面向执行-学习者的在线强化学习并行训练方法

孙正伦 乔 鹏 窦 勇 李青青 李荣春

(国防科技大学并行与分布处理国家重点实验室 长沙 410073)

摘 要 近年来,深度强化学习(Deep Reinforcement Learning,DRL)已经成为了人工智能领域中的研究热点.为了加速DRL训练,人们提出了分布式强化学习方法用于提升训练速度.目前分布式强化学习可以分为同策略方法、异策略方法以及最新的近同策略方法.近同策略方法改善了同策略方法和异策略方法的问题,但是由于其共享内存并行模型的限制,近同策略模型难以扩展到以网络互连的计算集群上,低可扩展性限制了近同策略方法能够利用的资源数量,增加了计算节点的负载,最终导致训练耗时增加.为了提升近同策略方法的可扩展性,提升收敛速度,本文提出了一种以消息传递为基础,使用Gossip算法与模型融合方法的并行执行者-学习者训练框架(Parallel Actor-Learner Architecture,PALA),这一方法通过增强训练的并行性和可扩展性来提升收敛速度.首先,该框架以Gossip算法作为通信基础,借助全局数据代理并使用消息传递模型创建了一套可扩展的多个并行单智能体训练方法.其次,为了保证探索-利用的同策略性,维持训练稳定,本文创建了一套可以用于多机之间进行隐式同步的进程锁.其次,本文面向含有CUDA张量的模型数据,提出了一种序列化方法,以保证模型数据能够通过节点间网络传递、聚合.最后,本文使用模型聚合方法对训练进行加速.基于上述优化和改进,PALA训练方法能够将负载均衡地映射到整个计算集群上,减少由于高负载而造成的长等待时间,提升收敛速度.实验表明,相较于之前使用共享内存模式的方法,PALA训练的智能体在达到相同水平时,训练时间缩减了20%以上,同时,PALA还有着较好的可扩展性,PALA可以扩展的硬件资源数量是原有方法的6倍以上.与其他方法相对比,PALA训练的智能体最终策略在几乎所有测试环境中达到了最优水平.

关键词 Gossip算法;强化学习;同策略学习;分布式强化学习;并行训练方法

中图法分类号 TP18 **DOI号** 10.11897/SP.J.1016.2023.00229

PALA: Parallel Actor-Learner Architecture for Distributed Deep Reinforcement Learning

SUN Zheng-Lun QIAO Peng DOU Yong LI Qing-Qing LI Rong-Chun

(*Department of Science and Technology on Parallel and Distributed Processing Laboratory, National University of Defense Technology, Changsha* 410073)

Abstract In recent years, Deep Reinforcement Learning has become a hot spot in the field of Artificial Intelligence, it has reached great success in many complex environments and even outperforms humans in several complex games. To accelerate DRL training, researchers proposed distributed reinforcement learning to improve training speed and scalability. At present, there are three types of distributed reinforcement learning: on-policy, off-policy, and near on-policy. Near

收稿日期:2021-11-26;在线发布日期:2022-09-30.本课题得到国家自然科学基金(61732018、61902415、61972409)和重点实验室开放基金(WDZC20205500104)资助.**孙正伦**,博士研究生,中国计算机学会(CCF)会员,主要研究方向为智能计算(强化学习、深度学习等)、高性能计算(并行计算等).E-mail:zhenglun_sun@nudt.edu.cn.**乔 鹏**(通信作者),博士,助理研究员,主要研究方向为高性能计算(并行计算、异构计算等)、智能计算(强化学习、分布式机器学习等)、人工智能应用(计算机视觉).E-mail:pengqiao@nudt.edu.cn.**窦 勇**,博士,研究员,国家杰出青年科学基金人选者,中国计算机学会(CCF)会士,主要研究领域为高性能计算、智能计算(机器学习、深度学习).**李青青**,硕士研究生,主要研究方向为人工智能应用(自然语言处理等)、智能计算(强化学习等).**李荣春**,博士,副研究员,中国计算机学会(CCF)会员,主要研究方向为高性能计算、智能计算(深度学习)、人工智能应用(人脸识别、视频理解等).

