

Performance and Cost-Efficient Spark Job Scheduling

Based on Deep Reinforcement Learning in Cloud

Computing Environments

Muhammed Tawfiqul Islam, Shanika Karunasekera and Rajkumar Buyya

摘要

大数据框架（如 Spark 和 Hadoop）广泛应用于研究和行业中的分析工作。云计算提供了价格合理、易于管理的计算资源。因此，许多组织正转向大数据计算集群的云部署。然而，在存在各种服务级别协议（SLA）目标的情况下，作业调度是一个复杂的问题，例如降低货币成本和提高作业性能。现有的大多数研究没有同时解决多个目标，也没有捕捉到固有的集群和工作量特征。描述了云部署 Spark 集群的作业调度问题，并提出了一种新的强化学习（RL）模型来适应 SLA 目标。开发了 RL 集群环境，并在 TF-Agents 框架中实现了两个基于深度强化学习（DRL）的调度程序。提议的基于 DRL 的调度代理在细粒度级别工作，以放置作业的执行器，同时利用云 VM 实例的定价模型。此外，基于 DRL 的代理还可以了解不同类型作业的固有特征，以找到适当的位置，从而降低集群 VM 的总使用成本和平均作业持续时间。结果表明，所提出的基于 DRL 的算法可以将虚拟机的使用成本降低 30%。

关键词：云计算；成本效率；性能改进；深度强化学习

1 引言

大型数据处理框架，如 Hadoop^[1]、Spark^[2]、Storm¹，由于它们在科学、商业和研究等许多重要领域的数据分析领域中的应用，变得极为流行。这些框架可以部署在本地物理资源或云上。然而，云服务提供商（CSP）在现收现付模式上提供灵活、可扩展且价格合理的计算资源。此外，云资源比物理资源更易于管理和部署。因此，许多组织正朝着在云上部署大数据分析集群的方向发展，以避免管理物理资源的麻烦。服务水平协议（SLA）是消费者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（QoS）要求。在大数据计算集群的作业调度问题中，最重要的目标是提高作业的性能。然而，当集群部署在云端时，如果存在其他重要的 SLA 目标，例如降低金钱成本，作业调度就会变得更加复杂。

在这项工作中，重点关注云部署 Apache Spark 集群的基于 SLA 的作业调度问题。选择了 Apache Spark，因为它是大数据处理最重要的框架之一。Spark 将中间结果存储在内存中，以加快处理速度。此外，它比其他平台更具伸缩性，适合运行各种复杂的分析作业。Spark 程序可以用许多高级编程语言实现，它还支持不同的数据源，如 HDFS^[3]、Hbase^[4]、Cassandra^[5]、Amazon S3²。

部署 Spark 集群后，可以使用它运行一个或多个作业。通常，当作业提交执行时，框架调度程序负责分配资源块（例如 CPU、内存），这些资源块称为执行器。一个作业可以与这些执行器并行运行一个或多个任务。默认的 Spark 调度程序可以在工作节点中以分布式方式创建作业的执行器。这种方法允许平衡地使用集群，避免了共存执行器之间的干扰，从而提高了计算密集型工作负载的性能。此

外，作业的执行器可以封装在更少的节点中。虽然打包放置给工作节点带来了更大的压力，但随着来自同一作业的执行器之间的通信变成节点内通信，它可以提高网络目标作业的性能。在 Spark 框架调度中，只有在用户必须在两个选项（排列或合并）之间进行选择的情况下，才能选择静态设置。但是，对于不同的作业类型，如果这些作业同时在集群中运行，那么默认调度程序将无法支持不同的放置策略。此外，框架调度器无法捕获资源和工作负载特性的固有知识，从而有效地适应目标。现有很多论文没有考虑执行器创建和目标对象的含义。大多数工作还假设集群设置是同质的。然而，云部署集群并非如此，在这种情况下，可以利用不同 VM 实例的定价模型来降低使用整个集群的总体资金成本。最后，基于启发式和基于性能模型的解决方案通常侧重于特定的场景，不能概括为在考虑工作负载固有特性的同时适应广泛的目标。

最近，基于强化学习（RL）的方法被用于解决复杂的现实世界问题^{[6],[7],[8]}。基于 RL 的代理可以接受训练，以在多个目标之间找到一个良好的平衡。RL 代理可以捕获各种固有的集群和工作负载特征，并自动适应变化。RL 代理事先不了解环境。相反，他们与真实环境互动，探索不同的情况，并根据行动收集奖励。代理利用这些经验制定一项策略，最大化整体回报

2 相关工作

2.1 云虚拟机和数据中心中的调度

在云虚拟机中调度任务和在数据中心中调度虚拟机创建都是研究得很好的问题。PARIS^[9]对不同 VM 类型的各种工作负载的性能进行了建模，以确定性能和成本节约之间的权衡。因此，可以使用此模型选择成本和性能都最佳的 VM，以便在云托管的 VM 中运行特定的工作负载。Yuan 等人^[10]提出了一种分布式绿色数据中心（DGDC）的双目标任务调度算法。他们为 DGDC 制定了一种多目标优化方法，通过联合确定多个 ISP 之间的任务划分和每个 GDC 的任务服务率，最大化 DGDC 提供商的利润，并最小化所有应用程序的平均任务损失可能性。Zhu 等人^[11]提出了一种称为匹配多轮分配（MMA）的调度方法，以优化所有提交的任务在安全性和可靠性约束下的完工时间和总成本。本文讨论的大数据集调度问题不同于云虚拟机中的调度任务或数据中心中的虚拟机提供。差异是由 Spark 作业的性质和内存体系结构范例造成的。此外，Spark 作业的执行器可能无法安装到单个 VM 中，并且计划程序应选择不同类型的 VM，同时创建执行程序以满足资源约束。此外，工作负载性能也因放置策略而不同（传播或整合），的目标是在不提供有关工作负载和集群动态以及性能模型的任何先验知识的情况下训练调度程序。

2.2 框架调度程序

Apache Spark 默认使用（先进先出）FIFO 调度程序，它以分布式方式（分散）放置作业的执行器，以减少单个工作节点（如果考虑云部署，则为 VM）的开销。尽管此策略可以提高计算密集型工作负载的性能，但由于网络 shuffle 操作的增加，网络密集型工作负荷可能会受到性能开销的影响。Spark 还可以整合核心使用，以最小化集群中使用的节点总数。然而，它没有考虑 VM 的成本和作业的运行时间。因此，昂贵的虚拟机可能会使用更长的时间，从而产生更高的虚拟机成本。基于 Fair3 和 DRF^[12]的调度程序提高了集群中多个作业之间的公平性。然而，这些调度程序并不能提高 SLA 目标，例如云部署集群中的成本效率。对于不同的工作负载类型，不同的执行器放置策略是合适的，而框架调度器无

法支持这些策略。这就是在工作中所说的细粒度执行器放置。

2.3 性能模型和基于启发式的调度程序

有一些工作试图改进基于 Spark 的作业调度的不同方面。这些方法大多基于不同的工作负载和资源特征构建性能模型。然后将性能模型用于资源需求预测，或设计复杂的启发式算法来实现一个或多个目标。

Sparrow^[13]是一种分散式调度程序，它使用基于随机抽样的方法来提高默认 Spark 调度的性能。Quasar^[14]是一个群集管理器，它在满足用户提供的应用程序性能目标的同时，将群集的资源利用率降至最低。它使用协作过滤来发现不同资源对应用程序性能的影响。然后，这些信息用于有效的资源分配和调度。Morpheus^[15]根据历史跟踪估计作业性能，然后执行容器的打包放置，以最小化集群资源使用成本。此外，Morpheus 还可以动态重新调配失败的作业，以提高集群的整体性能。Justice^[16]使用具有历史作业执行跟踪的每个作业的 deadline 约束进行准入控制和资源分配。它还可以自动适应工作负载的变化，为每个作业提供足够的资源，从而满足最后期限的要求。OptEx^[17]根据分析信息对 Spark 作业的性能进行建模。然后，性能模型被用来组成一个经济高效的集群，同时只使用满足其最后期限所需的最小 VM 集部署每个作业。此外，假设每个作业具有相同的执行器大小，即 VM 的总资源容量。Maroulis 等人^[18]利用 DVFS 技术调整传入工作负载的 CPU 频率，以降低能耗。Li 等人^[19]还提供了一个能效调度器，其中算法假设每个作业具有相等的执行器大小，这相当于虚拟机的总资源容量。

