集群资源详细信息

名称	CPU/core	内存/GB	数量/个	每秒的使用 价格/元
ecs.g7.xlarge	4	16	3	0.16
ecs.g7.2xlarge	8	32	3	0.32
ecs.g7.3xlarge	12	48	3	0.48

表4 工作负载和配置

应用程序	WordCount、Sort、PageRank
CPU/core	1~6
内存/GB	1~12
Executor	1~8

待操作分配了固定的负奖励（-2奖励），以促使 Agent 在有足够的集群资源可用时放置 Executor。Executor 被安置在错误的位置时，会导致 Episode 提前终止而获得巨大的负奖励（-300奖励）。固定的 Episode 奖励 R_{fixed} 被定得非常高（+10 000奖励）以激励 Agent 寻找更佳策略，从策略中寻求更多的改进，以进一步优化期望的目标。

（7）基线作业调度器

使用 7 种不同的作业调度器作为基线，其中前 5 种为普通基线作业调度器，后两种为基于 DRL Agent 的基线作业调度器。

- 循环（round robin, RR）调度：Spark 使用的默认作业调度器，用于将 Executor 放置在 VM 中。

- 短作业优先（shortest job first, SJF）调度：根据作业运行时间的长短来调度，作业运行时间越短，优先级越高。

- 首次适应（first fit, FF）：每次都寻找第一个可用的 VM，并把 Executor 放置在第一个可用 VM 中。

- 整数线性规划（integer linear programming, ILP）：将 Executor 放置建模成 ILP 问题，然后使用混合 ILP 求解器来查找当前作业对应的所有 Executor 的最佳放置位置。

- 自适应 Executor 放置（adaptive executor placement, AEP）：使用先前的作业配置信息在集中式和分散式 Executor 放置方法之间进行切换。在调度过程中，其可以为作业自由选择分散或合并放置方法。该算法使用首选放置策略把作业的 Executor 放置在第一个选择的 VM 中。

- DQN Agent^[34]：一种基于 Q-Learning 算法的作业调度器，与本文提出的 DRL Agent 作业调度器使用同样模拟仿真环境进行训练。

- REINFORCE Agent^[34]：一种基于

表 5 DRL Agent 的超参数和环境参数

参数	值
R_{fixed}	10 000
批次大小	64
Episode 数	10
惩罚指数	0.001
学习速率	0.001
折扣系数	0.99
α	0.0/0.2/0.5/0.8/1.0
迭代数/次	10 000/20 000 100 000/200 000
有效行动奖励	-2/2
Q-网络中的神经元个数	200
GAE 方差和偏倚系数	0.01
Executor 放置不当惩罚系数	0.3
Clip epsilon 系数	0.2
单轮更新的采样步数	25
数据复用次数/次	10 000
PPO 网络维度数/维	(200,100)
Episode 提前终止奖励	-300

策略梯度算法的作业调度器，与本文提出的 DRL Agent 作业调度器使用相同的模拟仿真环境进行训练。

4.2 DRL Agent 的收敛性

图 3 至图 6 分别展示了 C-DQN Agent 和 PPO+GAE Agent 在上述 4 个 Spark 作业集上的收敛性。使用不同的 α 参数值训练 DRL Agent，说明不同 α 参数对作业调度效果的影响。C-DQN Agent 每迭代 1 000 次对其进行一次评估，而 PPO+GAE Agent 每迭代 10 000 次对其进行一次评估。每次评估 Agent 时，Agent 执行 10 次并对结果取平均值。对于作业到达时间为正常模式的 Spark 作业集，本文对 DRL Agent 分别进行了 10 000 和 100 000 次迭代的训练；对于突发模式的 Spark 作业集来说，分别进行了 20 000 和 200 000 次

迭代训练。

在图3至图6中, α 值表示对集群总使用成本和作业平均响应时间的重视程度。 α 值越大表示 Agent 越重视集群的总使用成本; 较小的 α 值表示 Agent 更重视作业的平均响应时间。当 α 值为 0 时, Agent 只关注作业的平均响应时间, 忽略集群的总使用成本。当 α 的值为 0.5 时, Agent 同等关

注集群的总使用成本和作业的平均响应时间。当 α 值为 1 时, Agent 只关注集群的总使用成本, 而忽略作业的平均响应时间。因此, Agent 在训练期间获得的 Episode 奖励会根据 α 值的变化而变化。