on-policy solve the problem of on-policy and off-policy and improve training efficiency, but the near on-policy method is based on shared memory parallel model. Due to the limitation of the shared memory parallel model, near on-policy have trouble expanding to the cluster connected with the network. This problem made the near on-policy method hard to scale and limited the resource near on-policy can use, which ultimately lead to the increment of the workload in computation peer and the increment of the training time. In order to improve the scalability and speed up the convergence of near on-policy, we propose a parallel actor-learner architecture (Parallel Actor-Learner Architecture, PALA). The parallel model of this architecture is based on message passing parallel model, using the Gossip Algorithm and model average method. First, with the help of data proxy and message passing parallel model which performed in the Gossip algorithm way, we proposed a salable parallel training architecture of single agent training. This architecture could expand to multiple nodes connected with the network. Secondly, in order to stabilize training and keep the policy difference between learner and actor within a small bound, we proposed a process lock. This lock could be used in the cluster among multiple nodes for implicit synchronization. Without implicit synchronization, parallel training will not take much effect. Thirdly, this paper proposes a serialization method for model data containing CUDA tensors to ensure that model data can be transmitted and aggregated through the network between nodes. Finally, this paper uses model aggregation method to speed up training. Based on the above optimizations and improvements, the PALA training method can map the load to the entire computing cluster in a balanced manner, reduce the long waiting time caused by high load, and improve the convergence speed. By improving scalability, a large number of actors and learners are deployed by PALA which can fully explore the environment and jump out from local optimal. The experiment shows that compared with the former method which uses shared memory, PALA can accelerate training by more than 20% and it can also reach for more than 6 times computing resources. In some environments, PALA even outperforms the former method with up to 50% reward valve with the same training time. The final policies trained by PALA can outperform other methods in the vast majority of games. Experimental results show that PALA trained for 2.5 million steps already outperforms other methods after training for 4 million steps. When training for 2.5 million steps, PALA outperforms former methods by up to 90%. We also evaluated the reason why PALA reach higher performance compared with other method.

Keywords Gossip algorithm; reinforcement learning; on-policy; distributed reinforcement learning; parallel training

1 引 言

深度强化学习是一种融合了深度学习(Deep Learing,DL)与强化学习(Reinforcement Learning,RL)的人工智能算法.近年来,深度强化学习取得了重大进展[1].在即时战略游戏、围棋等复杂领域,深度强化学习取得了丰硕成果.2019 年,由 DeepMind 提出的 AlphaStar[2]在实时战略游戏《星际争霸 2》中超越了 99.8%参与过排位赛的人类选手.针对围棋提出的智能体 AlphaGo[3-4]同样战胜了人类职业选手.

但是,在深度强化学习中,训练这些智能体对计算资源的需求量极高并且耗时很长,本质的原因是智能体对于样本的利用率较低[5]导致训练样本的需求量很大.以在《星际争霸 2》中训练 AlphaStar 为例,为了训练 AlphaStar,DeepMind 使用的计算资源超过了 4000 个 CPU 和 128 个 TPU,整个训练过程超过了十个月.目前,为了提升训练效率,常用的方法之一是使用并行训练方法,即通过并行大量的执行者、学习者,加速采样和训练,改善训练所需要的时间或者步数.

深度强化学习主要由智能体组成,智能体分为

两个功能模块，分别是执行者(Actor)和学习者(learner).在训练中，执行者和学习者交替执行，执行者使用执行策略搜集样本，学习者则利用样本更新自身的目标策略，策略通常使用一个神经网络表示.同策略(On-policy)算法的执行策略和目标策略相同，异策略(Off-policy)算法的执行策略和目标策略不同.

为了提升强化学习训练的训练速度，Mnih 等人提出了被称作异步优势执行-评价者算法[6](Asynchronous Advanced Actor-ciritc Algorithm，A3C)的同策略算法提升了训练的并行度.A3C 使用多个执行者探索环境，异步地计算参数并更新网络，提升了样本采集的效率，极大地缩减了训练时间.Horgan 等人提出了 Ape-X[7]算法，相较于同策略算法，异策略算法 Ape-X 使用了带有优先级的重放缓存，将多个并行的执行者搜集到的样本放入重放缓存中，学习者从重放缓存中依据优先级取用样本进行训练.由于使用了缓存，这一方法具有比较好的可扩展性.不同于之前的同策略和异策略算法，Assran 等人提出了基于 Gossip 算法的执行-学习者训练框架[8](Gossip-based Actor-learner Architectures for Deep Reinforcement Learning，GALA).GALA 方法以 Gossip 算法作为节点间的通信模式，使用模型聚合(Model Average)[9]这一分布式梯度下降方法加速收敛.这一方法属于近同策略算法，通过放松同策略算法的一致性限制，减少了数据依赖，得到了比较好的可扩展性.GALA 算法在 Atari 2600 游戏环境中成功加速了训练并且得到了更高的分数.