性能模型和基于启发式的方法的问题是：（1）性能模型严重依赖于过去的的数据，由于集群环境中的各种变化，这些数据有时可能会过时（2）很难调整或修改基于启发式方法，以将工作负载和集群变化结合起来。因此，许多研究人员正在关注基于 RL 的方法，以更高效和可扩展的方式解决调度问题。

2.4 基于 DRL 的调度程序

Liu 等人^[20]为云资源分配开发了一个分层框架，同时减少了能耗和延迟降低。全局层使用 Q-learning 进行 VM 资源分配。相反，本地层为本地服务器使用基于 LSTM 的工作负载预测器和基于 RL 的无模型电源管理器。Wei 等人^[21]为云部署中的应用程序提出了一种支持 QoS 的作业调度算法。他们将 DQN 与目标网络和经验回放结合使用，以提高算法的稳定性。主要目标是提高平均作业响应时间，同时最大限度地提高 VM 资源利用率。DeepRM^[22]使用了 REINFORCE，一种策略梯度 DeepRL 算法，用于集群调度中的多资源打包。主要目标是最小化平均工作进度的减慢。然而，由于所有集群资源都被视为状态空间中的一大块 CPU 和内存，因此集群被认为是同质的。Decima^[23]还使用了策略梯度代理，其目标与 DeepRM 类似。在这里，代理和环境都是为了解决 Spark 中每个作业中的 DAG 调度问题而设计的，同时考虑相互依赖的任务。Li 等人^[24]考虑了一种基于 Actor Critical 的算法，用于处理 Apache Storm 中具有高可伸缩性的无界连续数据流。调度问题是将工作负载分配给特定的工作节点，而目标是减少平均端到端元组处理时间。这项工作还假设集群设置是同质的，并且不考虑成本效率。DSS^[25]是云计算环境中的一种自动化大数据任务调度方法，它结合了 DRL 和 LSTM，自动预测每个传入大数据作业应调度到的 VM，以提高大数据分析的性能，同时降低资源执行成本。Harmony^[26]是一个深度学习驱动的 ML 集群调度器，它将训练作业放置在最小化干扰和最大化平均作业完成时间的方式上。它使用基于 Actor Critical 的算法和具有工作意识的动作空间探索，并带有经验回放。此外，它还有一个奖励预测模型，该模型使用历史样本进行训练，并用于为不可见的位置产生奖励。Cheng 等

人^[27]将基于 DQN 的算法用于云计算中的 Spark 作业调度。这项工作的主要目标是优化带宽资源成本，同时最小化节点和链路能耗。Spear^[28]在同时考虑任务相关性和异构资源需求的同时，致力于最小化复杂 DAG 作业的完成时间。Spear 在任务调度中使用蒙特卡罗树搜索（MCTS），并训练 DRL 模型来指导 MCTS 中的扩展和推出步骤。Wu 等人^[29]提出了一种基于深度 CNN 和基于值函数的 Q-learning 的虚拟网络映射算法的最优任务分配方案。这里，任务被分配到适当的物理节点，目标是在满足任务要求的同时实现长期收入最大化。Thamsen 等人^[30]使用 Gradient Bandit 方法来提高 Spark 和 Flink 作业的资源利用率和作业吞吐量，其中 RL 模型学习共享资源上不同类型作业的协同位置优势。

3 本文方法

3.1 本文方法概述

本文中，学习代理是一种作业调度器，它在满足作业资源需求约束和虚拟机资源容量约束的同时，尝试在 Spark 集群中调度作业。它从环境中获得的回报与关键的调度目标直接相关，例如成本效率和平均工作持续时间的缩短。因此，通过最大化报酬，代理学习可以优化目标对象的策略。图 1 显示了提出的作业调度问题的 RL 框架。将所有组件视为集群环境的一部分（用大虚线矩形突出显示），调度程序除外。集群管理器监视集群的状态。它还控制工作节点（或云虚拟机）为任何作业放置执行器。在每个时间步骤中，调度代理都会从环境中获得一个观察结果，其中既包括当前作业的资源需求，也包括集群的当前资源可用性（由集群管理器的集群监控器指标显示）。操作是选择特定 VM 来创建作业的执行器。代理执行操作时，由集群管理员在集群中执行。然后，奖励生成器通过根据预定义的目标评估行动来计算奖励。请注意，在 RL 环境中，给予代理的奖励总是外部的。然而，RL 算法可以进行内部奖励（或参数）计算，并不断更新以找到更好的策略。

