12 并行计算

基于强化学习低采样效率的问题，并行计算作为解决方案可以高效地加速模型训练过程并提高学习效果。在本章中，我们将具体介绍强化学习中采用并行计算的系统架构。对应不同的应用场景，我们分别分析同步通信和异步通信，并详细阐述并行计算在多种网络拓扑结构中的不同运算方式。通过并行计算，经典的分布式强化学习算法和架构将被逐一介绍并互相比较。最终，我们将总结一般性的分布式计算架构的基本构成和组成元素。

12.1 简介

在深度强化学习中，针对模型的训练需要大量数据。以 OpenAI Five (OpenAI et al., 2019) 为例，为了使智能体能够通过学习在 Dota 游戏中做出明智的决策，每两秒钟就大概有 2 百万组数据被用来训练模型。不仅如此，从优化的角度，特别在基于策略梯度的方法中，大批量的训练数据能够有效地降低结果的方差。然而，由于在强化学习中，智能体和环境的交互限制于在时间上顺序执行，强化学习的算法在采集数据上往往存在低效率的问题，从而带来不理想的训练结果和缓慢的收敛速度。并行计算，即对相互分离独立的任务以同时计算的方式，为解决强化学习问题带来有效的解决方案。一般来说，并行计算中的并行性可以体现在以下两个方面：

- **计算的并行性**: 数据计算是包括特征工程、模型学习，以及结果评估等任务在内的核心过程，是由每一个计算单元具体操作执行的，不同的操作单元可以根据任务的大小和种类灵活地结合并扩展到不同的规模。在同等级规模的任务中，计算的性能和表现取决于以下两类策略：一类是合并多个计算单元共同计算一个任务，另一类是分别使用多个计算单元同时并行计算多个任务。基于第一类计算策略，随着越来越多的计算单元用于一个计算任务，完成任务的效率会先上升，然后会因为一些瓶颈的环节而逐渐收敛。因而在深度强化学习

中，在有计算资源充足的前提下，将一个计算任务拆分成多个相互独立的子任务，并且将每个子任务分配适当的计算资源进行并行计算，是寻求计算效率提升的重要方向。

- **数据传输的并行性**: 在拥有充足的计算资源时，计算资源之间的数据传输会成为解决问题效率的瓶颈。一般来说，为了避免传输的过多冗余，平衡网络中传输的数据量并且降低传输延时，基于不同的应用提出了不同的数据传输网络拓扑模型。在并行计算中，由于多个进程或线程可能同时需要完成不同的任务。管理数据的传输并且在有限传输带宽的网络中，保证传输效率是极具挑战的。

在监督学习的设定中，一种简单的提升学习速度的方法是同时训练多种不同的训练数据。然而，在深度强化学习中，智能体和环境需要在时间上顺序多次交互来逐步获得有效信息，因而不可能把所有数据集合在一起让模型同时学习。在深度强化学习中提高并行计算的能力可以通过让智能体在训练中同时并行学习多个训练轨迹，或者可以积累批量的数据来训练深度强化学习模型中的参数。在本章中，我们即从计算的并行性和数据传输的并行性的角度分析深度强化学习中可以采取并行计算的方面，并且在解决大规模深度强化学习问题的同时，我们将介绍当前重要的分布式计算的算法和架构。

12.2 同步和异步

在并行计算中，最普及的数据计算和传输方法采用类似星形的拓扑结构，是由一个主节点和多个奴隶节点组成的。主节点整体上管理数据信息，完成从奴隶节点的数据分发和收集。基于从奴隶节点收集到的数据，总体的网络参数将得到学习并更新。每一个奴隶节点，相应地，会从主节点收到分配的数据，进行具体的数据计算，并将计算的结果提交给主节点。由于在主节点的管理下，同时可以有多个奴隶节点进行数据计算。这样的并行计算可以合作高效地完成大规模模型参数训练问题。

星形结构在解决深度强化学习的问题中得到了广泛的应用，例如，在 Actor-Critic 方法的并行计算版本，通常会采用一个主节点，以及多个奴隶节点。每个奴隶节点会维护一个深度策略网络，该策略网络的结构和其他奴隶节点和主节点一样。因此，奴隶节点在初始化的时候会从主节点，同步策略网络的参数，然后其将独立与环境交互学习。在与环境交互之后，奴隶节点将再次和主节点通信，将学习到的信息提交给主节点，其中学习到的信息基于不同的架构可以是单步探索所得到的经验、连续探索的轨迹的经验、存储的带有权重的探索经验、网络参数的梯度信息，等等。在收集到每个奴隶节点探索并学习的经验和反馈之后，主节点将更新其网络的参数，并同步给所有奴隶节点，用于奴隶节点下一轮的探索。

星形拓扑结构清晰地将任务细分，通过多个奴隶节点的并行计算加速了智能体对策略的学习。然而基于不同的计算能力，不同奴隶节点完成探索并收集经验的时间可能不会完全相同，因而制定数据传输的模式会因所解决的问题和系统架构的不同而有所变化。一般来说，其分为同步通信和异步通信两种模式。

同步通信模式如图 12.1 所示，其中红色的区间代表数据通信所使用的时间，蓝色的区间表示与环境交互和数据计算所使用的的时间。值得注意的是，在同步通信模式中，所有奴隶节点将使用完全相同的时间区间进行信息交互。主节点将用相同固定的时间段同时与所有奴隶节点进行通信。然而，有更强算力的奴隶节点不得不等待其他所有弱算力的节点完成本轮计算任务之后才能继续和主节点通信，因而同步通信模式虽然对主节点来说在收集分析奴隶节点计算结果的时间分配上更为清晰固定，但是在奴隶节点中会有大量计算资源因为等待同步而造成浪费。