图3、图4展示了 C-DQN Agent 在 4 个作业集训练中的累积平均奖励变化。类似地, 图5、图6展示了 PPO+GAE Agent 在

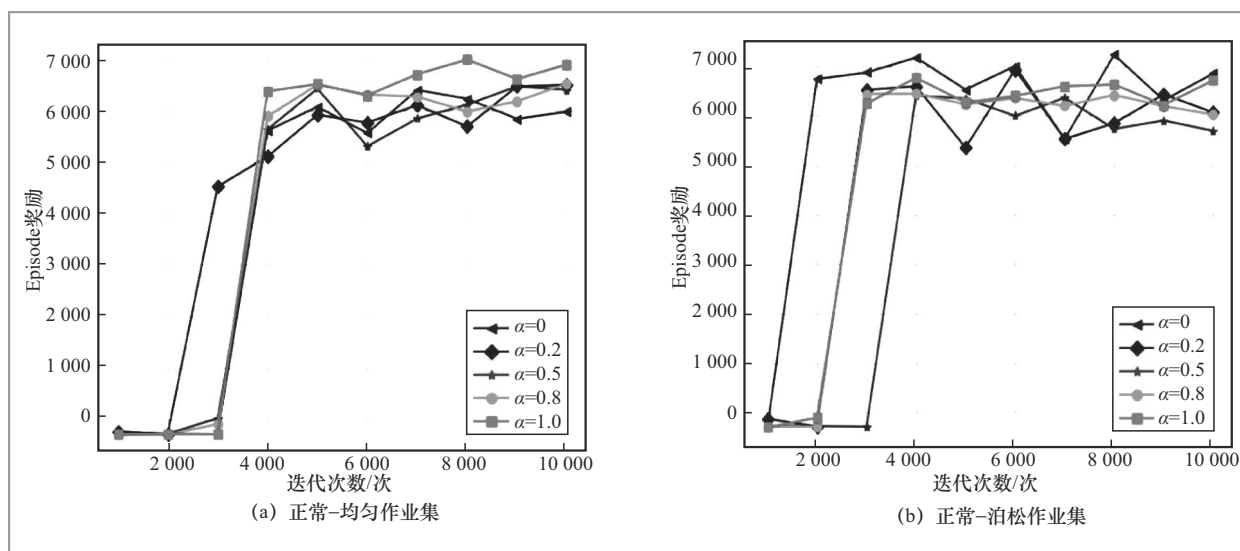


图3 C-DQN Agent在正常作业到达时间模式下的收敛性验证

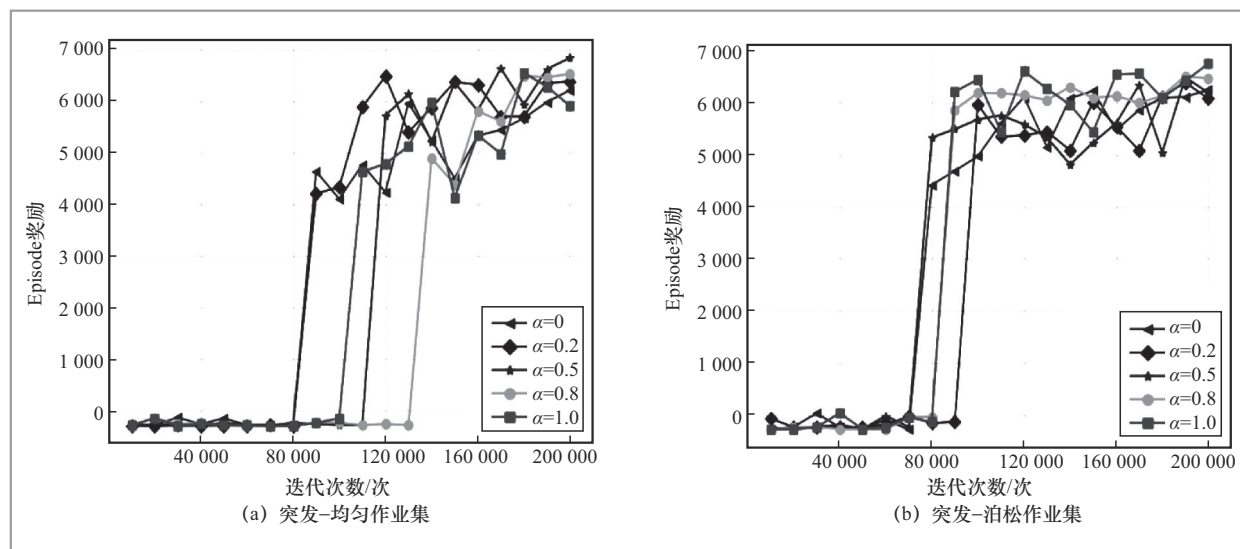


图4 C-DQN Agent在突发作业到达时间模式下的收敛性验证

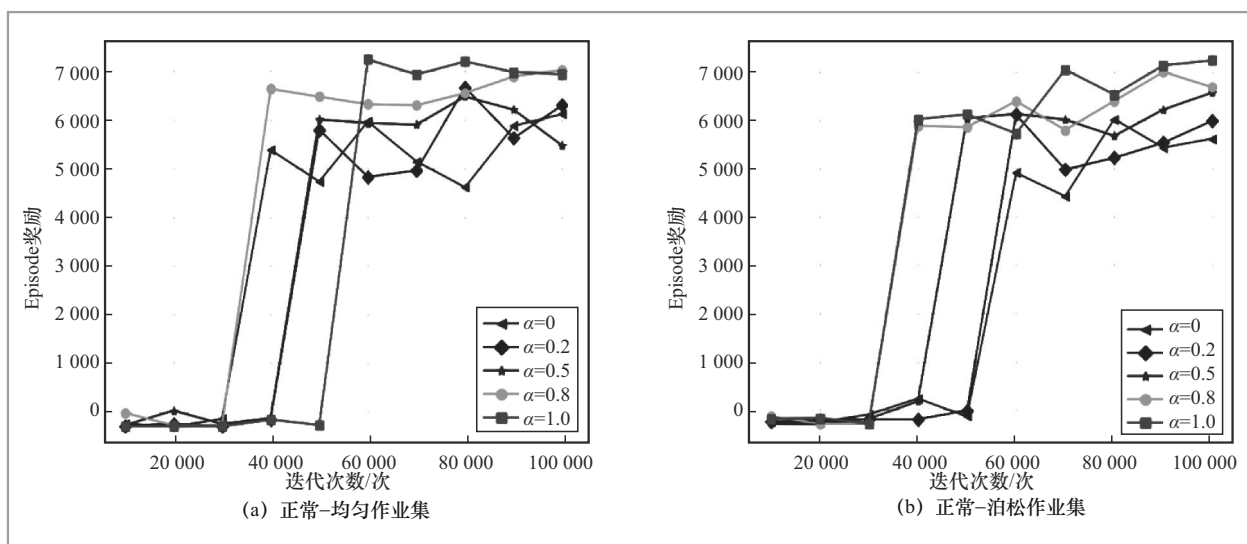


图5 PPO+GAE Agent在正常作业到达时间模式下的收敛性验证

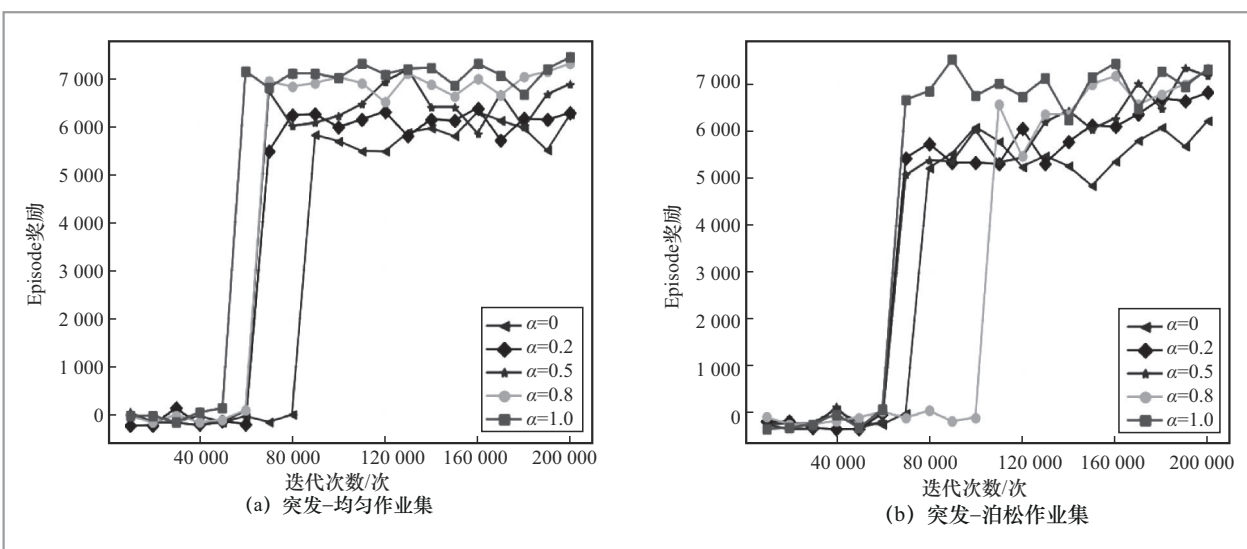


图6 PPO+GAE Agent在突发作业到达时间模式下的收敛性验证

训练迭代中的累积平均奖励变化。在训练过程中, DRL Agent 不断地进行学习, 并追求累积更高的奖励。高奖励表明 DRL Agent 学习到了更优的策略。初始阶段, 两个 DRL Agent 都只能获得负面奖励, 但随着多次迭代探索状态空间, 它们逐渐获得更多奖励, 并最终趋于稳定。在收敛过程中, DRL Agent 持续探索更多可能的调

度策略, 寻求更高的奖励, 有时可能会导致奖励暂时下降。由于作业数量增多且到达速率加快, 两个 DRL Agent 需要更多时间学习如何调度突发到达的作业模式, 因此它们的收敛需要更多的迭代次数。

