分布式强化学习综述

邹宇豪[1)]

[1)](华中科技大学计算机学院 武汉市 中国 430074)

摘 要 分布式强化学习是一种结合了强化学习与分布式计算的方法，旨在解决大规模、高维度强化学习问题。强化学习系统可以通过同时利用 GPU 加速的并行性和大量的分布式工作节点来进行可扩展性优化，在设计时应当考虑执行抽象，分布策略，加速支持和算法抽象四个方面。本文讨论了当前的分布式强化学习系统，分为基于函数、基于执行者和基于数据流三类，从执行抽象，分布策略，加速支持和算法抽象四个方面分析了这三类系统的各自特点。文章后续介绍了一种新的分布式强化学习系统，MSRL，它支持分布策略，控制强化学习训练过程如何并行化和分布在集群资源上，而不需要改变算法实现。MSRL 引入了一种名为数据流片段图的新抽象，它将 Python 函数从强化学习算法的训练循环映射到并行计算片段。通过将片段转换为低级数据流表示，在不同的设备上执行片段。

关键词 强化学习；深度强化学习；分布式学习

An Overview of Distributed Reinforcement Learning

Yuhao Zou[1)]

[1)](Huazhong University of Science and Technology, Wuhan, China,430074)

Abstract Distributed Reinforcement Learning is an approach that combines reinforcement learning and distributed computing to solve large-scale, high-dimensional reinforcement learning problems. Reinforcement learning systems must be scalable by exploiting both the parallelism of GPU acceleration and a large number of distributed worker nodes, and should be designed with four aspects in mind: execution abstraction, distribution policies, acceleration support and algorithm abstraction. This paper discusses the current distributed reinforcement learning systems, which are classified into three categories, namely function-based, actor-based, and dataflow-based, and analyses the respective characteristics of these three categories in terms of execution abstraction, distribution policies, acceleration support, and algorithmic abstraction. The paper follows up with the introduction of a new distributed reinforcement learning system, MSRL, which supports distribution policies that govern how RL training computation is parallelized and distributed on cluster resources, without requiring changes to the algorithm implementation. MSRL introduces the new abstraction of a fragmented dataflow graph, which maps Python functions from an RL algorithm ’ s training loop to parallel computational fragments. Fragments are executed on different devices by translating them to low-level dataflow representations.

Key words Reinforcement Learning; Deep Reinforcement Learning; Distributed Learning

1 引言

强化学习（Reinforcement Learning，RL）可以解决决策问题，智能体可以通过不断学习通常以深度神经网络（DNN）的形式表示的策略，以了解在未知环境中如何行动[1]。目前，强化学习在复杂的现实世界中取得了显著的成果：在围棋领域，AlphaGo [2]在棋盘游戏中击败了世界冠军；在生物学领域，AlphaFold [3]预测了蛋白质折叠的三维结构；在机器人技术领域，机器人可以在没有人类干

预的情况下通过基于强化学习的方法完成灵巧操作等任务。

但强化学习的发展伴随着对训练的计算需求的增加：AlphaStar 使用 384 个 TPU 和 1800 个 CPU，耗费超过 44 天的时间，训练 12 个智能体在星际争霸 2 中达到大师水平；OpenAI Five 训练了 10 个月，使用了 1536 个 GPU 和 172800 个 CPU，在 Dota2 中击败了 99.4%的人类玩家。因此，分布式强化学习系统必须在智能体数量、环境生成的训练数据量以及训练策略的复杂性方面进行扩展。

分布式强化学习旨在通过分解、并行处理、经验共享、异步更新和参数共享等手段，解决传统强化学习方法在处理大规模、高维度问题时面临的挑战。其基本思想是利用计算资源的优势，提高算法的效率和可扩展性，为解决复杂问题提供新的思路和方法。现有的分布式强化学习系统的设计，如 SEED RL [4]，ACME [5]，Ray [6]，RLlib [7] 和 Podracer[8]，是根据它们的算法结构，硬编码一个单一策略来并行化和分布强化学习训练算法。例如，如果一个强化学习算法被定义为一组针对智能体、学习者和环境的 Python 函数，那么系统可以直接调用智能体的 act()函数的实现来为环境生成新的操作。