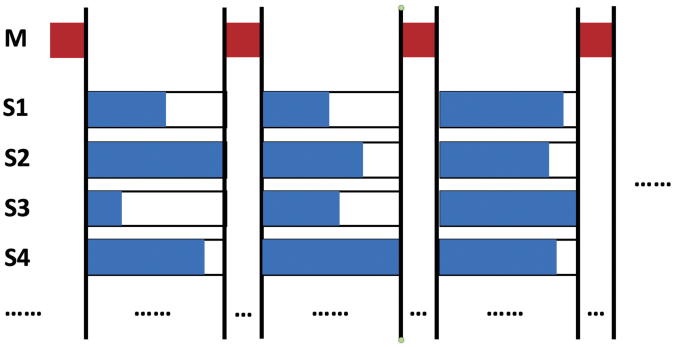

图 12.1　同步通信（见彩插）

为了减少奴隶节点的等待时间，提升计算资源的使用效率，异步通信模式相应被提出。如图 12.2 所示，只要奴隶节点完成了本轮的计算或探索任务，可以立刻将信息提交给主节点。只有奴隶节点提交信息，主节点才会用收集到的信息更新网络参数并与该奴隶节点进行同步，使其奴隶节点能够继续开始下一轮的计算或探索任务。基于此种方式，主节点和众多奴隶节点的数据通信将会在多段不同的时间区间内完成，主节点需要不定时地与不同的奴隶节点进行通信，确保奴隶节点中的计算资源得到了充分的利用。

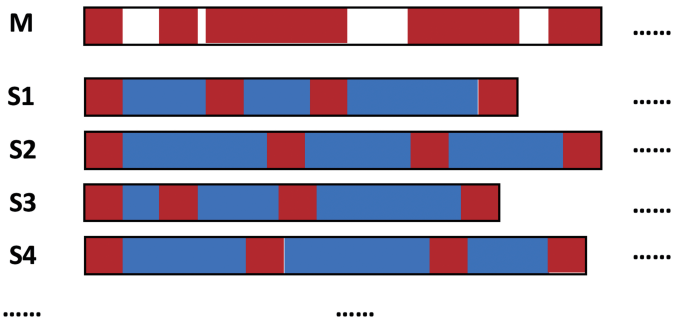

图 12.2　异步通信（见彩插）

12.3 并行计算网络

星形拓扑结构采用了集中控制式并行计算的方式，其中存在一个主节点管理并维护整个系统，使其确保所有分布式计算的任务能够进行得井井有条。然而，从另一个方面来说，主节点同样也是整个系统最薄弱的部分，为了确保计算的高性能，主节点需要在处理数据上比奴隶节点更具效率。另外，由于所有奴隶节点可能需要将信息同时传输给主节点，为了避免过大的传输延时，主节点的数据传输带宽需要足够大。不仅如此，系统的稳定性将大大取决于主节点，如果主节点有任何的停机事件，整个系统将停止工作，即使所有的奴隶节点存在大量可用的计算资源。

综上所述，对于很多对稳定性和大规模并行计算有需求的应用来说，拥有一个分布式数据计算和通信的结构十分必要。我们假设有多个相互独立的进程，其中每个进程会建立并维护一个结构相同的深度学习网络，并且进程与进程之间会保持频繁的通信，以使网络参数保持同步。由于每个进程需要与所有其他进程通信，当进程的数量增加时，进程间通信的成本将指数级上升。为了降低冗余的通信并高效地实现信息同步，具有消息传递接口（Message Passing Interfaces，MPI）的进程间通信（Inter-Process Communication，IPC）方式将被采纳。一般来说，MPI 提供基本的进程进行数据发送，广播和接收的接口标准，基于这些标准，不同的通信结构得以进一步提出并由此提高信息交流效率。一些经典的通信结构举例如下。

- **树形结构通信**: 设系统中有 N 个进程，当其中一个进程希望将其自身的网络参数信息广播给其他 $N-1$ 个进程时，可以通过如图 12.3 所示的树形结构通信。在树形通信结构中，该进程首先会将信息发送给其周围 $m-1$ 个进程，然后在下一步迭代中，其周围的 $m-1$ 个进程均会将信息并行发送新的 $m-1$ 个不同的进程。因此，随着并行通信数目的增加，该进程仅需要 $\lceil \log_m N \rceil$ 次迭代即可将信息发送给所有其他 $N-1$ 个进程，其中每个进程只需要完成 $(m-1)\lceil \log_m N \rceil$ 次通信。相比于星形结构中每个进程都需要把信息分别发送给所有其他的进程，树形结构通信通过提高迭代次数使用并行通信的方式大大降低了所有进程发送信息的总数。

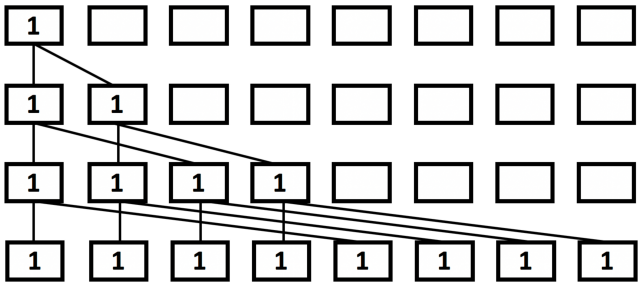

图 12.3 树形结构通信

- **蝴蝶形结构通信**: 当所有 N 个进程需要同时将自身的信息广播给所有其他进程时，每个进程将可以通过树形结构的形式进行通信，从而叠加为蝴蝶形通信结构。如图 12.4 所示，蝴蝶形通信结构中，每个进程首先将信息发送给其周围 $m-1$ 个进程，并同时收集并处理其他进程发来的信息用于下一次迭代中信息的发送。由于每个进程会在每次迭代中收集并处理其他进程发来的信息，所以其分布式的信息发送和处理效率得到了进一步的提升。一般来说，所有进程需要 $(m-1)\lceil \log_m N \rceil$ 次迭代完成所有信息的通信，并且每个进程一共只需要完成 $(m-1)\lceil \log_m N \rceil$ 次通信。另外，在这个系统中无论其中哪一个进程出现故障中断，其他进程之间仍可以继续完成信息的同步而不受到影响。