本文实验结果显示基于 DRL 的 C-DQN Agent 和 PPO+GAE Agent 能够在训练过程中逐步达到收敛状态, 说明它们能

够通过学习和优化，逐渐改进作业调度性能，且DRL Agent能够在探索和利用之间找到较好的平衡点。本文实验结果不仅证实了DRL Agent在面对复杂的、不确定性高的环境时，能够通过DRL方法有效地学习和优化决策策略，还表明DRL Agent在长期优化过程中能够通过不断学习和调整，在长期的时间尺度上达到更高的作业完成率，获得更高的资源利用率。特别地，DRL Agent能够灵活地调整作业调度策略来应对作业调度系统中随时间变化的工作负载、资源可用性和其他实时变化因素，

保持作业调度的高效性和稳定性。

4.3 集群总使用成本评估

验证了C-DQN Agent和PPO+GAE Agent的收敛性之后，本文进一步使用模拟环境对比评估了新提出的基于DRL Agent的作业调度器与传统基线作业调度器在集群总成本方面的表现。关于集群总使用成本、作业的平均响应时间和QoS达成率的计算，分别按照式（3）、式（4）和式（6）计算。**表6**详细展示了4个

表6 两种基于DRL的作业调度器与基线作业调度器的集群总使用成本对比

作业调度器	不同作业集下的总使用成本/元			
	正常-均匀作业集	正常-泊松作业集	突发-均匀作业集	突发-泊松作业集
FF	57.6192	56.6775	49.2352	47.2832
SJF	54.8352	54.8352	50.1568	46.7936
RR	69.0144	68.9408	53.0944	50.9152
AEP	62.3104	60.2624	52.4045	48.8320
ILP	43.4416	44.0688	43.6865	42.6865
DQN0.0	60.7651	61.9209	53.4792	47.7292
DQN0.2	56.3888	60.6874	49.0311	45.3270
DQN0.5	57.1302	58.9987	50.9734	46.1140
DQN0.8	55.8355	59.7968	51.2682	45.0972
DQN1.0	54.7936	55.8845	46.5247	44.7595
REINFORCE0.0	56.2726	55.4524	50.1493	47.8571
REINFORCE0.2	53.2947	50.8854	52.1880	48.9367
REINFORCE0.5	51.6262	51.0361	51.6477	49.8664
REINFORCE0.8	47.8220	49.1932	48.0789	45.6488
REINFORCE1.0	49.2153	48.4832	46.9984	44.3587
C-DQN0.0	56.5158	58.4816	50.8954	46.7622
C-DQN0.2	54.4762	56.9679	49.3458	45.2346
C-DQN0.5	53.5030	53.8963	51.6790	48.4477
C-DQN0.8	54.1725	51.4380	43.9975	44.4381
C-DQN1.0	52.4858	50.3677	46.3538	43.1210
PPO+GAE0.0	52.8755	55.1324	47.4870	43.8755
PPO+GAE0.2	50.9798	52.6483	51.8769	42.9798
PPO+GAE0.5	45.0085	50.8744	45.2345	45.5085
PPO+GAE0.8	48.4310	48.9876	43.7805	40.4310
PPO+GAE1.0	44.4410	46.1932	41.2783	41.4410

Spark 作业集下各个作业调度器的集群总使用成本。表中第一列为作业调度器的类型，后面的数字代表着基于 DRL Agent 的作业调度器的 α 值。

在正常的作业到达时间模式下，由于作业到达时间相对宽松，Spark 集群拥有足够的资源来执行作业。在这种情况下，ILP 每次都寻找最优的作业放置策略，从而能将集群总成本降至最低。然而，表现最好的 DRL——PPO+GAE 在成本上表现非常接近 ILP 作业调度器。具体而言，PPO+GAE 的表现仅比最佳的 ILP 差约 2%，但却比其他普通基线作业调度器表现好约 20%，比最好的基线 DRL 好约 5%。尽管在某些 α 值下，本文提出的基于 DRL 的作业调度器的集群总使用成本可能会比基线算法高，但从整体趋势来看，其通常比基线作业调度器更具成本效益上的优势。

作业到达时间模式为突发时，通常没有足够的 Spark 集群资源来调度所有作业，因此作业调度器必须等到资源被已经运行完毕的作业释放出来才能继续调度剩余作业。尽管 ILP 在该模式下依旧使总使用成本很低，但是由于作业到达密集、作业执行器放置不当，作业的执行时间增长，从而增加了集群的总使用成本。所以，ILP 在该模式下并不是最优的作业调度器。本文提出的基于 DRL 的作业调度器充分考虑了作业的特性以及环境状态，因此总能为作业提供最佳的调度策略。表现最佳的 PPO + GAE 比普通基线作业调度器好约 5%，比最好的基线 DRL 好约 10%。