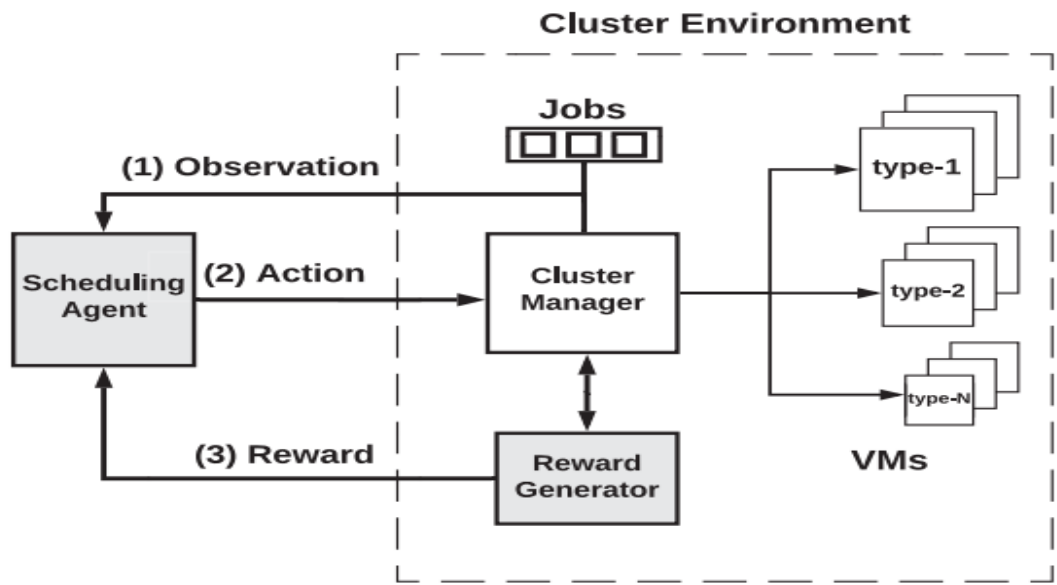


图 1: 作业调度问题的 RL 模型框架

3.2 RL 模型的关键组件

代理。代理充当调度程序，负责调度集群中的作业。在每个时间步骤，它都会观察系统状态并采取操作。根据动作，它会从环境中获得奖励和下一个可观察的状态。

事件。事件是从代理看到第一个作业和集群状态到完成调度所有作业的时间间隔。此外，如果代理选择了违反任何资源/作业限制的特定操作，则可以提前终止事件。

状态空间。在的调度问题中，状态是代理在执行每个操作后获得的观察结果。调度环境是已部署的集群，代理可以在其中观察每个操作后的集群状态。然而，只有在调度过程或事件开始时，代理才会收到初始状态，而不采取任何操作。集群状态具有以下参数：集群中所有 VM 的 CPU 和内存资源可用性、使用集群中每个 VM 的单价以及需要调度的当前作业规格。动作是代理（调度）为作业的每个执行器分配资源所做的决定或安排。每个执行器的资源分配被视为一个操作，在每个操作之后，环境将向代理返回下一个状态。环境的当前状态可以用一维向量表示，其中向量的第一部分是 VM 规格：，向量的第二部分是当前作业的规格：。这里是 $vm1\ cpu...vmN\ cpu$ 表示集群中所有 N 个 VM 的当前 cpu 可用性，而 $vm1\ mem...vmN\ mem$ 表示集群中所有 N 个 VM 的当前内存可用性。 $jobID$ 表示当前作业 ID， $e-cpu$ 和 $e-mem$ 分别表示当前作业的一个执行器的 CPU 和内存需求。由于一个作业的所有执行器都有相同的资源需求，因此唯一需要的其他信息是必须为该作业创建的执行器总数，它由 jE 表示。因此，状态空间仅随集群大小（VM 总数）的增加而增大，并不取决于作业总数。如果所有以前的作业都已安排好，则每个作业的规格仅在到达后作为状态的一部分发送给代理。在每次后续操作之后，集群资源容量将因执行器安排而更新。因此，下一个状态将反映最近操作后更新的集群状态。在放置当前作业的所有执行器之前，代理将继续接收当前作业的作业规格，唯一的更改是 jE 参数，在每次放置执行器后，该参数将减少 1。当它变为 0（作业已成功调度）时，只有这样，下一个作业规格才与更新的集群参数一起显示为下一个状态。请注意，在真实的 Spark 集群中，用户通过使用 $e-cpu$ （每个执行器的 cpu 内核数）、 $e-mem$ （每个执行器的内存数）和 $j-e$ （执行器总数）来指定作业的资源需求。然后，框架在为作业创建执行器时遵循此资源指定。因此，初始作业规格来自用户，而 VM 规格则由集群管理器提供。对于 RL 模型，这两个规格可以结合起来，在每次选择行动后提供一个环境观测状态。

行动空间。该操作是选择 VM，将在其中为当前作业创建一个执行器。如果集群没有足够的资源来放置当前作业的一个或所有执行器，那么代理也可以不做任何事情，等待先前计划的作业完成。此外，为了优化某个目标（例如成本、时间），代理可能会决定不在作业到达后立即安排作业。因此，如果集群中有 N 个 VM，则有 $N + 1$ 个可能的离散操作。这里，定义 Action 0 来指定代理将等待，并且不会创建执行器，其中 Action 1 到 N 指定了选择为当前作业创建执行器的 VM 的索引。

奖励。代理一旦采取行动，就会立即获得奖励。每一个行动都有积极或消极的奖励。积极的奖励激励代理采取良好的行动，并优化整个事件的整体奖励。相反，负面奖励通常会训练代理避免不良行为。在 RL 中，当想在事件结束时最大化总回报时，代理必须考虑立即和折扣后的未来回报才能采取行动。总体目标是最大化事件的累积奖励，因此有时会采取立即产生负面奖励的行动可能是朝着未来更大的正面奖励迈出的一大步。

3.3 DEEPRL 代理用于作业调度

为了解决所提出的 RL 环境中的作业调度问题，使用了两种基于 DRL 的算法。第一种是深度 Q 学习（DQN），这是一种基于 Q 学习的方法。另一种是策略梯度算法，称为 REINFORCE。选择这些算法是因为它们适用于具有离散状态和动作空间的 RL 环境。此外，这两种算法的工作过程不同，其中 DQN 优化状态动作值，而 REINFORCE 直接更新策略。从 Spark 作业调度上下文来看，RL 环境将提供类似于为实际工作负载运行的跟踪的作业规格。此外，集群资源也是相同的，因此 VM 资源可用性也将作为状态空间的一部分使用和更新。每次 DRL 代理采取行动（安排执行人）时，将提供即时

奖励。下一个状态也将取决于之前的状态，因为 VM 和作业规格将在每次放置后更新。最终，DRL 代理应该能够了解资源可用性和需求限制，并完成安排所有作业中的所有执行器来完成事件以获得事件奖励。

3.3.1 DQN 代理

DQN 是一种非策略算法，它在从环境中收集数据时使用不同的策略。原因是如果一直使用正在进行的改进策略，由于状态动作空间的覆盖不足，算法可能会偏离到次优策略。因此，使用 ϵ -贪心策略，选择概率为 $1-\epsilon$ 的贪心行为和概率为 ϵ 随机行为，以便可以观察到任何未探索的状态，从而确保算法不会陷入局部极大值。算法 1 总结了用于回放缓冲区和目标网络的 DQN[38] 算法。

Procedure 1 DQN Algorithm.

```
for iteration 1...N do
    Collect some samples from the environment by using the collect policy( $\epsilon$ -greedy), and store the samples in the replay buffer;
    Sample a batch of data from the replay buffer;
    Update the agent's network parameter  $u$ ;
end
```

3.3.2 REINFORCE 代理

使用 REINFORCE[39] 算法，如算法 2 所示。该算法通过使用蒙特卡罗滚动（通过执行一整集后计算奖励来学习）来工作。在收集步骤（第 2 行）之后，算法使用带有学习参数 α （第 4 行）的更新策略梯度更新底层网络。请注意，在对轨迹进行采样时，使用了贪心策略。

Procedure 2 REINFORCE Algorithm.

```
for iteration 1...N do
    Sample  $\tau_i$  from  $\pi_{\Theta}(a|s_t)$  by following the current policy in the environment;
    Find the policy gradient  $\Delta\Theta J(\Theta)$ ;
     $\Theta \leftarrow \Theta + \alpha \Delta\Theta J(\Theta)$ ;
end
```

3.4 RL 环境设计与实现

开发的环境的特点概括如下：

- 1) 环境在每个时间步向代理程序公开状态（包括最新的集群资源统计信息和下一个作业）。
- 2) 代理采取行动后，环境可以检测到有效/无效的安置，并相应地分配积极/消极奖励。
- 3) 根据代理在某一事件中的表现，环境可以奖励事件奖励（环境充当奖励生成器）。因此，对于具有工作负载跟踪的模拟集群，环境可以推导出查找事件奖励所需的成本和时间值。
- 4) 该环境考虑了由于公共云中的位置和争用，作业执行时间对在不同 VM 上放置执行器的影响，因为模拟环境使用的作业持续时间是通过在实验集群中运行实际作业从作业配置文件中收集的。
- 5) 环境也可以根据代理执行人安排的好坏来改变工作期限，并相应地将奖励分配给代理。

如前所述，不是表示实时的固定间隔；时间步长是指决策和行动的任意进展阶段。合并了 TF 代理 API 调用，以在每个时间步骤后返回转换或终止信号。图 2 显示了代理训练过程中的环境工作流。红色和绿色圆圈表示分别从环境中触发负面和正面奖励的事件。图 3 总结了“导致事件和奖励的行动”。在该图中，每个奖励的序号对应于图 2 所示的红色/绿色圆圈。所实施的环境可以与 TF 代理一起用于训练一个或多个 DRL 代理。具体来说，可以对代理进行训练，以实现一个或多个目标，如成本效率、性能改进。如前所述，设计了奖励信号，以实现成本效益和平均工作持续时间缩短。实施的环境可以