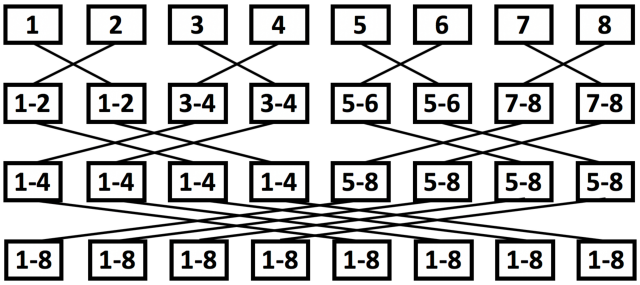

图 12.4　蝴蝶形结构通信

基于不同结构的通信，深度强化学习算法中并行数据计算和数据传输的方式是灵活且多样的。对于不同的应用，为了提高并行性和处理效率，深度强化学习算法的架构也会相应地有所调整，在下一节中，我们将进一步分析并总结深度强化学习中一般性的分布式计算架构。

12.4　分布式强化学习算法

12.4.1　异步优势 Actor-Critic

异步优势 Actor-Critic（Asynchronous Advantage Actor-Critic，A3C）(Mnih et al., 2016) 是基于优势 Actor-Critic（Advantage Actor-Critic，A2C）算法的分布式版本，如图 12.5 所示，多个行动-学习者（Actor-Learner）将与多个独立且完全相同的环境交互，并采用 A2C 算法学习并更新网络参数，因而每个行动-学习者都需要维护一个策略网络和一个价值网络来指导其在与环境的交互中采取明智的决策动作。为了使所有的行动-学习者的网络参数初始化相同并保持同步，在所有行动-学习者之外将建立参数服务器，并使其支持对所有行动-学习者的异步通信。

从每一个行动-学习者的角度，其具体的学习算法如算法 12.37 所述，在每个学习周期中，通过异步通信，每个行动-学习者首先会从参数服务器中同步其网络参数，基于更新的策略网络，行动-学习者会做出决策动作并与环境最多交互 $t_{\max}$ 次，与环境交互探索的经验会被收集并用来训练其自身的策略网络和价值网络，分别得到两个网络参数的更新梯度 $\mathrm{d}\theta$ 和 $\mathrm{d}\theta_v$。在行动-学习者

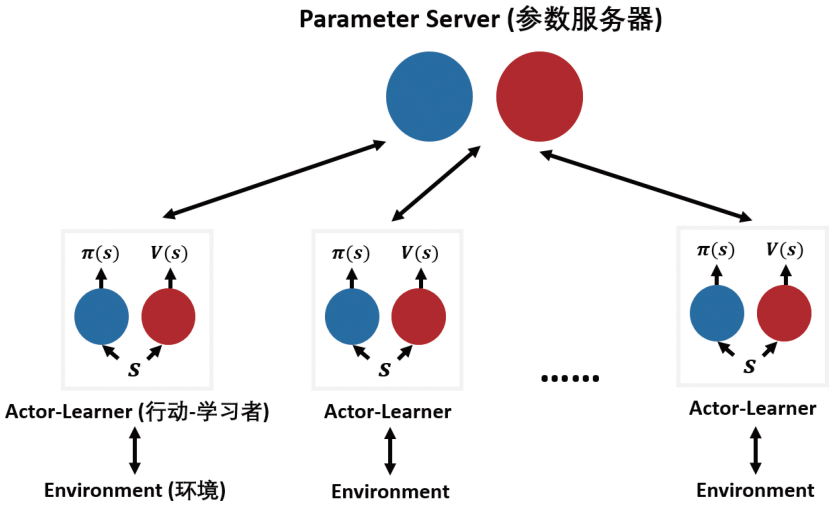

图 12.5 A3C 架构

算法 12.37 异步优势 Actor-Critic （Actor-Learner）

超参数: 总探索步数 $T_{\max}$，每个周期内最多探索步数 $t_{\max}$。
初始化步数 $t=1$
while $T \leqslant T_{\max}$ **do**
  初始化网络参数梯度: $\mathrm{d}\theta=0$ 和 $\mathrm{d}\theta_v=0$
  和参数服务器保持同步并获得网络参数 $\theta'=\theta$ 和 $\theta'_v=\theta_v$
  $t_{\text{start}}=t$
  设定每个探索周期初始状态 S_t
  while 达到终结状态 or $t-t_{\text{start}}==t_{\max}$ **do**
    基于决策策略 $\pi(S_t|\theta')$ 选择决策行为 a_t
    在环境中采取决策行为，获得奖励 R_t 和下一个状态 S_{t+1}。
    $t=t+1, T=T+1$。
  end while
  if 达到终结状态 **then**
    $R=0$
  else
    $R=V(S_t|\theta'_v)$
  end if
  for $i=t-1, t-2, \cdots, t_{\text{start}}$ **do**
    更新折扣化奖励 $R=R_i+\gamma R$。
    积累参数梯度 θ', $\mathrm{d}\theta=\mathrm{d}\theta+\nabla_{\theta'}\log\pi(S_i|\theta')(R-V(S_i|\theta'_v))$。
    积累参数梯度 θ'_v, $\mathrm{d}\theta_v=\mathrm{d}\theta_v+\partial(R-V(S_i|\theta'_v))^2/\partial\theta'_v$ 。
  end for
  基于梯度 $\mathrm{d}\theta$ 和 $\mathrm{d}\theta_v$ 异步更新 θ 和 θ_v
end while

和环境交互 $T_{\max}$ 次后，行动-学习者会将两个网络所有累积的梯度之和提交给参数服务器，使其能够分别异步更新网络服务器中的网络参数 θ 和 θ_v。

12.4.2　GPU/CPU 混合式异步优势 Actor-Critic

为了更好地利用 GPU 的计算资源从而提高整体计算效率，A3C 进一步优化提升为 GPU/CPU 混合式异步优势 Actor-Critic（Hybrid GPU/CPU A3C，GA3C）(Babaeizadeh et al., 2017)，如图 12.6 所示，在学习模型与环境的交互过程中，GA3C 算法主要由智能体、预测者（Predictor）和训练者（Trainer）三部分组成，每一部分的功能具体如下。