上述的并行深度强化学习训练方法分别关注三个方面，第一是使用重放缓存作为并行执行中的缓存来提升并行性；第二是异步并行多个单智能体来提升并行性；第三是放松异步并行中策略的限制来提升并行性.Ape-X 属于第一类，它通过使用带有优先级的重放缓存作为各个并行执行者的缓存，学习者在重放缓存中采样，减少了并行所需要的同步.在 Ape-X 中通常含有一个学习者和多个执行者.A3C 则属于第二类，它异步地并行多个使用相同策略的执行者进行采样，不需要使用缓存.在 A3C 中通常含有一个学习者和多个执行者.GALA 则属于第三类，它通过异步地并行多个使用不同策略的单智能体，以共享内存方式连接这些智能体，这些智能体使用模型聚合方式求整体梯度，加速训练.它在第二类的基础上放松了策略一致性的限制，即不要求执行策略与目标策略完全相同，减少了策略一致性的限制.GALA 中通常含有多个学习者和多个执行者.

虽然近同策略方法可以避免同策略学习方法对于执行策略和目标策略的一致性要求，也可以避免异策略学习方法中的训练不稳定[10]和锁冲突问题.但是由于其共享内存并行模型的限制，导致其难以扩展到以网络互连的整个集群上，如实验中分析.因此，针对这一问题，本文从训练的并行模型入手，提出了一种基于消息传递模型的并行单智能体在线策略分布式训练方法.原有并行框架使用的共享内存模型难以扩展到以网络进行互联的集群上，这就导致智能体在有限的计算资源下需要进行等待来获取资源.本文采用了一种基于消息传递模型的并行训练方法，这种并行训练方法能够将每个智能体所需的信息以消息传递的模式发送到各个以网络互联的计算节点上.另一方面，以智能体组成的训练框架使用模型聚合来加速收敛，即智能体在每次迭代中接受其他智能体发送的模型参数并取平均，框架内的智能体之间不涉及合作与竞争，这要求训练框架能够在网络间传输含有 CUDA 张量的模型，并使用可跨节点的锁并保证聚合过程维持次序.本文使用循环等待方法作为跨节点锁的实现方式，并附加参数序列化方法保证跨节点参数传递正常进行.实验表明相较于共享内存中的锁，循环等待锁的效率并没有对总体性能造成太大影响.

在实现中，PALA 使用优势演员-评论员算法(Advantage Actor-critic Algorithm，A2C)作为框架中的智能体，称为 PALA-A2C，这些智能体被分散地部署在集群的各个节点上.我们将 PALA-A2C 与 GALA-A2C、A2C、IMPALA[11]在六个常用的 Atari 游戏[12]上进行比较.

本文的贡献主要包括三个方面：

(1)相较于旧有方法，PALA 的训练效率得到了显著提升.通过提升可扩展性，PALA 部署的大量智能体可以对环境进行充分地探索，使得相同训练步数下 PALA 的训练耗时更短，得分更高.与 GALA-A2C 相比，PALA 在训练到相同水平时可以节约超过 20%的训练时间.

(2)相较于 GALA-A2C 在单个计算节点内部进行训练，使用消息传递模型的 PALA-A2C 有效地提升了其在集群内部的可扩展性.实验表明 PALA 可以利用的计算资源 6 倍于 GALA.

(3)PALA 最终训练出的策略可以在绝大多数游戏上超越其他方法.实验结果表明，训练 250 万步的 PALA 已经优于其他方法训练 400 万步后的结果.

2 相关工作

最初的深度强化学习算法如深度 Q 网络(Deep Q-Network,DQN)和执行-评价者算法(Actor-Critic algorithm,AC)[13]基本都是单线程的.显然,当训练环境复杂度提高之后,智能体从环境中获得样本的时间更长,单线程的训练方法无法满足训练时间的需求.