4.4 作业平均响应时间评估

本文在评估集群总使用成本的同时，还计算了一段时间内安排的所有作业的平均响应时间，以比较作业调度器的性能。

表7给出了各个作业调度器在 4 个 Spark 作业集下的作业平均响应时间。

对于正常的作业到达时间模式，ILP 虽然每次都能够通过寻找最优的放置策略来降低总使用成本，但这一过程却需要花费大量的时间，导致作业的平均响应时间最大。本文提出的基于 DRL 的作业调度器，即 C-DQN 和 PPO+GAE，要比普通基线作业调度器要好，但是另外的两个基线 DRL 作业调度器，即 REINFORCE 和 DQN，则表现得与普通基线作业调度器差不多。本文提出的基于 DRL 的作业调度器相比于所有参考基线作业调度器展现了卓越的性能，其实现的调度效果相较于同类型的基线 DRL 作业调度器，平均响应时间减少了大约 4%；与普通的基线作业调度器相比，平均响应时间缩短接近 10%。这一显著的改进表明优化后的基于 DRL 的作业调度器能够更有效地理解作业特性和掌握环境状态，为 Spark 集群作业调度提供了一个更高效的解决方案。

作业到达时间模式为突发时，作业通常必须等待更多的可用资源。而且，如果作业安排与作业特征不匹配，则作业的完成时间将增加，这将导致较长的平均作业持续时间。所以普通作业调度器很难学习到作业的特性，导致普通作业调度器在面对突发作业到达时间模式时会有更长的作业平均响应时间。基于 DRL 的作业调度器可以捕获作业持续时间和作业安排良好性之间的潜在关系，并将此信息作为减少作业持续时间的策略纳入经过训练的策略中，从而缩短作业平均响应时间。因此，本文提出的基于 DRL 的作业调度器在整体上能够将作业的平均响应时间相比基线调度器缩短约 5%。具体来说，本文提出的基于 DRL 的作业调度器的作业平均响应时间比基线 DRL 作业调度器缩短约 3%，比普通

表7 两种基于DRL的作业调度器与基线作业调度器的作业平均响应时间对比

作业调度器	不同作业集下的作业平均响应时间/s			
	正常-均匀作业集	正常-泊松作业集	突发-均匀作业集	突发-泊松作业集
FF	56.46	61.75	163.44	197.69
SJF	57.66	60.45	162.31	195.16
RR	62.78	67.14	178.98	214.63
AEP	59.16	63.20	164.62	193.06
ILP	68.80	72.80	184.36	220.47
DQN0.0	54.80	56.78	158.34	186.63
DQN0.2	56.83	58.17	156.87	192.42
DQN0.5	55.14	59.22	159.97	189.31
DQN0.8	57.61	61.62	165.99	195.88
DQN1.0	58.23	60.47	162.83	197.50
REINFORCE0.0	56.42	60.15	160.35	189.40
REINFORCE0.2	60.94	64.38	159.39	191.11
REINFORCE0.5	59.46	62.46	161.44	192.01
REINFORCE0.8	63.82	63.51	162.64	190.75
REINFORCE1.0	64.62	66.67	163.88	193.10
C-DQN0.0	52.81	55.05	155.49	185.12
C-DQN0.2	53.48	56.48	157.48	187.81
C-DQN0.5	55.61	57.16	156.13	190.48
C-DQN0.8	56.67	60.17	163.89	189.82
C-DQN1.0	58.17	58.86	159.32	194.20
PPO+GAE0.0	55.84	58.24	150.97	182.84
PPO+GAE0.2	57.87	60.02	157.32	186.87
PPO+GAE0.5	56.65	61.56	152.48	185.65
PPO+GAE0.8	58.59	60.48	155.98	193.59
PPO+GAE1.0	59.99	62.89	162.47	189.99

基线作业调度器缩短约 8%。

4.5 QoS 达成率评估

要满足 QoS 要求，人们需要同时考虑集群总使用成本和作业平均响应时间。所以对于普通基线作业调度器来说，集群的总使用成本和作业的平均响应时间都较高，导致其 QoS 达成率相对比较低。而基于 DRL 的作业调度器则能够通过不断地学习作业的特征和 Spark 集群的状态变化，降低集群的总使用成本或者缩短作业的平均响应时间，获得更高的 QoS 达成率。

表 8 展示了基于 DRL 的作业调度器和基线作业调度器的差异，其中 α 的取值为 [0,1]。调整 α 的值可以使 Agent 侧重于不同的目标，也可以使基于 DRL 的作业调度器的 QoS 达成率不断变化。如果选择 α 为 0，则 Agent 将接受训练，基于 DRL 的作业调度器的 QoS 达成率的提高只依赖于缩短作业的平均响应时间；如果选择 α 为 1，则 Agent 将只降低集群总体资源使用成本，QoS 达成率的提高将只依赖于降低集群总体使用成本。因此，当 α 等于 0.5 或者接近 0.5 时，基于 DRL 的作业调度器将注重集群总使用成本和作业平均响应时间的