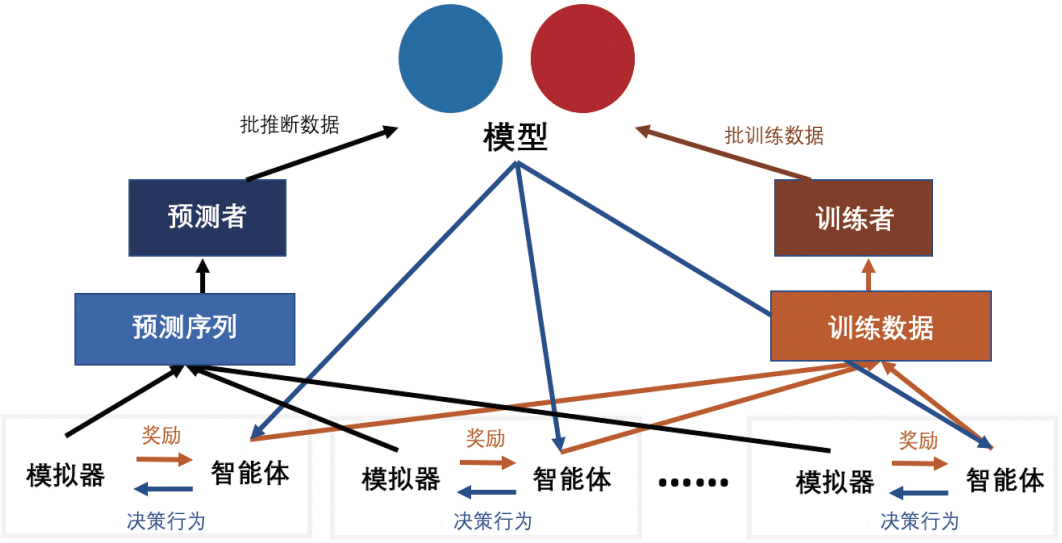

图 12.6　GA3C 架构

- **智能体**: 在 GA3C 算法中，多个智能体分别与模拟的环境进行交互，然而每个智能体自身不需要维护一个策略网络来指导其做出决策动作，而是基于当前的状态 S_t，将如何决策的请求发送给预测序列，预测者则会根据整体策略网络顺序为预测序列中的请求提供决策建议。当智能体采取 A_t 决策动作，通过与环境的交互获得奖励 R_t 并进入状态 S_{t+1} 时，智能体会将其探索经验的信息 (S_t, A_t, R_t, S_{t+1}) 发送到训练队列，用于学习网络参数的训练提升。
- **预测者**: 当预测者从智能体中收集决策请求并存储在预测队列中，再进行模型推论时，将从预测队列中批量获取决策请求，并将其输入决策网络中，从而为每一个请求得到建议的决策动作。由于批量的数据输入使得模型在推论时可以利用 GPU 的并行计算能力，因而提升了学习模型的计算效率。基于不同的请求数量，预测者和其预测队列的数目可以随之调整用于控制信息的处理速度，降低计算延迟，从而进一步提升计算效率。

- **训练者**: 在收到多个智能体的交互经验信息后，训练者会将信息存储在训练队列中，并从中批量选取数据，用于整体策略网络和价值网络的模型训练。同样，在模型训练中，批量的信息输入利用 GPU 的并行计算能力提升了计算效率，同时也降低了训练结果的方差。

12.4.3 分布式近端策略优化

分布式近端策略优化（Distributed Proximal Policy Optimization，DPPO）是 PPO 算法的分布式版本，如图 12.7 所示，其中领导者和工人分别与 A3C 算法中的参数服务器和行动-学习者的功能相对应。DPPO 算法将数据的采集和梯度计算分布在多个工人中执行，从而大大降低了学习的时间。周期性地接收每一个工人提交的平均梯度值，领导者会更新其自身的网络参数并将最新的网络参数同步更新给所有工人。

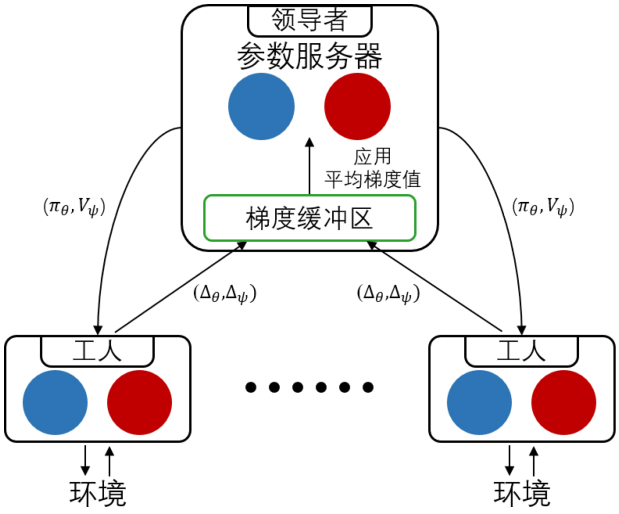

图 12.7 DPPO 架构

DPPO 算法的伪代码从领导者（Chief）和工人（Worker）的角度分别由算法 12.38、算法 12.39 和算法 12.40 所述，由于工人可以基于 PPO 算法两种版本 PPO-Penalty 和 PPO-Clip 中的一个，因而本节相对应地提出两种 DPPO 算法，分别为 DPPO-Penalty 和 DPPO-Clip。这两种算法在领导者的部分是相同的，唯一的区别是工人计算梯度的方法，具体可见如下伪代码。

算法 12.38 DPPO (Chief)