2 分布式强化学习

2.1 强化学习

强化学习是机器学习的一种范式。强化学习的框架如图 1 所示，其主要包含智能体(agent)和环境(environment)两部分。强化学习的运行是智能体与环境两者不断交互的过程，其中：环境为智能体提供当前状态(state)和数值奖励(reward)，而智能体根据已有信息(通常是当前状态)向环境输出动作(action)，环境再给出执行动作后的状态和奖励。如此循环。直到任务终止(done)。在这一过程中。智能体往往以最大化期望累积奖励为目标进行动作选择和策略学习。

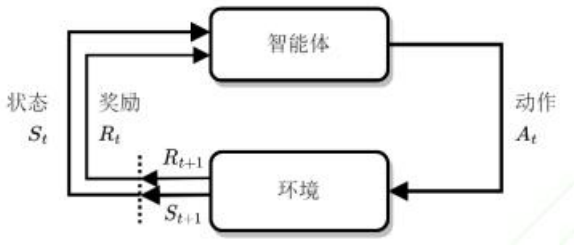

图 1 强化学习框架

强化学习的环境模型普遍基于马尔可夫决策过程(Markov decision process，MDP)构建。MDP 由一个四元组$\langle S,A,R,T\rangle$定义，其中，S是环境状态集合，A是可选动作集合，状态转移函数 $T:S\times A\times S\to[0,1]$给出由状态 s 和动作 a 转移到状态 s'的概率，奖励函数$R:S\times A\times S\to[0,1]$给出每一步的奖励数值。

智能体强化学习算法需要给出一个策略 π，策略在每个状态 s 决定动作 a 的执行与否(确定性策略)或执行概率(非确定性策略)。经典的强化学习算法认为环境的 MDP 模型是事先给定的，策略 π的优化目标为最大化期望累积折扣奖励。设参数化策略 π_θ的参数为 θ，则算法计算最优参数 θ^*的公式为：

$$\theta^*=\arg\max_\theta E_{\pi_\theta}\left[\sum_{t=0}^{T}\gamma^t r_t\right]$$

其中，T 指代环境运行的时间步数; 折扣因子 $\gamma\in[0,1]$用于权衡长期奖励与短期奖励，在 T 过大的环境中能显著稳定强化学习算法。

智能体强化学习算法主要分为基于值函数、基于策略梯度两类。

基于值函数的强化学习算法依据状态-动作值函数$Q^{\pi}(s,a)$决策。$Q^{\pi}(s,a)$指在状态s下执行动作a后，继续依据策略π决策的期望累积奖励值。因此，最优动作a^{*}即为状态s下取最大$Q^{\pi}(s,a)$的动作a：

$$a^{*}=\arg\max_{a}Q^{\pi}(s,a)|s$$

$Q^{\pi}(s,a)$的计算则基于期望累积折扣奖励的表达式，以贝尔曼最优方程(Bellman optimality equation)进行迭代，具体如下:

$$Q_{k+1}^{\pi}(s,a)=E_{s'}\left[r+\gamma\max_{a'}Q_{k}^{\pi}\left(s',a'\right)|s,a\right]$$

在基于值函数的深度强化学习算法中，$Q^{\pi}(s,a)$由神经网络构建，并辅以一些设计以增强算法稳定性。该类算法在离散动作环境中表现更好，但难以扩展到连续动作环境，常用算法有深度Q网络(deep Q-network，DQN)[9]，使用深度神经网络（DNN）来近似值函数，即，在一个状态下执行一个给定操作的预期返回的映射。智能体学习操作的值，并根据这些估计值选择操作。

基于策略梯度的强化学习算法直接对策略函数$\pi_{\theta}(a|s)$建模并优化。同样从期望累积折扣奖励的优化目标出发，策略参数θ的梯度为

$$\nabla_{\theta}J(\theta)=E_{\pi}[Q^{\pi}(s,a)\nabla_{\theta}log\pi_{\theta}(a|s)]$$