表8 两种基于DRL的作业调度器与基线作业调度器的QoS达成率对比

作业调度器	不同作业集下的QoS达成率			
	正常-均匀作业集	正常-泊松作业集	突发-均匀作业集	突发-泊松作业集
FF	0.67	0.64	0.47	0.46
SJF	0.69	0.69	0.50	0.52
RR	0.58	0.56	0.38	0.39
AEP	0.64	0.61	0.48	0.47
ILP	0.52	0.50	0.42	0.44
DQN0.0	0.69	0.68	0.54	0.63
DQN0.2	0.72	0.71	0.70	0.61
DQN0.5	0.77	0.76	0.66	0.68
DQN0.8	0.68	0.73	0.57	0.67
DQN1.0	0.75	0.70	0.68	0.54
REINFORCE0.0	0.77	0.72	0.64	0.68
REINFORCE0.2	0.76	0.70	0.65	0.61
REINFORCE0.5	0.80	0.79	0.69	0.66
REINFORCE0.8	0.74	0.74	0.62	0.71
REINFORCE1.0	0.72	0.75	0.62	0.61
C-DQN0.0	0.74	0.74	0.66	0.64
C-DQN0.2	0.78	0.80	0.63	0.62
C-DQN0.5	0.86	0.78	0.73	0.72
C-DQN0.8	0.82	0.84	0.68	0.67
C-DQN1.0	0.76	0.73	0.70	0.68
PPO+GAE0.0	0.78	0.75	0.72	0.69
PPO+GAE0.2	0.83	0.79	0.68	0.66
PPO+GAE0.5	0.87	0.85	0.76	0.71
PPO+GAE0.8	0.80	0.82	0.71	0.74
PPO+GAE1.0	0.79	0.74	0.64	0.72

平衡，使两方面都下降，从而得到了更高的QoS达成率。总体来讲，本文提出的基于C-DQN的作业调度器和基于PPO+GAE的作业调度器的QoS达成率比基线DRL作业调度器好约10%，比普通基线作业调度器好约20%。

5 结束语

本文应用DRL来解决云环境下Spark集群作业的调度问题，基于训练的两种核心DRL Agent，即C-DQN Agent和PPO+

GAE Agent，设计了能够同时优化多个QoS要求的Spark作业调度器。本文为了帮助Agent理解资源限制、作业性能变化和集群VM成本，模拟了有效的奖励信号。C-DQN Agent和PPO+GAE Agent可以自动地与集群环境进行交互学习进而优化作业调度目标，无须预先了解关于作业或集群的具体信息。实验结果显示，本文设计的DRL Agent作业调度器具备降低总使用成本和缩短平均响应时间的能力，表现显著优于基线作业调度器。此外，其还展现出优秀的多目标优化平衡性能，相较于基线作业调度器，在成本和响应时间的优

化方面均有明显改进。但本文设计的作业调度器还存在如下问题需要进一步改善,在今后的工作中值得深入探索与研究。

- 未来应探讨不同作业特性如何影响作业持续时间,研究更加有效的奖励模型,使其可以在即时奖励的情况下调整成本和作业持续时间,从而使DRL Agent在执行作业调度过程中更高效。

- 未来应提取在训练DRL Agent过程中基于当前状态做出决策的策略,用于真正的大规模Spark集群部署,以进一步训练Agent获得更精确和高效的作业调度决策模型,并把这个模型应用在真实集群中来解决真实的Spark集群作业调度问题。

- 未来应把调度目标从作业的平均响应时间改为作业期限的最大延迟,并以此为导向来拓展DRL Agent的架构,使其能够有效地优化作业调度,实现进一步减少作业延迟和提高资源利用率的目的。

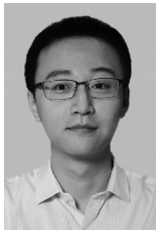
参考文献:

- [1] VAVILAPALLI V K, MURTHY A C, DOUGLAS C, et al. Apache hadoop YARN: yet another resource negotiator[C]//Proceedings of the 4th annual Symposium on Cloud Computing. New York: ACM, 2013: 1-16.
- [2] ZAHARIA M, XIN R S, WENDELL P, et al. Apache spark[J]. Communications of the ACM, 2016, 59(11): 56-65.
- [3] CHENG L, VAN DONGEN B F, VAN DER AALST W M P. Scalable discovery of hybrid process models in a cloud computing environment[J]. IEEE Transactions on Services Computing, 2020, 13(2): 368-380.
- [4] LIU J W, SHEN H Y, CHI H M, et al. A low-cost multi-failure resilient replication scheme for high-data availability in cloud storage[J]. IEEE/ACM Transactions on Networking, 2021, 29(4): 1436-1451.
- [5] ZAHARIA M, CHOWDHURY M, FRANKLIN M J, et al. Spark: Cluster computing with working sets[C]//Proceedings of the 2nd USENIX Workshop on Hot Topics in Cloud Computing[S.l.:s.n.], 2010.
- [6] SHI W H, LI H J, ZENG H. DRL-based and bsld-aware job scheduling for apache spark cluster in hybrid cloud computing environments[J]. Journal of Grid Computing, 2022, 20(4): 44.
- [7] TANG S J, HE B S, YU C, et al. A survey on spark ecosystem: big data processing infrastructure, machine learning, and applications[J]. IEEE Transactions on Knowledge and Data Engineering, 2022, 34(1): 71-91.
- [8] ZAHARIA M, CHOWDHURY M, TATHAGATA D, et al. Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing[C]//Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation. Berkeley: USENIX, 2012: 15-28.
- [9] SAEED M M, AL AGHBARI Z, ALSHARIDAH M. Big data clustering techniques based on spark: a literature review[J]. PeerJ Computer Science, 2020, 6: e321.
- [10] LU S X, ZHAO M M, LI C L, et al. Time-aware data partition optimization and heterogeneous task scheduling strategies in spark clusters[J]. The Computer Journal, 2024, 67(2): 762-776.
- [11] ZHANG X D, LI X P, DU H A, et al. Task scheduling for spark applications with data affinity on heterogeneous clusters [J]. IEEE Internet of Things Journal, 2022, 9(21): 21792-21801.