超参数: workers 数目 W，可获得梯度的 worker 数目门限值 D, 次迭代数目 M, B。

输入: 初始全局策略网络参数 θ，初始全局价值网络参数 ϕ。

for k $= 0, 1, 2, \cdots$ **do**

  for $m \in \{1, \cdots, M\}$ **do**

等待至少可获得 $W-D$ 个 worker 计算出来梯度 θ，去梯度的均值并更新全局梯度 θ。

end for

for $b \in \{1, \cdots, B\}$ **do**

等待至少可获得 $W-D$ 个 worker 计算出来梯度 ϕ，去梯度的均值并更新全局梯度 ϕ。

end for

end for

领导者从工人中收集网络参数的梯度信息并用于更新其自身的网络参数。由算法 12.38 所示，在每一次迭代中，策略网络和价值网络分别将执行 M 和 B 次子迭代。在每一次子迭代中，领导者至少等待所有工人提交 $(W-D)$ 组梯度数据，然后用所有这些梯度的均值来更新网络参数。更新的网络参数将会和所有工人同步，用于其之后的采样和梯度计算。

从自身的角度会收集数据样本并计算梯度，然后将梯度传递给领导者。算法 12.39 和算法 12.40 除在计算策略梯度的部分外大致相同，在每次迭代中，工人首先会收集一组数据 $\mathcal{D}_k$，并根据收集的数据计算算法中的 $\hat{G}_t$ 或 $\hat{A}_t$，将当前探索的策略 π_θ 存储为 π_{old}，在策略网络和价值网络中分别重复 M 和 B 次子迭代过程。

算法 12.39 DPPO (PPO-Penalty worker)

超参数: KL 惩罚系数 λ, 自适应参数 $a=1.5, b=2$, 次迭代数目 M, B。

输入: 初始局部策略网络参数 θ, 初始局部价值网络参数 ϕ。

for k $= 0, 1, 2, \cdots$ **do**

通过在环境中采用策略 π_θ 收集探索轨迹 $\mathcal{D}_k = \{\tau_i\}$

计算 rewards-to-go $\hat{G}_t$

基于当前价值函数 V_{ϕ_k} 计算对 advantage 的估计，$\hat{A}_t$（可选择使用任何一种 advantage 估计方法）。

存储部分轨迹信息

$\pi_{\text{old}} \leftarrow \pi_\theta$

for $m \in \{1, \cdots, M\}$ **do**

$$J_{\text{PPO}}(\theta) = \sum_{t=1}^{\text{T}} \frac{\pi_\theta(A_t|S_t)}{\pi_{\text{old}}(A_t|S_t)} \hat{A}_t - \lambda \text{KL}[\pi_{\text{old}}|\pi_\theta] - \xi \max(0, \text{KL}[\pi_{\text{old}}|\pi_\theta] - 2\text{KL}_{\text{target}})^2$$

if $\text{KL}[\pi_{\text{old}}|\pi_\theta] > 4\text{KL}_{\text{target}}$ **then**

break 并继续开始 $k+1$ 次迭代

end if

计算 $\nabla_\theta J_{\text{PPO}}$

发送梯度数据 θ 到 chief

等待梯度被接受或被舍弃，更新网络参数。

end for

for $b \in \{1, \cdots, B\}$ **do**

$L(\phi) = -\sum_{t=1}^{\mathrm{T}} (\hat{G}_t - V_\phi(S_t))^2$

计算 $\nabla_\phi L$

发送梯度数据 ϕ 到 chief

等待梯度被接受或被舍弃，更新网络参数。

end for

计算 $d = \hat{\mathbb{E}}_t \left[\mathrm{KL}[\pi_{\text{old}}(\cdot|S_t), \pi_\theta(\cdot|S_t)] \right]$

if $d < d_{\text{target}}/a$ **then**

$\lambda \leftarrow \lambda / b$

else if $d > d_{\text{target}} \times a$ **then**

$\lambda \leftarrow \lambda \times b$

end if

end for

算法 12.40 DPPO (PPO-Clip worker)

超参数: clip 因子 ϵ, 次迭代数目 M, B。

输入: 初始局部策略网络参数 θ, 初始局部价值网络参数 ϕ。

for k $= 0, 1, 2, \cdots$ **do**

通过在环境中采用策略 π_θ 收集探索轨迹 $\mathcal{D}_k = \{\tau_i\}$

计算 rewards-to-go $\hat{G}_t$

基于当前价值函数 V_{ϕ_k} 计算对 advantage 的估计，$\hat{A}_t$（可选择使用任何一种 advantage 估计方法）。

存储部分轨迹信息

$\pi_{\text{old}} \leftarrow \pi_\theta$

for $m \in \{1, \cdots, M\}$ **do**

通过最大化 PPO-Clip 目标更新策略：

$$J_{\text{PPO}}(\theta) = \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in D_k} \sum_{t=0}^{\mathrm{T}} \min \left(\frac{\pi_\theta(A_t|S_t)}{\pi_{\text{old}}(A_t|S_t)} \hat{A}_t, \left(\frac{\pi(A_t|S_t)}{\pi_{\text{old}}(A_t|S_t)}, 1-\epsilon, 1+\epsilon\right) \hat{A}_t \right)$$

计算 $\nabla_\theta J_{\text{PPO}}$

发送梯度数据 θ 到 chief

等待梯度被接受或被舍弃，更新网络参数。

end for

for $b \in \{1, \cdots, B\}$ **do**

通过回归均方误差拟合价值方程：

$$L(\phi) = -\frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{\mathrm{T}} \left(V_\phi(S_t) - \hat{G}_t \right)^2$$

计算 $\nabla_\phi L$
发送梯度数据 ϕ 到 chief
等待梯度被接受或被舍弃，更新网络参数。
end for
end for

在 DPPO-Clip 中，网络参数 λ 会在所有工人中共享，但是否更新取决于每一个工人计算的平均 KL 散度。另外，在工人们共享的数据中建议使用统计值，例如，对观察到的数据，奖励和优势函数通过计算均值和标准差，使其具有归一化。另外，在 DPPO-Clip 算法中，当 KL 散度超过一定数值后会添加额外的处罚项。对于策略网络，在每次子迭代过程中会采用早停法来进一步提高算法稳定性。