扩展或修改，以包含一个或多个奖励/目标、连续状态和训练其他 DRL 代理。

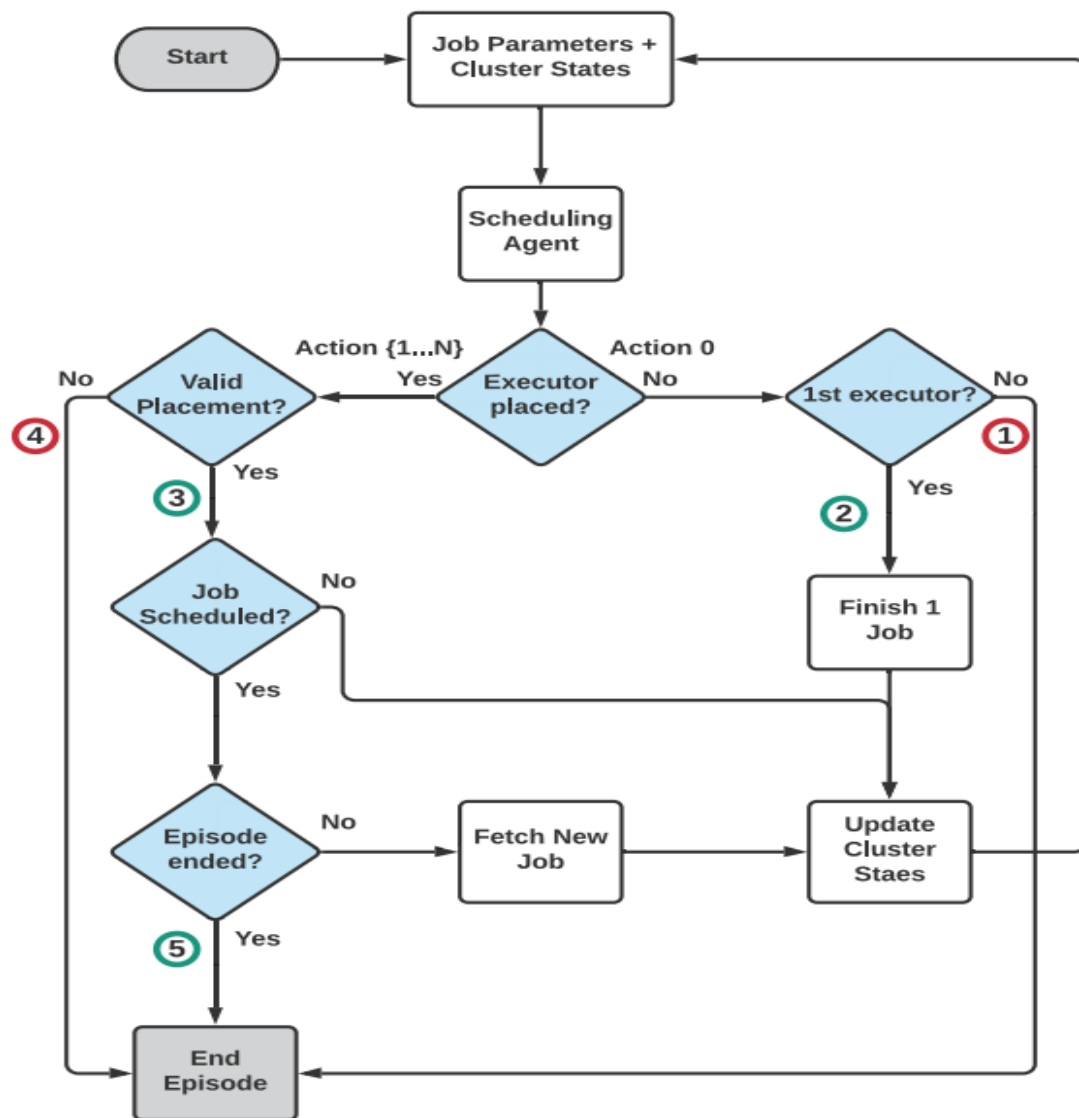


图 2: 代理训练过程中的环境工作流

No.	Action	Event	Reward
1	0	Previously placed 1 or more executors of the current job, but now waiting to place any remaining executor (s)	-200
2	0	No placement	-1
3	1 ... N	Proper placement of one executor of the current job	+1
4	1 ... N	Improper placement of an executor: resource capacity or resource demand constraints violation	-200
5	1 ... N	All jobs are scheduled, a successful completion of an episode	R_{epi} (Eqn. (14))

图 3: 导致事件和奖励的行为

4 复现细节

4.1 与已有开源代码对比

DQN 代理的代码对比图 4 和 5。


```

ain.py x DQN_tfagent.py x
def compute_avg_return(environment, policy, num_episodes=10):
    total_return = 0.0
    for _ in range(num_episodes):
        time_step = environment.reset()
        episode_return = 0.0

        while not time_step.is_last():
            action_step = policy.action(time_step)
            time_step = environment.step(action_step.action)
            episode_return += time_step.reward
            total_return += episode_return

    avg_return = total_return / num_episodes
    return avg_return.numpy()[0]

def collect_step(environment, policy, buffer):
    time_step = environment.current_time_step()
    action_step = policy.action(time_step)
    next_time_step = environment.step(action_step.action)
    traj = trajectory.from_transition(time_step, action_step, next_time_step)

    buffer.add_batch(traj)

def collect_data(env, policy, buffer, steps):
    for _ in range(steps):
        collect_step(env, policy, buffer)

def train_dqn(
    num_iterations=20000,
    initial_collect_steps=1000,
    collect_steps_per_iteration=1,
    replay_buffer_max_length=100000,
    fc_layer_params=(200,),
    batch_size=64,
    learning_rate=1e-3,
    log_interval=200,

```

图 4: DQN 代理的实现

```

dqn_train_eval.py x
19 root_dir = os.path.expanduser(root_dir)
20 train_dir = os.path.join(root_dir, 'train')
21 eval_dir = os.path.join(root_dir, 'eval')
22
23 train_summary_writer = tf.compat.v2.summary.create_file_writer(
24     train_dir, flush_millis=summaries_flush_secs * 1000)
25 train_summary_writer.set_as_default()
26
27 eval_summary_writer = tf.compat.v2.summary.create_file_writer(
28     eval_dir, flush_millis=summaries_flush_secs * 1000)
29
30 eval_metrics = [
31     tf_metrics.AverageReturnMetric(buffer_size=num_eval_episodes),
32     tf_metrics.AverageEpisodeLengthMetric(buffer_size=num_eval_episodes)
33 ]
34
35 global_step = tf.compat.v1.train.get_or_create_global_step()
36 with tf.compat.v2.summary.record_if(
37     lambda: tf.math.equal(global_step % summary_interval, 0)):
38     tf_env = tf_py_environment.TFPyEnvironment(ClusterEnv())
39     eval_tf_env = tf_py_environment.TFPyEnvironment(ClusterEnv())
40
41     if train_sequence_length != 1 and n_step_update != 1:
42         raise NotImplementedError(
43             'train_eval does not currently support n-step updates with stateful '
44             'networks (i.e., RNNs)')
45
46     if train_sequence_length > 1:
47         q_net = q_rnn_network.QRnnNetwork(
48             tf_env.observation_spec(),
49             tf_env.action_spec(),
50             input_fc_layer_params=input_fc_layer_params,
51             lstm_size=lstm_size,
52             output_fc_layer_params=output_fc_layer_params)
53     else:
54         q_net = q_network.QNetwork(
55             tf_env.observation_spec(),
56             tf_env.action_spec(),
57             fc_layer_params=fc_layer_params)
58     train_sequence_length = n_step_update

```

图 5: 原 DQN 代理的实现

图 4 为复现实现的代码，相比较图 5 更能实现论文中的要求，能够更好的实现平均奖励的计算和 TF 代理应用于 spark 调度。

REINFORCE 代理的代码对比图 6和 7。

```
def collect_episode(environment, policy, num_episodes, replay_buffer):
    episode_counter = 0
    environment.reset()

    while episode_counter < num_episodes:
        time_step = environment.current_time_step()
        action_step = policy.action(time_step)
        next_time_step = environment.step(action_step.action)
        traj = trajectory.From_transition(time_step, action_step, next_time_step)

        replay_buffer.add_batch(traj)

        if traj.is_boundary():
            episode_counter += 1

def compute_avg_return(environment, policy, num_episodes=10):
    total_return = 0.0
    for _ in range(num_episodes):
        time_step = environment.reset()
        episode_return = 0.0

        while not time_step.is_last():
            action_step = policy.action(time_step)
            time_step = environment.step(action_step.action)
            episode_return += time_step.reward
            total_return += episode_return

    avg_return = total_return / num_episodes
    return avg_return.numpy()[0]

def train_reinforce(
    num_iterations=20000,
    collect_episodes_per_iteration=2,
    replay_buffer_max_length=10000,
    fc_layer_params=(100,),
    learning_rate=1e-3,