- [12] TANG Z, ZENG A L, ZHANG X D, et al. Dynamic memory-aware scheduling in spark computing environment[J]. Journal of Parallel and Distributed Computing, 2020, 141: 10–22.
- [13] SHABESTARI F, RAHMANI A M, NAVIMIPOUR N J, et al. A YARN-based energy-aware scheduling method for big data applications under deadline constraints[J]. Journal of Grid Computing, 2022, 20(4): 38.
- [14] ISLAM M T, SRIRAMA S N, KARUNASEKERA S, et al. Cost-efficient dynamic scheduling of big data applications in apache spark on cloud[J]. Journal of Systems and Software, 2020, 162: 110515.
- [15] 张晨浩, 肖利民, 秦广军, 等. 面向大数据处理应用的广域存算协同调度系统[J]. 大数据, 2021, 7(5): 82–97.
- ZHANG C H, XIAO L M, QIN G J, et al. A wide-area collaborative scheduling system oriented to big data processing applications[J]. Big Data Research, 2021, 7(5): 82–97.
- [16] DUAN J J, SHI D, DIAO R S, et al. Deep-reinforcement-learning-based autonomous voltage control for power grid operations[J]. IEEE Transactions on Power Systems, 2020, 35(1): 814–817.
- [17] JIANG L, HUANG H Y, DING Z H. Path planning for intelligent robots based on deep Q-learning with experience replay and heuristic knowledge[J]. IEEE/CAA Journal of Automatica Sinica, 2020, 7(4): 1179–1189.
- [18] LIN J P, CUI D L, PENG Z P, et al. A two-stage framework for the multi-user multi-data center job scheduling and resource allocation[J]. IEEE Access, 2020, 8: 197863–197874.
- [19] LI H J, LU L, SHI W H, et al. Energy-aware scheduling for spark job based on deep reinforcement learning in cloud[J]. Computing, 2023, 105(8): 1717–1743.
- [20] WANG X G, CAO J, BUYYA R. Adaptive cloud bundle provisioning and multi-workflow scheduling via coalition reinforcement learning[J]. IEEE Transactions on Computers, 2023, 72(4): 1041–1054.
- [21] MARC G B, WILL D, R MI M. A distributional perspective on reinforcement learning[C]//Proceedings of the 34th International Conference on Machine Learning, NSW: JMLR, 2017: 449–458.
- [22] SCHULMAN J, WOLSKI F, DHARIWAL P, et al. Proximal policy optimization algorithms[EB]. arXiv preprint, 2017, arXiv: 1707.06347.
- [23] DENNINNART C, GENTRY J, MOKHTARI A, et al. Efficient task pruning mechanism to improve robustness of heterogeneous computing systems[J]. Journal of Parallel and Distributed Computing, 2020, 142: 46–61.
- [24] HUSSAIN S M, BEGH G R. Hybrid heuristic algorithm for cost-efficient QoS aware task scheduling in fog-cloud environment[J]. Journal of Computational Science, 2022, 64: 101828.
- [25] ZHU J Y, YANG R Y, SUN X Y, et al. QoS-aware co-scheduling for distributed long-running applications on shared clusters[J]. IEEE Transactions on Parallel and Distributed Systems, 2022, 33(12): 4818–4834.
- [26] SHARMA N, SONAL, GARG P. Ant colony based optimization model for QoS-based task scheduling in cloud computing environment[J]. Measurement: Sensors, 2022, 24: 100531.
- [27] TAGHINEZHAD-NIAR A, PASHAZADEH S, TAHERI J. QoS-aware online scheduling of multiple workflows under task execution time uncertainty in clouds[J]. Cluster Computing, 2022, 25(6): 3767–3784.