上式可以利用REINFORCE算法[10]近似估计，其中，$Q^{\pi}(s,a)$由真实的采样轨迹(trajectory)计算，但这样计算$Q^{\pi}(s,a)$的方差很大，使策略难以提升。演员-评论家(actor-critic，AC)框架将$Q^{\pi}(s,a)$作为独立的“评论家”模块学习，评论家模块为“演员”策略提供参考。经验表明，该方法可显著提升策略训练效果。基于策略梯度的常用算法均为演员-评论家架构，该类算法在连续动作环境中表现更好，包括深度确定性策略梯度算法(deep deterministic policy gradient，DDPG)[11]、近端策略优化算法(proximal policy optimization，PPO)[12]、演员-评论家算法(Advantage Actor-Critic，A2C)[13]等。

多智能体强化学习（MARL）有多个智能体，当与环境或其他智能体交互时，每个智能体都会优化自己的累积奖励。A3C[13]在不同的环境实例上异步执行智能体；MAPPO[14]将PPO算法扩展到多智能体，其中智能体共享全局参数化策略。在训练过程中，每个智能体维护一个中央批评者，它将联合信息（例如状态/动作对）作为输入，同时学习一个只依赖于本地状态的策略。

2.2 对于分布式强化学习系统的需求

强化学习的好处是以牺牲其计算成本为代价的[15]。复杂的环境和设置需要探索大的动作空间、状态和DNN参数，并且这些空间随着智能体的数量呈指数级增长。因此，强化学习系统必须通过同时利用GPU加速的并行性和大量的分布式工作节点来进行可扩展性优化。

分布式强化学习是一种结合了强化学习与分布式计算的方法，旨在解决大规模、高维度强化学习问题。它通过将强化学习任务分解为多个子任务，并在多个计算节点上进行并行处理，有效地利用计算资源，提高算法的效率和可扩展性。在分布式强化学习中，各个计算节点独立地学习和更新策略，同时通过共享经验和奖励信息来协同工作。具体来说，每个智能体在其自身的环境中进行探索和交互，收集数据和经验，并在本地进行学习。这些智能体之间通过通信机制共享经验和策略参数，从而在全局范围内进行优化和协同学习。

目前有一系列关于强化学习系统如何并行化和分发强化学习训练的建议：对于单智能体强化学习，环境执行，策略推理和训练可以分布在工作节点上[8]，可能使用GPU[4]；对于多智能体强化学习，智能体可以使用通信库进行分布[7]和交换训练状态。通常，环境实例可以并行[8]或分布执行，以加快执行速度。

对于所有可能的强化学习算法，上述并行化和分发强化学习训练的策略，无论是在获得最低的迭代时间方面还是在具有最佳的可扩展性方面都不是最优的。训练瓶颈会随着不同的算法、训练工作负载和硬件资源发生变化：例如，我们的测量表明，对于PPO算法，环境执行占用高达98%的执行时间；对于MuZero[18]，一个具有许多智能体的大型多智能体强化学习算法，环境执行不再是瓶颈，97%的时间花在策略推理和训练上。

分布式强化学习系统设计应具有根据工作量改变执行方法的灵活性，而不是硬编码一个针对并行化和分布式强化学习计算的特定方法。这使得我们的设计旨在满足以下要求：

1) 执行抽象。强化学习系统应该有一个灵活的执行抽象，使它能够并行化和分发计算，而不受算法定义方式的影响。虽然这种执行抽象在基于编译的深度学习引擎中很常见，但它们在当前的强化学习系统中并不存在。

2) 分布策略。强化学习系统应该支持不同的策略来跨工作节点分配强化学习计算任务。应该允许用户为单个强化学习算法指定多个分布策略，并根据训练工作量在它们之间进行切换，而不需要改变算法的实现。分布策略应该适用于不同类别的强化学习算法。

3) 加速支持。强化学习系统应该利用设备的并行性，如多核 CPU、GPU 或其他 AI 加速器。小至细粒度的向量化的执行，大到粗粒度进程并行的任务，强化学习系统都应该在 CPU 上提供支持。加速不应仅局限于策略训练和推理，而应涵盖完整的训练循环，包括环境执行[19]。

4) 算法抽象。尽管将执行与算法规范解耦，但用户仍希望使用熟悉的 API 来定义围绕算法组件[20]的算法，如智能体、执行者、学习者、策略和环境等。因此，强化学习系统应该提供标准的 API，例如根据策略推理、环境执行和策略训练来定义主要的训练循环。