```

图 6: REINFORCE 代理的实现

```
train_sequence_length = n_step_update

tf_agent = dqn_agent.DqnAgent(
    tf_env.time_step_spec(),
    tf_env.action_spec(),
    q_network=q_net,
    epsilon_greedy=epsilon_greedy,
    n_step_update=n_step_update,
    target_update_tau=target_update_tau,
    target_update_period=target_update_period,
    optimizer=tf.compat.v1.train.AdamOptimizer(learning_rate=learning_rate),
    td_errors_loss_fn=common.element_wise_squared_loss,
    gamma=gamma,
    reward_scale_factor=reward_scale_factor,
    gradient_clipping=gradient_clipping,
    debug_summaries=debug_summaries,
    summarize_grads_and_vars=summarize_grads_and_vars,
    train_step_counter=global_step)
tf_agent.initialize()

train_metrics = [
    tf_metrics.NumberOfEpisodes(),
    tf_metrics.EnvironmentSteps(),
    tf_metrics.AverageReturnMetric(),
    tf_metrics.AverageEpisodeLengthMetric(),
]

eval_policy = tf_agent.policy
collect_policy = tf_agent.collect_policy

replay_buffer = tf_uniform_replay_buffer.TFUniformReplayBuffer(
    data_spec=tf_agent.collect_data_spec,
    batch_size=tf_env.batch_size,
    max_length=replay_buffer_capacity)

collect_driver = dynamic_step_driver.DynamicStepDriver(
    tf_env,
    collect_policy,
```

图 7: 原 REINFORCE 代理的实现

图 6为复现实现的代码，相比较图 7更能实现论文中的要求，能够更好的实现平均奖励的计算和

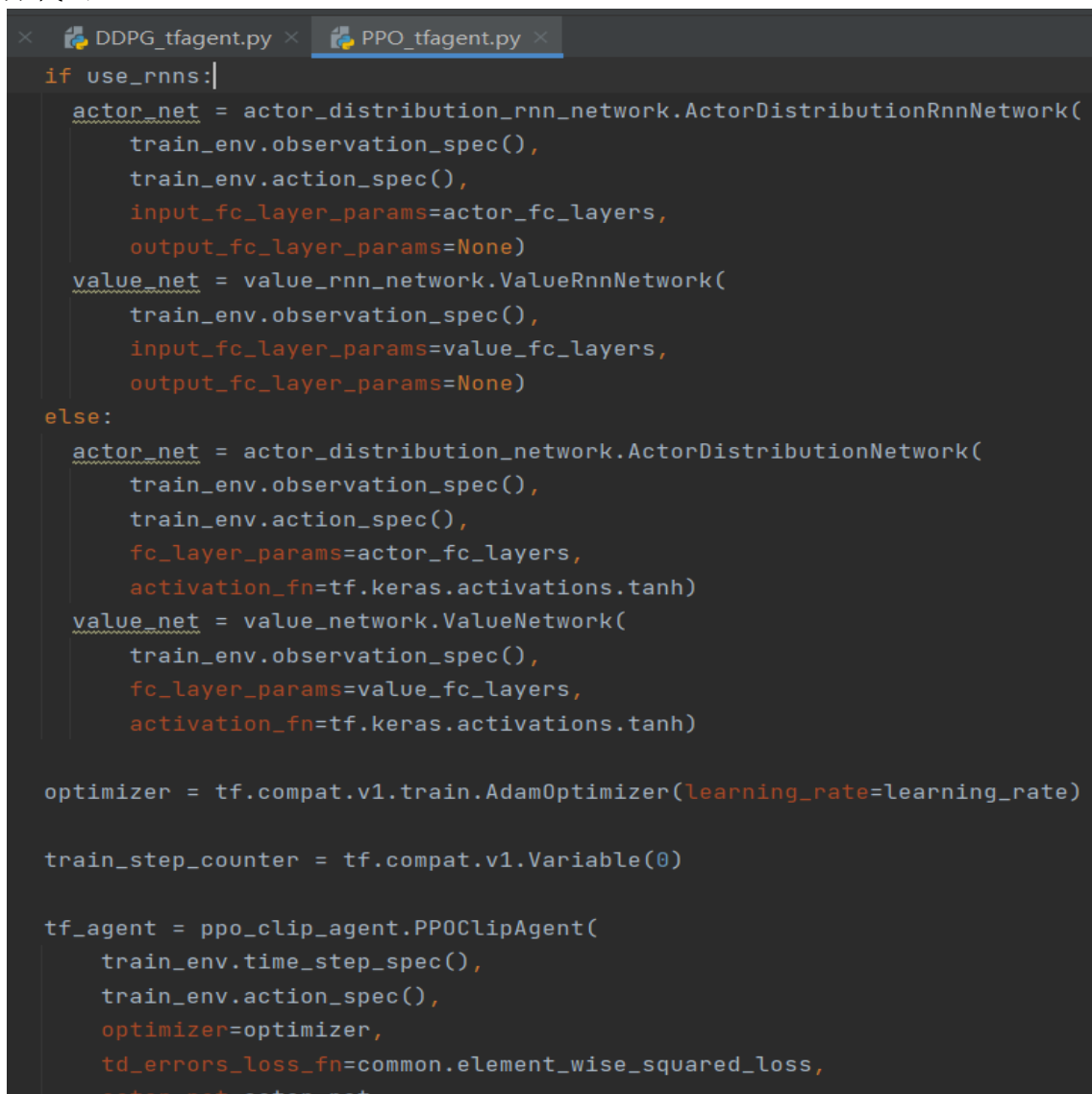
TF 代理应用于 spark 调度。

4.2 实验环境搭建

模拟集群资源。选择了具有不同定价模型的不同 VM 类型，以便可以训练和评估代理以优化成本。VM 机包括三种类型，每种类型 4 台机器，一共 12 台模拟机器。模拟工作量。选取了 3 个不同的应用程序作为集群中的作业，它们是：WordCount（CPU 密集型）、PageRank（网络或 IO 密集型）和 Sort（内存密集型）。使用了统一分布来生成 1-6（CPU 内核）、1-10（GB 内存）和 1-8（执行器总数）范围内的作业需求。模拟作业到达时间。选择了两种工作到达模式：正常（50 个工作在 1 小时内到达）和突发（100 个工作在 10 分钟内到达）。模拟集群环境。使用 Python 语言并使用 PycharmIDE 开发并实现了 RLSpark 调度的模拟，使用 Anaconda 导入 Python3.6 并使用 tensorflow2.0.0 和 tfagent0.5.0 作为强化学习的代理。