2016 年,Mnih 等人则提出了 A3C 算法,A3C 是在 A2C 的基础上开发的异步版本.A3C 使用多个执行者同时进行采样,为 AC 算法提供了一种并行训练的方法,这一方法将训练时间缩短到了 DQN 的八分之一.

2018 年,IMPALA 针对强化学习训练出来的智能体通用性不强的问题,提出多任务训练的概念,即使用同一套神经网络控制不同的任务.IMPALA 属于异策略学习算法,使用 V-trace 对样本进行权值更新.其中执行者和学习者异步执行.

2018 年,Horgan 等人提出了 Ape-X 算法.Ape-X 为 DQN 算法提供了并行训练策略,它通过并发多个执行者进行采样,使用全局重放缓存作为样本的存储空间,通过提升并行度和可扩展性提升了计算效率.

2019 年谷歌提出 SEED RL[14].SEED RL 旨在提升推理速度,降低参数传递所需的带宽并充分利用计算资源,SEED RL 使用专用硬件对神经网络进行推理,并确保参数保持在本地以便节约带宽.同时使用 V-trace 等方法对样本进行修正和动作重放.

鉴于 A3C 作为同策略异步架构在并行方面的优秀表现,很多学者在 A3C 的基础上进行了许多改进,如 Assran 等人在并行多个执行者的同时并行多个学习者,使用模型平均这种分布式求梯度方法,创造了 GALA 算法.GALA 算法同时并行多个智能体,多个智能体之间使用全连接,以 Gossip 算法进行交互,在提升采样能力的同时提升了总体的学习能力.在 GALA 的实验中,它在一个含有 48 个 CPU 核与 1 个 GPU 单机进行部署和训练.GALA 算法使用的共享内存模型限制了其可扩展性,在节点间以网络进行互联的集群中,GALA 算法较难进行有效扩展.

针对执行-评价者这一强化学习框架,目前也存在一些强化学习方法针对其中参数敏感的问题进行优化.Self-Tuning Actor-Critic[15](STAC)方法通过对可微分的超参数进行自动调优,在相关任务中取得了 2～3 倍的性能提升.

3 背 景

3.1 强化学习

强化学习的目的在于让智能体从一个复杂环境中获得最高的奖励值[16].深度强化学习中的智能体会在状态 s 使用动作 a,然后迁移到下一个状态并且得到奖励值 r,通过这个过程,智能体在环境中不断探索和学习.目前强化学习大致可以分为两条发展路径,分别是基于价值的求解方法和基于策略的求解方法.基于价值的求解方法会显式学习一个用于刻画环境状态好坏的价值函数 $V(s;\theta)$,用于指导智能体的行为.基于策略的求解方法则要显式地学习一个策略 $\pi(a_t|s_t,\omega)$,策略在得知当前观测的环境状态后会返回使用不同动作的概率.

智能体在环境中采取动作并搜集状态转移轨迹作为训练样本.在离散的时间步 t 时,智能体处于状态 s_t.环境在接受到智能体的动作时会给予智能体相应的奖励值 r_t 作为反馈,为了最大化奖励回报,智能体会根据策略 $\pi(a_t|s_t,\omega)$ 或者价值函数 $V(s;\theta)$ 选择合适的动作 a_t,其中 ω 代表策略网络的参数.在智能体动作执行完成之后就进入到下一个状态 s_{t+1} 并且根据环境得到奖励 r_t.(s_t,a,s_{t+1},r_t) 组成了一个用于训练的轨迹样本,神经网络使用轨迹样本进行训练来更新策略.

在搜集完成样本后,智能体需要使用样本训练网络.为了获得最大的得分,智能体会利用策略梯度方法[17](Policy Gradient,PG)进行梯度上升来优化网络参数 ω、PG 方法以奖励值作为权重,动作的奖励值越高,这个样本的权重越大.依据这个方法,PG 方法会促使梯度向得分更高的方向前进,进而促使智能体得到更高的得分.

3.2 并行强化学习的训练架构

智能体与模拟器交互并采样所用的时间是训练速度提升的瓶颈之一.并行强化学习的主要思路是通过并行多个执行者增加采样效率,从而提升收敛性.近年来,使用基于策略的强化学习方法得到了长足的发展,为其提供扩展性高的并行训练方法成为重要需求之一.基于策略的强化学习方法会学习一个策略,将智能体选择动作转化为一个分类问题:策略网络的输入是环境,输出是要选择的动作.策略网络使用智能体搜集到的样本进行训练,使用 PG 算

法求梯度.