3 现有方案

为了了解现有的强化学习系统是如何支持上述需求的，我们调查了强化学习系统的设计空间。现有的设计可分为基于函数、基于执行者和基于数据流三类（见图 2）：

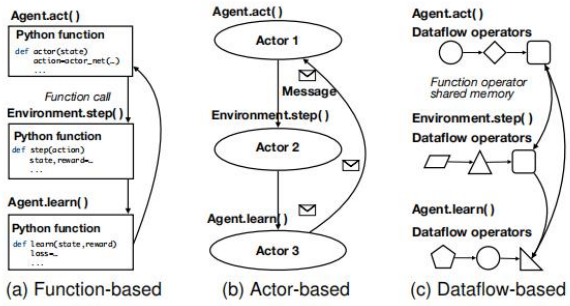

图 2 现有强化学习系统设计

3.1 基于函数的强化学习系统

基于函数的强化学习系统是最常见的类型。它们通常将强化学习算法表示为由工作节点直接执行的 Python 函数（见图 2a）。强化学习的训练循环是直接调用函数来实现的。例如，Acme [5]和 SEED RL[4]将算法组织为执行者/学习者函数；RLGraph [17]使用组件抽象，用户注册 Python 回调来定义功能。分布式执行被委托给后端引擎，例如 TensorFlow、PyTorch、Ray[6]。

3.2 基于函数的强化学习系统

基于执行者的强化学习系统通过部署在工作节点上的一组执行者之间的消息传递来执行算法（参见图 2b)。Ray [6]将算法定义为执行者模型中的并行任务。任务使用远程调用分布在节点之间。然而，在完全分布式模型中定义控制流是一种负担。为了克服这个问题，RLlib[7]在 Ray 之上添加了逻辑上的集中控制。类似地，MALib[21]为使用 Ray 作为后端的基于人群的多智能体强化学习算法提供了一个高级的智能体/评估者/学习者抽象。

3.3 基于函数的强化学习系统

基于数据流的强化学习系统通过数据并行操作符来定义算法，这些操作符被映射到 GPU 内核或分布式任务（见图 2c）。操作符通常是预先定义的，用户必须从一组固定的 API 中进行选择。例如，Podracer[8]使用 JAX 来编译子程序作为数据流操作符，以便在 TPU 上进行分布式执行。WarpDrive[19]将数据流操作符定义为 CUDA 内核，并使用 GPU 线程块执行完整的强化学习训练循环。RLlib Flow[16]使用分布式数据流操作符，作为具有消息传递的迭代器实现。

3.4 现有强化学习系统特点

3.4.1 执行抽象

基于函数和执行者的系统分别通过实现的（Python）函数和用户定义的（编程语言）执行者直接执行强化学习算法。这阻止了系统对强化学习算法进行并行化或分布式的优化。相比之下，基于数据流的系统使用预定义/编译的操作符[8]或 CUDA 内核[19]来执行计算，这提供了优化的机会。

3.4.2 分布策略

大多数基于函数的系统都采用一种固定的策略，即只用一个学习者来分配执行者，以实现环境执行的并行化。基于执行者的系统通常使用一个贪婪的调度器将有状态的执行者和无状态的任务分配到多个工作节点之间，而没有特定于领域的规划。基于数据流的系统通常硬编码数据流操作符如何分配给工作节点。Podracer[8]提供了两种策略：Anakin 在每个 TPU 核心上共同托管一个环境和一个智能体；Sebulba 将环境、学习者和执行者分配到不同的 TPU 核心上；RLlib Flow 将数据流操作符分散到分布式 Ray 行动者中。

3.4.3 加速支持

大多数强化学习系统只加速 DNN 策略推断和训练。一些基于数据流的系统（例如，Podracer[8] 和 WarpDrive[19]）也会加速训练的其他部分，这需要自定义数据流实现：例如，Podracer 可以加速 TPU 核心上的环境执行；WarpDrive 使用 CUDA 在单个 GPU 上执行整个训练循环。

3.4.1 算法抽象

基于函数的强化学习系统可以轻松地提供直观的执行者/学习者/环境 API。基于执行者的强化学习系统通常为分布式组件提供低级的难以使用的 API（例如 Ray 的获取/等待/远程[6]）；高级库（例如 RLlib 的策略优化器 API [7]）试图弥补这一差距。基于数据流的系统带有固定的数据流操作符，要求用户重写算法。例如，JAX 为向量化提供了 vmap 操作符，为单程序多重数据（SPMD）并行化提供了 pmap。