12.4.4　重要性加权的行动者-学习者结构和可扩展高效深度强化学习

基于 A2C 学习算法，重要性加权的行动者-学习者结构（Importance Weighted Actor-Learner Architecture，IMPALA）在分布式计算中使用智能体探索轨迹的所有经验作为通信信息。如图 12.8 所示，IMPALA 架构由行动者和学习者组成，具体介绍如下：

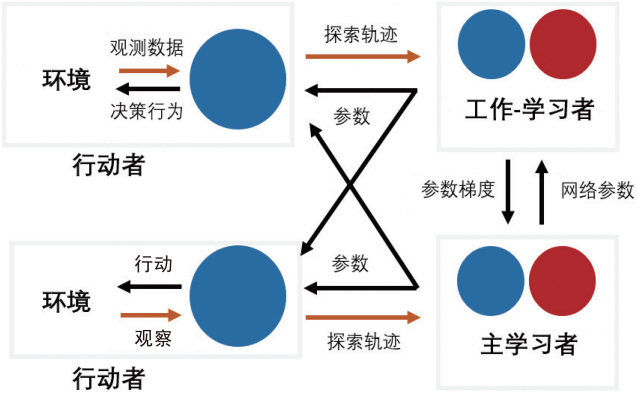

图 12.8　重要性加权的行动者-学习者结构

- **行动者**：每个行动者中会有一个复制的策略网络，用于在和模拟的环境交互时做出决策，在交互时收集到的经验将会存储到缓冲区中，在与环境交互固定次数后，每个行动者会将其

存储的探索轨迹经验发送给学习者，并和其他行动者以同步通信的方式从学习者中收到更新的策略网络参数信息。

- **学习者**：通过和行动者通信，学习者收到所有行动者收集的轨迹经验信息并用其训练模型，设在状态 S_T 下的价值估计为 n 步 V 轨迹 Target，定义如下

$$\text{Target} = V(S_T) + \sum_{t=T}^{T+n-1} \gamma^{t-T}(\Pi_{i=T}^{t-1} c_i)\delta_t V, \tag{12.1}$$

其中 $\delta_t V = \rho_t(R_t + \gamma V(S_{t+1}) - V(S_t))$ 表示时间差分。$\rho_t = \min(\bar{\rho}, \frac{\pi(S_t)}{\mu(S_t)})$。$c_i = \min(\bar{c}, \frac{\pi(S_i)}{\mu(S_i)})$。$\pi$ 为学习者的决策策略，为上一轮同步时所有行动者的策略 μ 的均值。

大规模模型训练算法可以存在多个学习者，细分为工人学习者和主学习者，每个学习者会和不同的行动者通信并独立完成模型训练，但周期性地，所有工人学习者需要和主学习者通信，每个工人学习者会将学习到的网络参数梯度发送给主学习者，然后主学习者会更新其自身的网络参数并同步更新到所有的工人学习中。

可扩展高效深度强化学习（Scalable，Efficient Deep-RL，SEED）架构 (Espeholt et al., 2019) 和 IMPALA 十分类似，主要的区别在于策略网络的推断过程会从行动者部分转移到学习者中，从而降低了行动者的算力要求和通信延时。具体的 SEED 架构如图 12.9 所示，由于每个行动者中只需要完成和环境的交互，很多弱算力的计算资源可以加入架构中并成为独立的行动者。根据学习者指导的决策动作，每一个行动者将一步的经验反馈给学习者，其反馈的经验信息将首先存储在学习者的经验缓冲器中。在多次迭代之后，批量的轨迹经验数据将提供给学习者进行模型训练，其中方程 (12.1) 中 V 轨迹目标也被使用作为状态的价值估计。

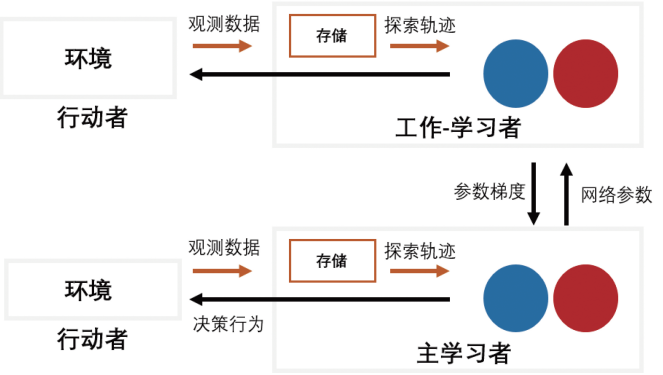

图 12.9 可扩展高效深度强化学习架构

12.4.5　Ape-X、回溯-行动者和分布式深度循环回放 Q 网络

在分布式网络中，考虑到智能体和环境的频繁交互，将带有优先级的经验回放加入架构中对规模化场景很有助益。Ape-X (Horgan et al., 2018) 是典型的包含带有优先级的经验回放部分的分布式架构，如图 12.10 所示。设有多个相互独立的行动者，在每个行动者中会有一个智能体在策略网络的指导下与环境进行交互。基于从多个行动者中收集的经验信息，学习者将训练其网络参数，从而学习最优的动作策略。不仅如此，除了行动者和学习者，算法中还有回放缓冲区收集所有行动者采集的信息，维护并更新每一个存储经验的优先级，并且根据优先程度从中批量选取数据发送给学习者进行模型训练。经过回放缓冲区的处理，批量且标有优先级的训练数据能够有效地提升计算效率及模型训练结果。