AC 算法是一种结合了价值函数与策略梯度的强化学习算法，这一算法内部包括了两个网络，分别是代表当前奖励期望值的 Q 网络和作为基线的价值函数 V. 如果当前的奖励期望值高于基线，那么在求策略梯度时就给予其正向权重，反之则给予负向权重. 通过这种方法，策略梯度不断促使策略网络参数向得分更高的方向移动. AC 算法的架构图如图 1 所示.

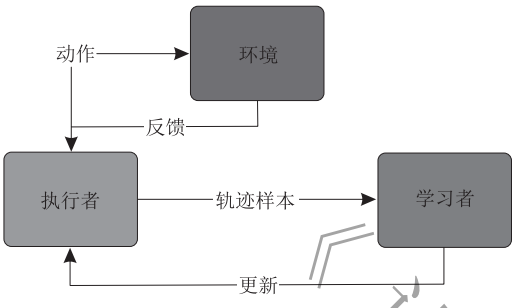


图 1　AC 算法架构图

A2C 算法和 A3C 算法则是在 AC 算法基础上衍生而出的同策略并行训练方法. 为了提升样本采集效率，这两种算法都同时并行了多个执行者用于搜集样本，执行者在搜集完样本后会在本地网络计算梯度，随后使用自己的梯度更新全局网络. A2C 和 A3C 的区别在于，对于全局网络的更新，A2C 是同步的，A3C 则是异步的. A2C 会等待所有执行者都算出梯度后进行汇总，求得平均之后再传给全局网络，A3C 则是当某一个执行者算出梯度后就对全局网络进行更新. 并行 AC 算法的架构图如图 2 所示.

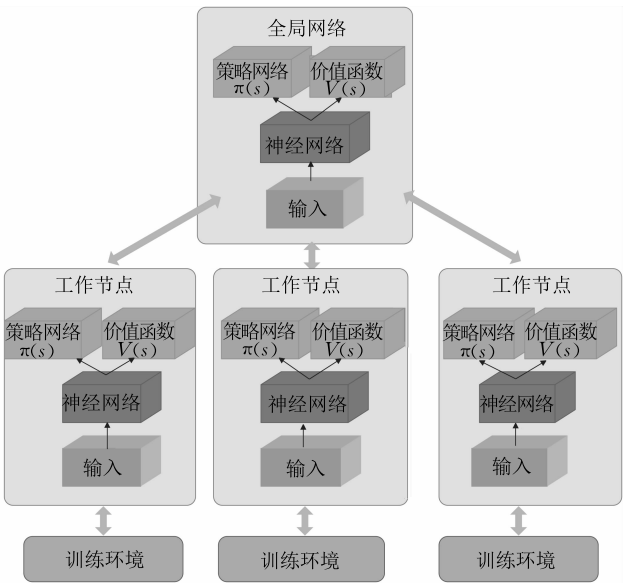


图 2　并行 AC 算法架构图

近同策略算法 GALA 与 A2C 和 A3C 不同，它通过同时并行多个 A2C 智能体来增强并行度. 这些智能体内部含有一个学习者和多个执行者，学习者之间通过模型聚合方法做全局梯度，加速训练.

3.3　并行强化学习方法的训练流程

并行强化学习目前有两种训练流程分别是：分散执行加中心训练、分散执行加分散训练. 以 A2C 这个分散执行加中心训练算法为例，多个执行者并行执行动作，等待执行者与环境交互完成后，每个执行者将自己搜集到的轨迹 (s_t, a, r, s_{t+1}) 交给学习者进行训练. 学习者使用 $\nabla_\omega(-\log\pi(a_t|s_t;\omega)A_t)$ 来更新参数 θ. 其中 $A_t=G_t^N-V(s_t;\theta_t)$，$G_t^N$ 是从当前状态执行 N 步的奖励值，$V(s_t;\theta_t)$ 是价值函数.

GALA 算法则是属于分散执行加分散训练. GALA 框架中含有多个 A2C 智能体，每个 A2C 智能体下的执行者并行采集样本，每个智能体内的学习者异步通信，使用模型平均进行全局梯度计算. 模型平均会对当前通信的节点模型参数求平均. 这种方法可以显著加快梯度下降. GALA 在增加了框架整体采样能力的同时还增加了学习能力，提升了训练速度和训练效果.