4.3 创新点

实现了两个新的算法并应用于搭建的 RL 模拟环境中，图 8 和 9 分别为 PPO 算法和 DDPG 算法的实现的部分代码。



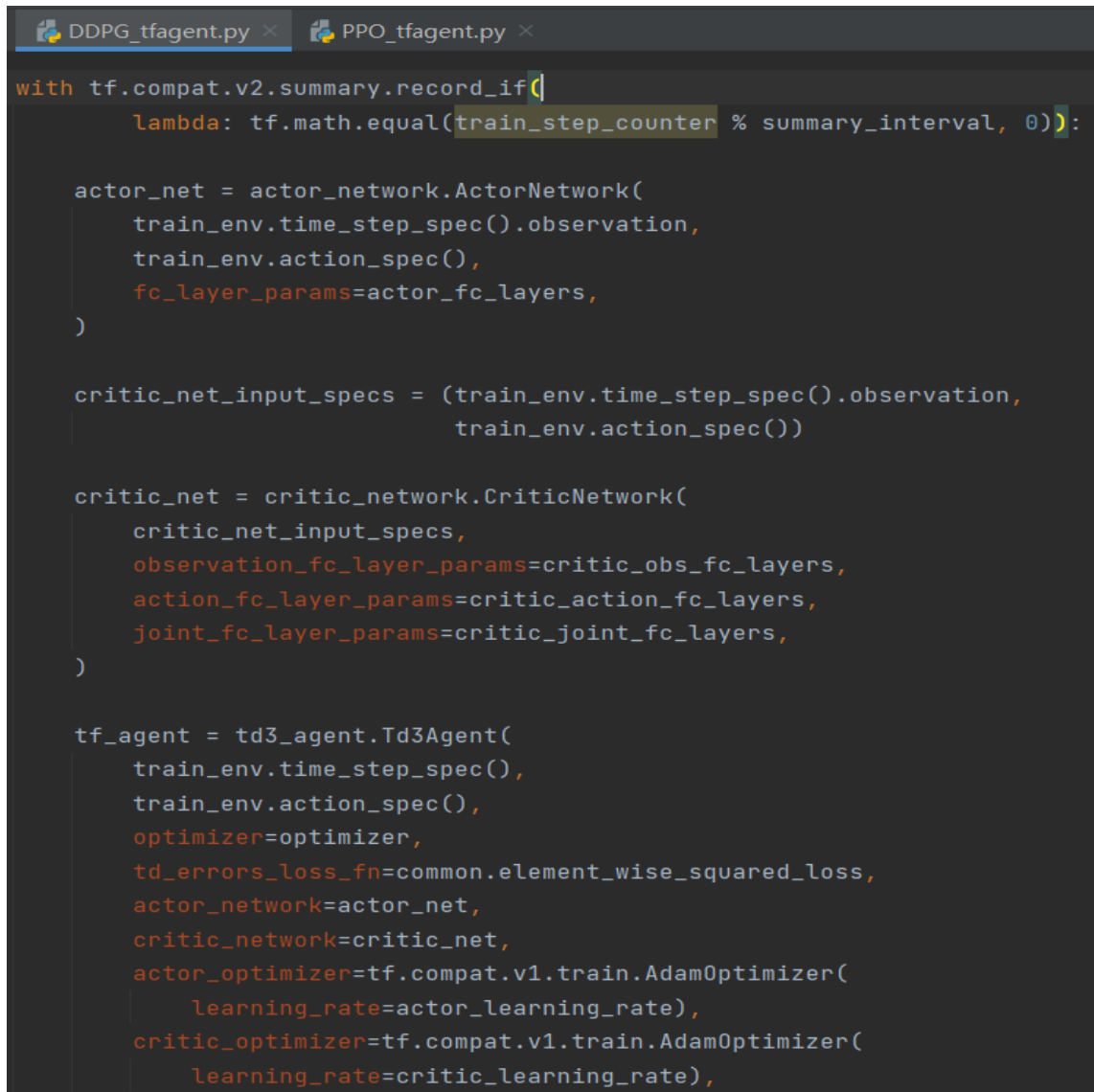
```
if use_rnns:
    actor_net = actor_distribution_rnn_network.ActorDistributionRnnNetwork(
        train_env.observation_spec(),
        train_env.action_spec(),
        input_fc_layer_params=actor_fc_layers,
        output_fc_layer_params=None)
    value_net = value_rnn_network.ValueRnnNetwork(
        train_env.observation_spec(),
        input_fc_layer_params=value_fc_layers,
        output_fc_layer_params=None)
else:
    actor_net = actor_distribution_network.ActorDistributionNetwork(
        train_env.observation_spec(),
        train_env.action_spec(),
        fc_layer_params=actor_fc_layers,
        activation_fn=tf.keras.activations.tanh)
    value_net = value_network.ValueNetwork(
        train_env.observation_spec(),
        fc_layer_params=value_fc_layers,
        activation_fn=tf.keras.activations.tanh)

optimizer = tf.compat.v1.train.AdamOptimizer(learning_rate=learning_rate)

train_step_counter = tf.compat.v1.Variable(0)

tf_agent = ppo_clip_agent.PPOClipAgent(
    train_env.time_step_spec(),
    train_env.action_spec(),
    optimizer=optimizer,
    td_errors_loss_fn=common.element_wise_squared_loss,
    actor_net=actor_net,
```

图 8: PPO 算法代理的实现



```

with tf.compat.v2.summary.record_if(
    lambda: tf.math.equal(train_step_counter % summary_interval, 0)):

    actor_net = actor_network.ActorNetwork(
        train_env.time_step_spec().observation,
        train_env.action_spec(),
        fc_layer_params=actor_fc_layers,
    )

    critic_net_input_specs = (train_env.time_step_spec().observation,
                              train_env.action_spec())

    critic_net = critic_network.CriticNetwork(
        critic_net_input_specs,
        observation_fc_layer_params=critic_obs_fc_layers,
        action_fc_layer_params=critic_action_fc_layers,
        joint_fc_layer_params=critic_joint_fc_layers,
    )

    tf_agent = td3_agent.Td3Agent(
        train_env.time_step_spec(),
        train_env.action_spec(),
        optimizer=optimizer,
        td_errors_loss_fn=common.element_wise_squared_loss,
        actor_network=actor_net,
        critic_network=critic_net,
        actor_optimizer=tf.compat.v1.train.AdamOptimizer(
            learning_rate=actor_learning_rate),
        critic_optimizer=tf.compat.v1.train.AdamOptimizer(
            learning_rate=critic_learning_rate),

```

图 9: DDPG 算法代理的实现

5 实验结果分析

图 10和图 11表示 DQN 算法的收敛性。图 12和图 13表示 REINFORCE 算法的收敛性。用不同的 β 参数值训练了 DRL 代理，以展示单个或多个奖励最大化的效果。算法的评估是在每 1000 次迭代之后进行的，在这里，计算经过训练的策略的 10 次测试运行的平均回报。对于正常的作业到达模式，对代理进行了 10000 次迭代的训练，对于突发作业到达模式来说，对这些代理进行了 20000 次迭代的训练。

β 值越高，表示代理因优化 VM 使用成本而获得的奖励越多。相反， β 值越低，表示代理的优化程度越高，以减少平均作业持续时间。将 β 的值从 0 更改为 1，其中值 1 表示代理仅针对成本进行优化。因此，在情景奖励中忽略了优化平均工作持续时间的奖励。相反， β 的值为 0 表示代理仅为缩短平均作业持续时间而优化。 β 的任何值（不包括 0 和 1）都表示混合操作模式，其中代理尝试优化具有不同优先级（值 0.25 和 0.75）或具有相同优先级（值 0.50）的两种奖励。请注意，事件奖励可能会有所不同，并根据集群资源状态、作业规格和到达率进行计算。此外，最终的事件奖励因不同的优化目标而异，因此不同的训练设置会为一个事件带来不同的最大奖励。图 10和图 11分别代表 DQN 代理在正常和突发工作到达模式的训练中累积的平均奖励。同样，图 12和图 13代表 REINFORCE 代理在训

训练迭代中的平均奖励累积。请注意，平均奖励由每个成功的执行器安置收到的固定奖励和最终的事件奖励组成，与实际 VM 使用成本或平均工作持续时间值不同。然而，累积更高的总报酬意味着代理人已经学会了更好的策略，可以优化实际目标。最初，两个代理都会收到消极的奖励，并在多次迭代探索状态空间后逐渐开始收到更多的奖励。由于贪婪引起的随机性，有时两种算法的奖励都会下降。然而，REINFORCE 代理的训练比 DQN 代理更稳定。由于有更多作业，这两个代理都需要更多的时间来与具有突发作业到达模式的工作负载聚合，并且当集群没有足够的资源来满足突发作业的资源需求时，代理必须学会等待（操作 0）。