虽然将强化学习算法的定义与其执行策略整合在一起简化了设计和实施，但这意味着系统无法以最有效的方式并行化和分布强化学习算法。

3.5 基于数据流片段图的强化学习系统

目前有一个新的强化学习系统[22]，该系统结合了基于函数的系统的可用性和基于数据流的系统的加速潜力。这种设计需要一个新的执行抽象，该执行抽象保留了在强化学习算法规范之上应用不同的加速和分发策略的灵活性。

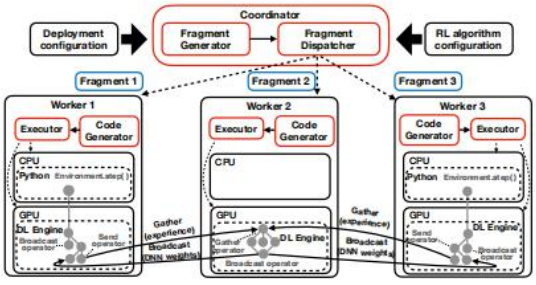

图 3 MSRL 体系结构

MSRL 的体系结构遵循协调器/工作节点的设计（见图 3）：

1) 用户将强化学习算法实现提交给协调器；

2) 协调器生成数据流片段图（Fragmented Dataflow Graphs）并将片段发送给节点；

3) 节点为接收到的片段生成可执行代码，这些代码可以由深度学习引擎运行。

它将强化学习算法的规范与其执行解耦。这使得 MSRL 能够改变其并行化和分发不同强化学习算法的方式，以减少迭代时间，并增加训练的可伸缩性。为了实现这一点，MSRL 的设计做出了以下贡献：

1) 与执行无关的算法规范。在 MSRL 中，用户通过以传统的方式将其算法组件（如智能体、执行者、学习者）实现为 Python 函数来指定一个强化学习算法。这个实现没有假设算法将如何执行：组件之间的所有运行时交互都通过调用 MSRL API 来管理。一个单独的部署配置定义了可供执行的设备。

2) 数据流片段图抽象。从强化学习算法的实现中，MSRL 构建了一个数据流片段图（FDG）。FDG 是计算的数据流表示，它允许 MSRL 在部署时将算法映射到异构设备（CPU 和 GPU）。该图由片段组成，这些片段是强化学习算法的一部分，可以并行化和跨设备分布。MSRL 对算法实现使用静态分析，将其功能分组为片段，其实例可以部署在设备上。片段之间的边界是在算法实现中通过用户提供的分区注释来确定的。注释指定了同步点，需要在不同设备之间复制的片段之间进行通信，以增加数据并行性。

3) 使用深度学习引擎执行片段。为了执行，MSRL 部署了特定于硬件的片段实现，允许实例单独或共同在 CPU 和 GPU 上运行。MSRL 支持不同的片段实现：CPU 实现使用普通的 Python 代码，而 GPU 实现则是在片段使用运算符或直接使用 CUDA 实现的情况下，作为深度学习引擎的编译计算图生成。MSRL 融合了多个数据并行片段，以提高执行效率。

4) 分布策略。由于算法实现是通过 FDG 与执行相分离的，因此 MSRL 可以应用不同的分发策略来管理如何将片段映射到设备。MSRL 支持一套灵活的分发策略，其中包括当前强化学习系统的硬编码分发策略：例如，分布策略可以分配多个执行者以扩展与环境的交互（如 Acme [5]）；分布执行者但分配策略推理给学习者（如 SEED RL [4]）；同时分布执行者和学习者（如 Sebulba [8]）；以及在单个 GPU 上表示完整的强化学习训练循环（如 WarpDrive [19]和 Anakin [8]）。由于算法的配置、其超参数或硬件资源的变化，MSRL 可以在不同的分布策略之间进行切换，以保持较高的训练效率，而不需要改变强化学习算法的实现。