3.4　并行编程模型

并行编程模型主要用于编写大规模并行程序[18]. 目前常用的并行编程模型有共享内存模式、消息传递模式[19]、多线程模式[20]以及数据并行模式[21].

在拥有全局内存时，可以使用共享内存模式[22]编写大规模并行程序. 在共享内存模式中，所有处理器共享一个内存空间. GALA 使用的便是这种方法，它使用单个节点的 48 个 CPU 核与 1 个 GPU 进行训练. 这一方法有着编程方式简单的优势，但是可扩展性不强，难以被用来构建大规模系统.

消息传递模式主要用于所有处理器无法共享相同的内存空间时. 在消息传递模式中，处理器之间通过网络进行互联. 相较于共享内存模式，这一编程模型有着比较好的可扩展性，符合并行强化学习对于算力扩展的需求. 本文使用消息传递模式构建了并行强化学习训练框架 PALA.

3.5　Gossip 算法

Gossip 算法[23]通常在一些端到端的网络中被用于实现异步通信[24]求平均. 假设整体的网络中存在 n 个需要进行通信的节点，每个节点有一个参数向量 $\boldsymbol{x}_i(i=1,\cdots,n)$，这些节点以端到端的网络进行连接. Gossip 算法的目的就在于计算出所有参数向量的平均值 $\frac{1}{n}\sum_{i=1}^{n}\boldsymbol{x}_i$. Gossip 的迭代形式为 $\boldsymbol{X}^{k+1}=$

$\boldsymbol{P}^k\boldsymbol{X}^k$，其中 $\boldsymbol{X}$ 是各个节点的参数向量按行拼接形成的矩阵，$\boldsymbol{P}$ 是聚合矩阵，k 代表第 k 次迭代. 聚合矩阵 $\boldsymbol{P}$ 定义了网络的拓扑结构，每次迭代中，节点 v_i 只需要与聚合矩阵元素 $p_{i,j}$ 不等于 0 的 v_j 通信即可. Gossip 算法能够减少通信量，并且减少同步次数，提升计算效率.

Gossip 算法是一个最终一致性算法. 使用 Gossip 算法的节点在经过有限次迭代后，各个节点包含的信息仍然存在一定的差异，这种差异导致 GALA 和 PALA 的策略模式属于近同策略. 在强化学习场景中，策略传递与模型融合对于一致性的要求不是很强，因此可以使用 Gossip 算法.

4 多个智能体并行的分布式强化学习框架

本节将对GALA进行简要的概述，并阐明 PALA 的相关细节.

4.1 基于 Gossip 算法的执行-学习者算法

GALA 的架构如图 3 所示. GALA-A2C 在单节点上设置 n 个 A2C 智能体，每个 A2C 智能体包含 1 个学习者和多个执行者，这些智能体 v 含有策略网络 $\pi(a_t|s_t;\omega_i)$ 和价值函数 $V(s_t;\theta_i)$. 智能体通过环境交互，收集样本并更新策略与价值函数. 在 GALA-A2C 中，每个智能体依照聚合矩阵与其他智能体相连，使用 Gossip 方法从与其相连的节点获取模型参数并聚合. GALA 在实现上使用了稀疏的拓扑结构和 Gossip 算法，在理论上能够将各个智能体的参数差距限制范围在一定范围内. 通过移除严格的策略一致性限制，GALA 提升了可扩展性和计算效率.

GALA-A2C 的伪代码如算法 1 所示. $p_{i,j}$ 是聚合矩阵 $\boldsymbol{P}$ 中的元素，$\boldsymbol{P}$ 定义了智能体互联的拓扑结构. 智能体 v_i 仅需要和 $p_{i,j}\neq 0$ 的智能体 v_j 交互信息. v_i 将从 $\mathcal{N}_i^{\text{in}}:=\{v_j\,|\,p_{i,j}>0\}$ 集合中得到最新的模型参数并聚合，并对 $\mathcal{N}_i^{\text{out}}:=\{v_j\,|\,p_{j,i}>0\}$ 集合中的节点发送模型参数以及其他信息.