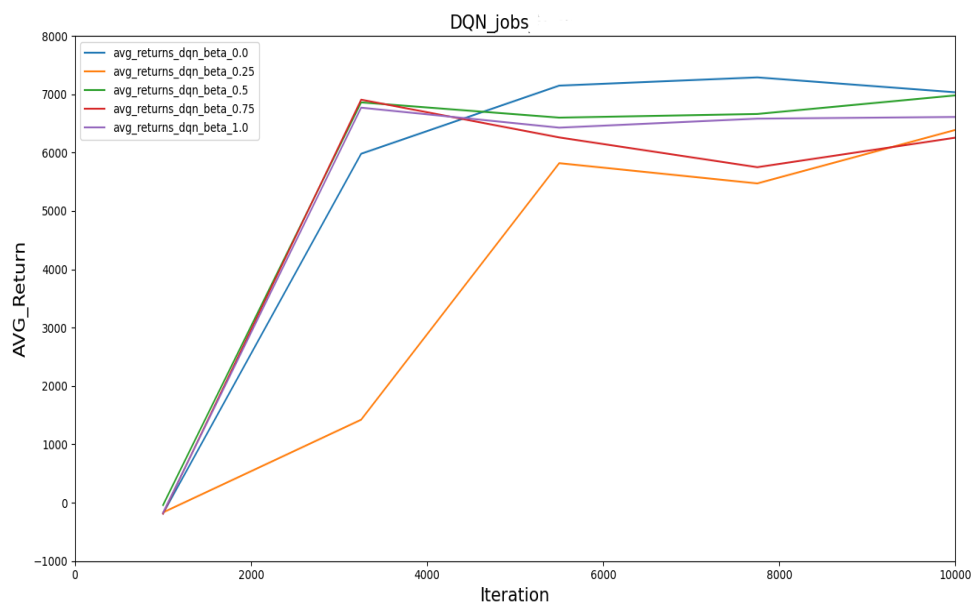


图 10: DQN 代理在正常工作到达模式的训练中累积的平均奖励

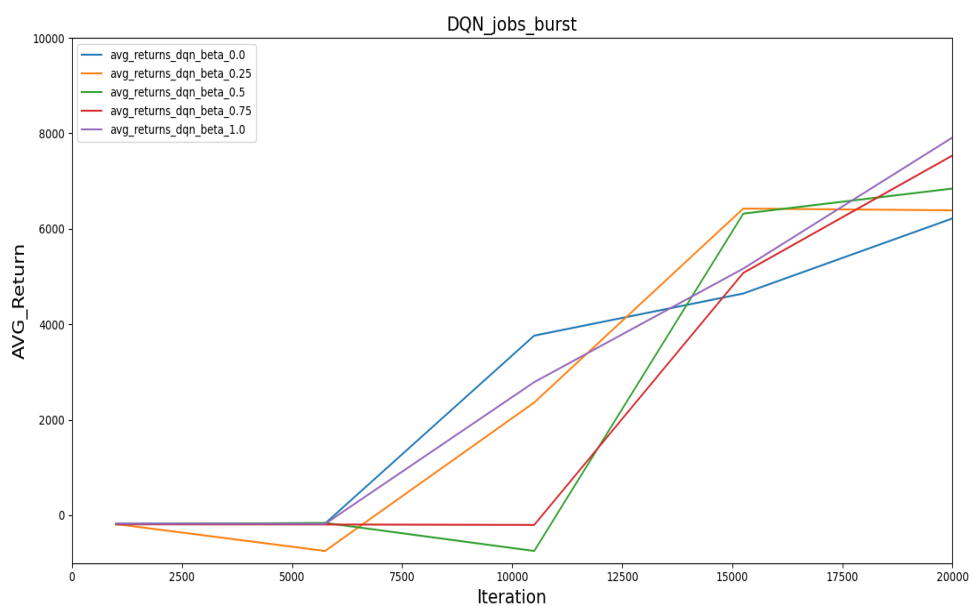


图 11: DQN 代理在突发工作到达模式的训练中累积的平均奖励

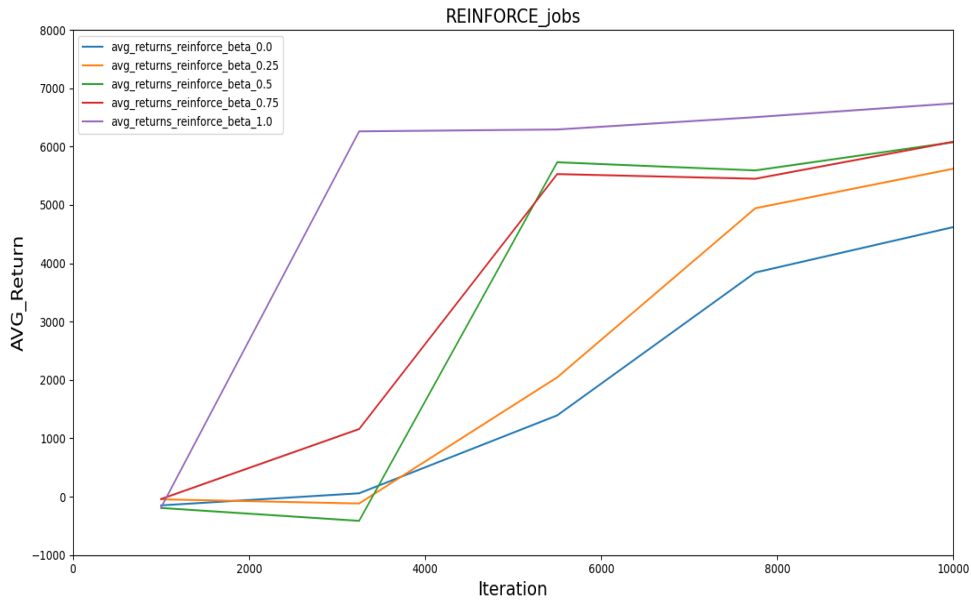


图 12: REINFORCE 代理在正常工作到达模式的训练中累积的平均奖励

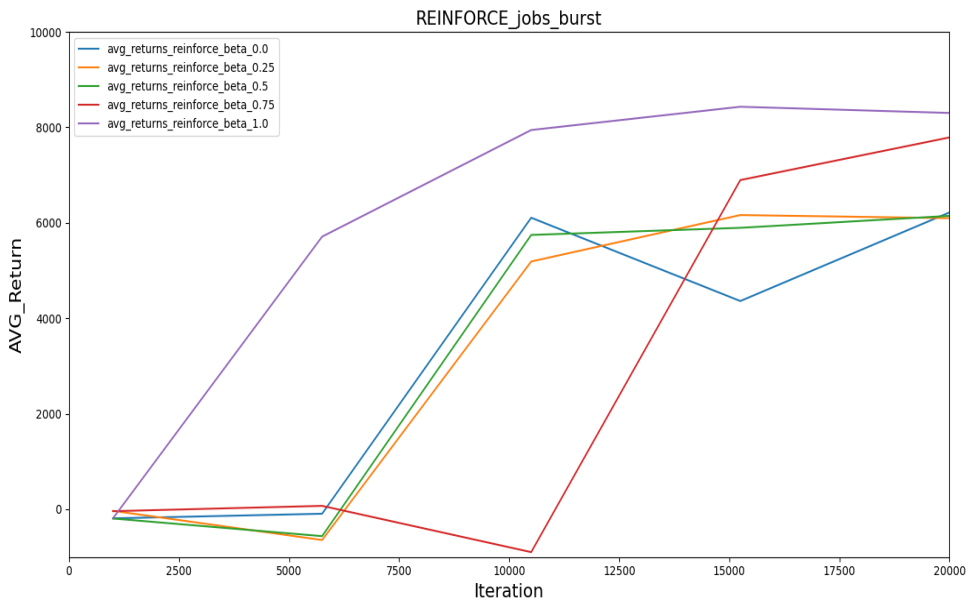


图 13: REINFORCE 代理在突发工作到达模式的训练中累积的平均奖励

6 总结与展望

云环境中大数据应用程序的作业调度是一个具有挑战性的问题，因为存在许多固有的 VM 和工作负载特征。传统的框架调度器、基于 LP 的优化和基于启发式的方法主要侧重于一个特定的目标，不能概括为在捕获或学习底层资源或工作负载特征的同时优化多个目标。本文介绍了云环境下 Spark 作业调度问题的 RL 模型。在 TF 代理中开发了一个原型 RL 环境，可用于训练基于 DRL 的代理以优化一个或多个目标。此外，还使用原型 RL 环境训练了两个基于 DRL 的代理，即 DQN 和 REINFORCE。设计了复杂的奖励信号，帮助 DRL 代理了解资源限制、作业性能可变性和集群 VM 使用成本。代理可以学习优化目标目标，而无需事先提供关于作业或集群的任何信息，但只能通过在与集群环境交互时观察即时和偶然的回报。已经证明，提出的代理在优化成本和时间目标的同时，性能优于基线算法，