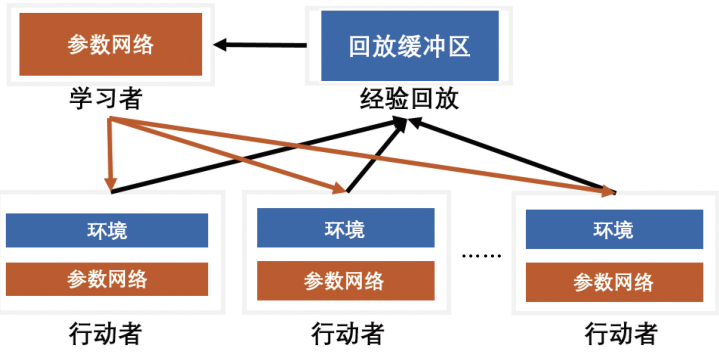

图 12.10　Ape-X 架构

在行动者中的算法如算法 12.41 所示，其中每一个行动者首先和学习者在策略网络参数上保持同步，更新的网络参数信息将指导智能体和环境发生交互。在收到环境带来的反馈后，行动者会计算其中探索经验数据的优先级，并将带有优先级信息的数据发送给回放缓冲区。

当回放缓冲区从行动者中收集到确定数目的经验信息后，学习者将开始从回放缓冲区获取批量信息进行学习。从学习者的角度如算法 12.42 所示，在每个模型训练周期中，学习者首先将从回放缓冲区中获得带有优先级的批量经验数据，每个数据信息用 $(i, \boldsymbol{d})$ 表示，其中 i 表示数据的索引编号，$\boldsymbol{d}$ 为具体的经验数据信息，其中包括初始状态、决策动作、奖励和采取动作后到达的状态四个部分。批量的经验数据将用来训练学习者的网络参数，并周期性地将更新的网络参数与所有行动者中的策略网络参数保持同步。另外，在模型训练之后，采样数据的优先级将会被调整并在回放缓冲区中更新。由于容量大小的限制，在回放缓冲区中会周期性地将具有较低优先级的数据删除。

当训练模型分别使用 DQN 或 DPG 算法的时候，基于如上架构，Ape-X 深度 Q 网络（Ape-X DQN）和 Ape-X 深度策略梯度（Ape-X DPG）相对应被提出。在 Ape-X DQN 中，Q 网络存在学习者和所有行动者中，在行动者中智能体的决策动作受网络产生的 Q 值指导；在 Ape-X DPG 中，

算法 12.41 Ape-X (Actor)

超参数: 单次批量发送到回放缓冲区的数据大小 B、迭代数目 T。
与学习者同步并获得最新的网络参数 θ_0
从环境中获得初始状态 S_0
for t $= 0, 1, 2, \cdots, T-1$ **do**
  基于决策策略 $\pi(S_t|\theta_t)$ 选择决策行为 A_t
  将经验 (S_t, A_t, R_t, S_{t+1}) 加入当地缓冲区
  if 当地缓冲区存储数据达到数目门限值 B **then**
    批量获得缓冲数据 B
    计算获得缓冲数据的优先级 p
    将批量缓冲数据和其更新的优先级发送回放缓冲区
  end if
  周期性同步并更新最新的网络参数 θ_t
end for

算法 12.42 Ape-X (Learner)

超参数: 学习周期数目 T
初始化网络参数 θ_0
for $t = 1, 2, 3, \cdots, T$ **do**
  从回放缓冲区中批量采样带有优先级的数据 (i, d)
  通过批数据进行模型训练
  更新网络参数 θ_t
  对于批数据 d 计算优先级 p
  更新回放缓冲区中索引 i 数据的优先级 p
  周期性地从回放缓冲区中删除低优先级的数据
end for

学习者将构建策略网络和价值网络，而在行动者里只有相同结构的策略网络，用于指导其智能体制定策略动作。

同样设立带有优先级的分布式回放缓冲区，回溯-行动者（Retrace-Actor, Reactor）(Gruslys et al., 2017) 基于 Actor-Critic 架构被提出，取代之前的单个经验信息，一个序列的经验信息将被同时输入缓冲区中，并采用 Retrace(λ) 算法来更新对 Q 值的估计。在神经网络中，LSTM 网络将在策略和价值网络中使用，并获得很好的模型训练结果。

类似地，分布式深度循环回放 Q 网络（Recurrent Replay Distributed DQN，R2D2）(Kapturowski et al., 2019) 在带有优先级的分布式回放缓冲区中采用具有固定长度序列的经验格式，基于深度 Q 网络（DQN）算法，R2D2 同样在策略网络中使用 LSTM 层，并且用存在回放缓存区中的状态数据训练网络。

12.4.6　Gorila

基于深度 Q 网络算法（General Reinforcement Learning Architecture，Gorila）(Nair et al., 2015) 如图 12.11 所示。在此架构中，当和参数服务器中深度 Q 网络的参数保持同步之后，行动者将在深度 Q 网络的指导下和环境进行交互，并将通过交互收集到的经验直接发送到回放缓冲区。回放缓冲区将存储并管理所有从行动者中收集经验信息。当从回放缓冲区中获取批量数据后，学习者将会进行模型学习并计算 Q 网络中参数的梯度。在学习者中会用一个学习 Q 网络和一个目标 Q 网络来计算 TD 误差，其中学习 Q 网络将会在学习的每一步和参数服务器中的网络参数保持同步，然而目标 Q 网络只在每过 N 步之后和参数服务器同步。参数服务器将周期性地从学习者中接收网络参数的梯度信息，并更新自身的网络参数，以使之后的探索更具效率。