4 总结

强化学习在许多领域取得了重大成就，但往往伴随着巨大的计算成本。分布式强化学习是一种结合了强化学习与分布式计算的方法，旨在解决大规模、高维度强化学习问题。强化学习系统可以通过同时利用 GPU 加速的并行性和大量的分布式工作节点来进行可扩展性优化。不同的强化学习训练算法为分布和并行化计算提供了不同的机会，分布式强化学习系统设计应具有根据工作量改变执行方法的灵活性，而不是硬编码一个针对并行化和分布式强化学习计算的特定方法，在设计时应当考虑执行抽象，分布策略，加速支持和算法抽象四个方面。

本文讨论了当前的分布式强化学习系统，分为基于函数、基于执行者和基于数据流三类，从执行抽象，分布策略，加速支持和算法抽象四个方面分析了这三类系统的各自特点，研究显示它们对特定的分布策略进行硬编码，并且只加速 GPU 工作节点的计算的特定部分（例如策略网络更新）。从根本上说，这三类系统缺乏将强化学习算法与执行解耦的抽象。文章后续介绍了一种新的分布式强化学习系统，MSRL，它支持分布策略，控制强化学习训练过程如何并行化和分布在集群资源上，而不需要改变算法实现。MSRL 引入了一种名为数据流片段图的新抽象，它将 Python 函数从强化学习算法的训练循环映射到并行计算片段。通过将片段转换为低级数据流表示，在不同的设备上执行片段。

参 考 文 献

[1] Sutton R S , Barto A G .Reinforcement Learning: An Introduction[J].IEEE Transactions on Neural Networks, 1998, 9(5):1054.DOI:10.1109/TNN.1998.712192.

[2] Silver D , Huang A , Maddison C J ,et al.Mastering the game of Go with deep neural networks and tree search[J].Nature[2023-12-29].DOI:10.1038/nature16961.

[3] [3]Jumper J , Evans R , Pritzel A ,et al.Highly accurate protein structure prediction with AlphaFold[J].Nature, 2021:1-11.DOI:10.1038/s41586-021-03819-2.

[4] Espeholt L ,Marinier, Raphaël, Stanczyk P ,et al.SEED RL: Scalable and Efficient Deep-RL with Accelerated Central Inference[J].arXiv, 2019.DOI:10.48550/arXiv.1910.06591.

[5] Hoffman M , Shahriari B , Aslanides J ,et al.Acme: A Research Framework for Distributed Reinforcement Learning[J]. 2020.DOI:10.48550/arXiv.2006.00979.

[6] Moritz P , Nishihara R , Wang S ,et al.Ray: a distributed framework for emerging AI applications[C]//Operating Systems Design and Implementation.USENIX Association, 2018.DOI:10.5555/3291168.3291210.

[7] Liang E , Liaw R , Moritz P ,et al.RLlib: Abstractions for Distributed Reinforcement Learning[J]. 2017.DOI:10.48550/arXiv.1712.09381.

[8] Hessel M , Kroiss M , Clark A ,et al.Podracer architectures for scalable Reinforcement Learning[J]. 2021.DOI:10.48550/arXiv.2104.06272.

[9] Volodymyr M , Koray K , David S ,et al.Human-level control through deep reinforcement learning[J].Nature, 2015, 518(7540):529-33 页.DOI:10.1038/nature14236.

[10] Williams R J .Simple statistical gradient-following algorithms for connectionist reinforcement learning[J].Machine Learning, 1992, 8(3-4):229-256.DOI:10.1007/BF00992696.

[11] Paul L T , James H J , David S ,et al.CONTINUOUS CONTROL WITH DEEP REINFORCEMENT LEARNING[J]. 2018.

[12] Schulman J , Wolski F , Dhariwal P ,et al.Proximal Policy Optimization Algorithms[J]. 2017.DOI:10.48550/arXiv.1707.06347.

[13] Mnih V, Badia A P, Mirza M, et al. Asynchronous methods for deep reinforcement learning[C]//International conference on machine learning. PMLR, 2016: 1928-1937.

[14] Yu C , Velu A , Vinitsky E ,et al.The Surprising Effectiveness of MAPPO in Cooperative, Multi-Agent Games[J]. 2021.DOI:10.48550/arXiv.2103.01955.

[15] Ceron J O , Castro P S .Revisiting Rainbow: Promoting more insightful and inclusive deep reinforcement learning research[C]//International Conference on Machine Learning.PMLR, 2021.

[16] Liang E , Wu Z , Luo M ,et al.RLlib Flow: Distributed Reinforcement Learning is a Dataflow Problem[J]. 2020.DOI:10.48550/arXiv.2011.12719.