并且在优化两个目标的同时表现出平衡的性能。还讨论了 DRL 代理发现的一些有效报酬最大化的关键策略。

在未来的工作中，将探讨不同工作的同处优势如何影响工作持续时间。还将研究复杂的奖励模型，该模型可以在即时奖励的情况下调整成本和工作持续时间。这样，代理将更高效地执行，也可以支持长时间运行的批处理作业。

还计划提取经过训练的策略，用于真正的大规模集群，以进一步训练代理。这将简化训练过程，代理在实际集群中部署时不会从头开始。这使能够调查 RL 代理是否能够了解集群动态的任何新变化或特征，以更有效地优化目标。

参考文献

- [1] Et AL. V K V. Apache Hadoop YARN: Yet another resource negotiator[J]. Proc. 4th ACM Annu. Symp. Cloud Comput., 2013: 1-6.
- [2] Et AL. M Z. Apache spark: A unified engine for big data processing[J]. Commun. ACM, 2016, 59(11): 56-65.
- [3] SHVACHKO K, KUANG H, RADIA S, et al. The hadoop distributed file system[J]. Proc. 26th IEEE Symp. Mass Storage Syst. Technol., 2010: 1-10.
- [4] GEORGE L, HBase. The Definitive Guide: Random Access to Your Planet-Size Data[J]. Newton and MA and USA: O' Reilly Media and Inc., 2011.
- [5] LAKSHMAN A, MALIK P. Cassandra: A decentralized structured storage system[J]. ACM SIGOPS Oper. Syst. Rev., 2010, 44(2): 35-40.
- [6] Et AL. D S. Mastering the game of go without human knowledge[J]. Nature, 2017, 550: 354-359.
- [7] CAO Z, LIN C, ZHOU M, et al. Scheduling semiconductor testing facility by using cuckoo search algorithm with reinforcement learning and surrogate modeling[J]. IEEE Trans. Automat. Sci. Eng., 2019, 16(2): 825-837.
- [8] JIANG L, HUANG H, DING Z. Path planning for intelligent robots based on deep Q-learning with experience replay and heuristic knowledge[J]. IEEE/CAA J. Automatica Sinica, 2020, 7(4): 1179-1189.
- [9] YADWADKAR N J, HARIHARAN B, GONZALEZ J E, et al. Selecting the best VM across multiple public clouds: A data-driven performance modeling approach[J]. Proc. Cloud Comput. and New York and NY and USA, 2017: 452-465.
- [10] YUAN H, BI J, ZHOU M, et al. Biobjective task scheduling for distributed green data centers[J]. IEEE Trans. Automat. Sci. Eng., 2021, 18(2): 731-742.
- [11] YUAN H, ZHOU M, LIU Q, et al. Fine-grained resource provisioning and task scheduling for heterogeneous applications in distributed green clouds[J]. IEEE/CAA J. Automatica Sinica, 2020, 7(5): 1380-1393.

- [12] GHODSI A, ZAHARIA M, HINDMAN B, et al. Dominant resource fairness: Fair allocation of multiple resource types[J]. Proc. 8th USENIX Conf. Netw. Syst. Des. Implementation, 2011: 323-336.
- [13] OUSTERHOUT K, WENDELL P, ZAHARIA M, et al. Sparrow[J]. Proc. 24th ACM Symp. Oper. Syst. Principles, New York, NY, USA, 2013: 69-84.
- [14] DELIMITROU C, KOZYRAKIS C. Quasar: Resource-efficient and QoS-aware cluster management[J]. Proc. 19th Int. Conf. Archit. Support for Program. Lang. Oper. Syst., 2014: 127-144.
- [15] Et AL. S A J. Morpheus: Towards automated slos for enterprise clusters[J]. Proc. 12th USENIX Conf. Operating Syst. Des. Implementation, 2016: 117-134.
- [16] DIMOPOULOS S, KRINTZ C, WOLSKI R. Justice: A deadline-aware and fair-share resource allocator for implementing multi-analytics[J]. Proc. IEEE Int. Conf. Cluster Comput., 2017: 233-244.
- [17] SIDHANTA S, GOLAB W, MUKHOPADHYAY S. OptEx: A deadline-aware cost optimization model for spark[J]. Proc. 16th IEEE/ACM Int. Symp. Cluster and Cloud Grid Comput., 2016: 193-202.
- [18] MAROULIS S, ZACHEILAS N, KALOGERAKI V. A framework for efficient energy scheduling of spark workloads[J]. Proc. Int. Conf. Distrib. Comput. Syst., 2017: 2614-2615.
- [19] LI H, WANG H, FANG S, et al. An energy-aware scheduling algorithm for big data applications in Spark [J]. Cluster Comput. J., 2020, 23: 593-609.
- [20] Et AL. N L. A hierarchical framework of cloud resource allocation and power management using deep reinforcement learning[J]. Proc. 37th IEEE Int. Conf. Distrib. Comput. Syst., 2017: 372-382.
- [21] WEI Y, PAN L, LIU S, et al. DRL-Scheduling: An intelligent QoS-aware job scheduling framework for applications in clouds[J]. IEEE Access, 2018, 6: 55112-55125.
- [22] MAO H, ALIZADEH M, MENACHE I, et al. Resource management with deep reinforcement learning [J]. Proc. 15th ACM Workshop Hot Top. Netw., 2016: 50-56.
- [23] MAO H, SCHWARZKOPF M, VENKATAKRISHNAN S B, et al. Learning scheduling algorithms for data processing clusters[J]. SIGCOMM Proc. Conf. ACM Special Interest Group Data Commun., 2019: 270-288.
- [24] LI T, XU Z, TANG J, et al. Model-free control for distributed stream data processing using deep reinforcement learning[J]. Proc. VLDB Endowment, 2018, 11(6): 705-718.
- [25] RJOUB G, BENTAHAR J, WAHAB O A, et al. Deepsmart scheduling: A deep learning approach for automated big data scheduling over the cloud[J]. Proc. Int. Conf. Future Internet Things Cloud, 2019: 189-196.
- [26] BAO Y, PENG Y, WU C. Deep learning-based job placement in distributed machine learning clusters [J]. Proc. IEEE INFOCOM Conf. Comput. Commun., 2019: 505-513.

- [27] CHENG Y, XU G. A novel task provisioning approach fusing reinforcement learning for big data[J]. IEEE Access, 2019, 7: 143699-143709.
- [28] HU Z, TU J, LI B. Spear: Optimized dependency-aware task scheduling with deep reinforcement learning[J]. Proc. Int. Conf. Distrib. Comput. Syst., 2019: 2037-2046.
- [29] WU C, XU G, DING Y, et al. Explore deep neural network and reinforcement learning to large-scale tasks processing in big data[J]. Int. J. Pattern Recognit. Artif. Intell., 2019, 33(13): 1-29.
- [30] THAMSEN L, BEILHARZ J, TRAN V T, et al. Mary and Hugo and Hugo*: Learning to schedule distributed data parallel processing jobs on shared clusters[J]. Concurrency Comput., 2020, 33(18): 1-12.