- [28] FU Z M, TANG Z, YANG L, et al. An optimal locality-aware task scheduling algorithm based on bipartite graph modelling for spark applications[J]. IEEE Transactions on Parallel and Distributed Systems, 2020, 31(10): 2406–2420.
- [29] LI H J, WEI Y J, XIONG Y, et al. A frequency-aware and energy-saving strategy based on DVFS for Spark[J]. The Journal of Supercomputing, 2021, 77(10): 11575–11596.
- [30] 刘汪根, 郑淮城, 荣国平. 云环境下大规模分布式计算数据感知的调度系统[J]. 大数据, 2020, 6(1): 81–98.
- LIU W G, ZHENG H C, RONG G P. A scheduler system for large-scale distributed data computing in cloud[J]. Big Data Research, 2020, 6(1): 81–98.
- [31] LI C L, CAI Q Q, LUO Y L. Dynamic data replacement and adaptive scheduling policies in spark[J]. Cluster Computing, 2022, 25(2): 1421–1439.
- [32] LIU L B, XU H. Elasecutor: elastic executor scheduling in data analytics systems[J]. IEEE/ACM Transactions on Networking, 2021, 29(2): 681–694.
- [33] ISLAM M T, WU H M, KARUNASEKERA S, et al. SLA-based scheduling of spark jobs in hybrid cloud computing environments[J]. IEEE Transactions on Computers, 2022, 71(5): 1117–1132.
- [34] MOHAMMAD HASANI ZADE B, MANSOURI N. Improved red fox optimizer with fuzzy theory and game theory for task scheduling in cloud environment[J]. Journal of Computational Science, 2022, 63: 101805.
- [35] ZHANG Z X, ZHAO M K, WANG H, et al. An efficient interval many-objective evolutionary algorithm for cloud task scheduling problem under uncertainty[J]. Information Sciences, 2022, 583: 56–72.
- [36] ISLAM M T, KARUNASEKERA S, BUYYA R. Performance and cost-efficient spark job scheduling based on deep reinforcement learning in cloud computing environments[J]. IEEE Transactions on Parallel and Distributed Systems, 2022, 33(7): 1695–1710.
- [37] GUO W X, TIAN W H, YE Y F, et al. Cloud resource scheduling with deep reinforcement learning and imitation learning[J]. IEEE Internet of Things Journal, 2021, 8(5): 3576–3586.
- [38] MAO H Z, SCHWARZKOPF M, VENKATAKRISHNAN S B, et al. Learning scheduling algorithms for data processing clusters[C]//Proceedings of the ACM Special Interest Group on Data Communication. New York: ACM, 2019: 270–288.
- [39] ZHAO Z H, SHI X Y, SHANG M S. Performance and cost-aware task scheduling via deep reinforcement learning in cloud environment[M]//Service-Oriented Computing. Cham: Springer Nature Switzerland, 2022: 600–615.
- [40] MOHAMMAD HASANI ZADE B, MANSOURI N, JAVIDI M M. A two-stage scheduler based on New Caledonian Crow Learning Algorithm and reinforcement learning strategy for cloud environment[J]. Journal of Network and Computer Applications, 2022, 202: 103385.
- [41] PRADHAN A, BISOY S K, KAUTISH S, et al. Intelligent decision-making of load balancing using deep reinforcement learning and parallel PSO in cloud environment[J]. IEEE Access, 2022, 10: 76939–76952.
- [42] GAO W L, ZHAN J F, WANG L, et al. BigDataBench: a scalable and unified big data and AI benchmark suite[EB]. arXiv preprint, 2018, arXiv: 1802.08254.

作者简介



何玉林（1982–），男，博士，人工智能与数字经济广东省实验室（深圳）研究员，主要研究方向为大数据系统计算技术、面向大数据的数据挖掘和机器学习算法设计及应用。



莫沛恒（2000–），男，深圳大学计算机与软件学院硕士生，主要研究方向为大数据分布式计算、Spark 性能优化技术、高性能数据挖掘与机器学习算法设计。



Philippe Fournier-Viger（1980–），男，博士，深圳大学大数据技术与应用研究所特聘教授，主要研究方向为数据挖掘、人工智能、知识表示和推理、认知模型构建。



黄哲学（1959–），男，博士，深圳大学大数据技术与应用研究所特聘教授、所长，主要研究方向为数据挖掘、机器学习、大数据处理与分析、大数据系统计算技术。

收稿日期: 2024-07-02
通信作者: 何玉林, yulinhe@gml.ac.cn
基金项目: 广东省自然科学基金项目(No.2023A1515011667);广东省基础与应用基础研究基金项目(No.2023B1515120020);深圳市科技重大专项项目(No.KJZD20230923114809020)
Foundation Items: The Natural Science Foundation of Guangdong Province (No.2023A1515011667), Guangdong Basic and Applied Basic Research Foundation (No.2023B1515120020), Science and Technology Major Project of Shenzhen (No. KJZD20230923114809020)