[17] Schaarschmidt M , Mika S , Fricke K ,et al.RLgraph: Modular Computation Graphs for Deep Reinforcement Learning.2018[2023-12-29].

[18] Schrittwieser J , Antonoglou I , Hubert T ,et al.Mastering Atari, Go, chess and shogi by planning with a learned model[J].Nature[2023-12-29].DOI:10.1038/s41586-020-03051-4.

[19] Lan T , Srinivasa S , Zheng S .WarpDrive: Extremely Fast End-to-End Deep Multi-Agent Reinforcement Learning on a GPU[J]. 2021.DOI:10.48550/arXiv.2108.13976.

[20] Gronauer S, Diepold K. Multi-agent deep reinforcement learning: a survey[J]. Artificial Intelligence Review, 2022: 1-49.

[21] Zhou M , Wan Z , Wang H ,et al.MALib: A Parallel Framework for Population-based Multi-agent Reinforcement Learning[J]. 2021.DOI:10.48550/arXiv.2106.07551.

[22] Zhu H, Zhao B, Chen G, et al. {MSRL}: Distributed Reinforcement Learning with Dataflow Fragments[C]//2023 USENIX Annual Technical Conference (USENIX ATC 23). 2023: 977-993.

附录.

为什么 MSRL 比其他的分布式强化学习系统更灵活？或者说，其他 RL 系统的缺陷在哪里？

根据文中的描述,现有的三种强化学习(RL)系统的主要缺点如下:

1. (function-based):算法实现与执行高度耦合,系统无法灵活地并行化和分布执行算法的不同部分。

算法使用 Python 函数直接实现和执行,系统无法对算法进行优化和转换。硬编码了某种并行化/分布策略,无法根据工作负载变化调整，例如只分布 actor 来加速环境交互。只支持模型训练的 GPU 加速,其他部分依然顺序执行。

这些缺点的根本原因是缺乏执行抽象,算法定义方式限制了执行的灵活性。

2. (actor-based):完全分布式的 actor 模型编程过于繁琐,控制流难以定义。

完全分布式的 actor 编程模型过于复杂,控制流难以定义。依赖底层系统的调度,无法根据领域信息进行规划。只支持模型训练的 GPU 加速。

这些缺点的原因是采用了低级的 actor 抽象,忽略了 RL 算法的特点,增加了编程难度。

3. (dataflow-based):算法必须使用预定义的操作符,用户需重写算法逻辑。并行度和分布策略都是硬编码的。

用户必须使用预定义且固定的 operator 集合来重写算法。只能硬编码某种并行化/分布策略,无法调整。这是因为数据流抽象过于限制,缺乏灵活性,用户无法自然地表达算法。以上系统的共同缺陷在于没有抽象出算法定义与执行之间的界限,导致执行无法根据工作负载进行调优。MSRL 的优势在于找到了这样的抽象。

相比上述系统,MSRL 的优势在于:

MSRL 能够实现灵活的执行和分布的主要措施如下:

1. 算法定义与执行分离

用户用组件化方式定义算法,不涉及执行。训练循环通过调用 MSRL 的交互 API 实现。部署配置区分开算法定义和执行设备。

这样保证了变更执行策略不需要改动算法定义。

2. 引入碎片化数据流图(FDG)抽象

通过静态分析和用户标注,识别算法的切分点,构建数据流图。在切分点将图分解为碎片,表示潜在的并行化/分布代码段。碎片间通过同步接口通信,接口根据切分点的标注选择实现。

FDG 抽象解耦了算法定义和并行/分布执行。

3. 支持多种碎片执行方案

CPU 上为 Python 函数,GPU 上为计算图或 CUDA。融合同设备的碎片实例,使用数据流引擎的 SIMD 加速。计算图实现可以自动优化,CUDA 可以充分利用 GPU。

多种执行保证了跨设备的灵活执行。

4. 定义多种分布策略

策略基于 FDG 对碎片和通信进行不同的分配。可根据算法、模型、资源变化切换策略。策略空间涵盖了现有系统的固定策略。

分布策略实现了跨节点的灵活分布。

通过这些关键措施,MSRL 实现了 RL 训练的灵活性,同时保持了易用性。