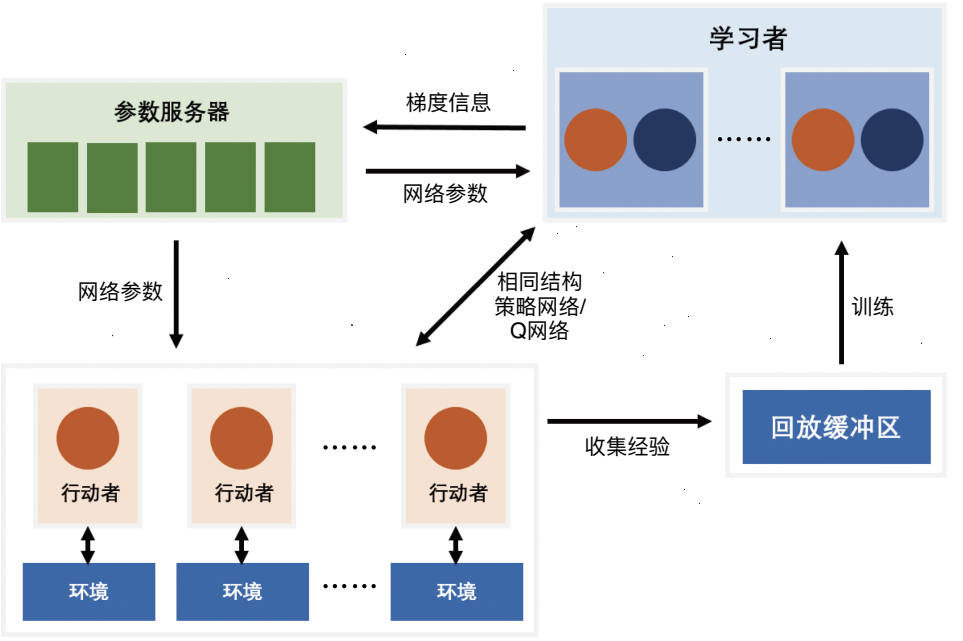

图 12.11　Gorila 架构

12.5　分布式计算架构

基于并行计算的基本模式和结构，在分布式强化学习中，大规模并行计算架构能够得以进一步探索和研究。一般来说，其系统一般会有如下基本组成元素：

- **环境**（Environments）：环境是智能体需要与其交互的场景。在深度学习的大规模并行计算

中，环境可能会存在多个复制版本，并分别对应到多个行动者中，使其能够相互独立地并行探索，获取经验，并且，在基于模型的强化学习中，通常会用多个模拟的环境来使其在探索和学习中具备并行性。

- **行动者**（Actors）：系统中行动者通常指直接和环境进行交互的部分。其中，可能会有多个行动者和一个或多个真实或模拟的环境进行交互，并且其中每一个行动者都能在所给出的环境状态下独立地做出决策动作。其决策动作可以由行动者自身的策略网络或者 Q 网络推断产生，或者是由参数服务器或者其周围的学习者中共享的策略网络或 Q 网络产生的。当行动者在环境中进行了连续多步的探索之后，探索轨迹将形成。形成的探索轨迹将会被提交到回放存储缓冲器或者直接提交给学习者。由于行动者和环境的交互需要花费很多时间，所以，行动者探索和数据收集的并行性能够提升获得经验数据的速度，从而可以将批量数据发送给学习者训练模型，训练效果得到很大提升。
- **回放存储缓冲区**（Replay Memory Buffers）：回放存储缓冲区将会从所有行动者中收集探索轨迹，并将其整理后提供给学习者用于策略学习或者 Q 学习。由于存储缓冲区需要快速的数据读写和数据打乱重排，数据存储的结构同样需要支持动态且并行的方式。并且，由于大多数学习者依赖回放存储缓冲区中的数据进行模型训练。为了保证模型训练的高效率，建议在回放存储缓冲区分配在学习者的周围并高效连通。
- **学习者**（Learners）：学习者是深度强化学习的关键组成部分，基于不同的深度强化学习方法，学习者的结构也将各有不同。通常来说，每个学习者会维护一个策略网络或 Q 网络，并且用从回放存储缓冲区中得到的行动者与环境交互的经验信息来训练深度网络参数。在训练前后，学习者均会和参数服务器进行通信，使其在训练前同步深度网络参数信息，并在训练后提交训练得到的参数梯度信息。多个学习者和参数服务器的通信方式可以是同步或异步的。
- **参数服务器**（Parameter Servers）：参数服务器是从学习者中收集所有信息并维护管理策略网络或者 Q 网络中的参数信息。参数服务器将周期性地和所有学习者保持同步，使每个学习者获得其他学习者学习得到的信息，并且参数服务器能够在行动者和环境交互时帮助其制定决策策略。在大规模深度强化学习系统中，为了保证参数服务器和学习者及行动者通信时的稳定性和高效率，参数服务器自身也可以具有多种不同的架构，其内部也可采用集中式或分布式的数据通信方式。

一般性的分布式计算架构可以采纳其中元素组合形成。由于其中的每一部分可独立且并行进行数据的存储，传输或计算，其架构可以根据要解决的问题适应性调整并灵活改变，从而充分满足应用中多样的学习任务需求。

参考文献

BABAEIZADEH M, FROSIO I, TYREE S, et al., 2017. Reinforcement learning thorugh asynchronous advantage actor-critic on a gpu[C]//ICLR.

ESPEHOLT L, MARINIER R, STANCZYK P, et al., 2019. Seed rl: Scalable and efficient deep-rl with accelerated central inference[J]. arXiv preprint arXiv:1910.06591.

GRUSLYS A, DABNEY W, AZAR M G, et al., 2017. The reactor: A fast and sample-efficient actor-critic agent for reinforcement learning[Z].

HORGAN D, QUAN J, BUDDEN D, et al., 2018. Distributed prioritized experience replay[Z].

KAPTUROWSKI S, OSTROVSKI G, DABNEY W, et al., 2019. Recurrent experience replay in distributed reinforcement learning[C]//International Conference on Learning Representations.

MNIH V, BADIA A P, MIRZA M, et al., 2016. Asynchronous methods for deep reinforcement learning[C]//International Conference on Machine Learning (ICML). 1928-1937.

NAIR A, SRINIVASAN P, BLACKWELL S, et al., 2015. Massively parallel methods for deep reinforcement learning[Z].

OPENAI, :, BERNER C, et al., 2019. Dota 2 with large scale deep reinforcement learning[Z].