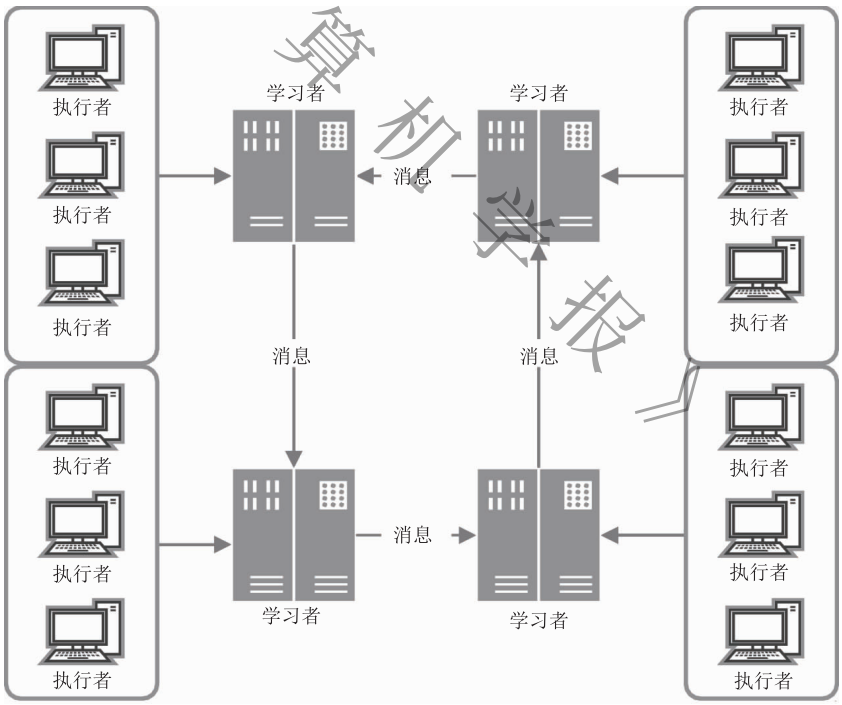


图 3 使用环状方式连接的 GALA 架构图

算法 1. 使用A2C作为智能体的GALA-A2C.

输入：初始化可训练参数 $x_i=(\omega_i,\theta_i)$

1. FOR $t=0,1,2,\cdots$, DO
2. 根据策略π_{ω_i}采取N步动作并存储轨迹(s_t,a_t,r_t,s_{t+1}).
3. 计算$G_t^N=\sum_{i=0}^{N-1}\gamma^i r_{t+i}+\gamma^N V(s_{t+n},\theta_i)$和优势函数$A_t=\sum_{i=0}^{N-1}\gamma^i r_{t+i}+\gamma^N V(s_{t+n},\theta_i)-V(s_t,\theta_i)$.
4. 利用$\nabla_\theta(G_t^N-V(s_t;\theta))^2$及策略梯度[25]，使用参数$x_i$执行 A2C 的优化步骤.
5. 将新参数 x_i广播给所有传出节点$\mathcal{N}_i^{\text{out}}$.
6. IF 缓存含有来自传入节点$\mathcal{N}_i^{\text{in}}$中$v_j$的信息$m_j$ THEN
7. $$x_i=\frac{1}{1+|\mathcal{N}_i^{\text{in}}|}\left(x_i+\sum_j m_j\right)$$
8. END IF
9. END FOR

输出：训练完成的参数 $x_i'=(\omega_i',\theta_i')$

GALA 的共享内存模型导致其难以在没有共享内存的节点间传递信息，因此部署 GALA 的最优方法是将其部署在含有多个 GPU 的单节点上.

4.2　PALA 的算法与结构

通过提升并行性和可扩展性可以提升强化学习算法的收敛速度. 局限于单个计算节点的强化学习算法会受到计算资源不足的影响而导致采样效率受限. PALA 通过提升可扩展性增强了收敛效率. 为了能够在集群内部没有共享内存的条件下扩展到所有节点并利用全局计算资源，本文使用消息传递模型构建了以 Gossip 模式进行消息交换的 PALA. PALA 的架构图如图 4 所示，算法如算法 2 所示. 相比与算法 1，PALA-A2C 算法使用一个中心服务器来管理所有的参数和锁. 这些参数和锁存储在位于中心服务器的数据代理中，数据代理的主要作用是协助智能体在网络之间传递消息. 在训练逻辑上，PALA 和 GALA 等价以便保证计算收敛.

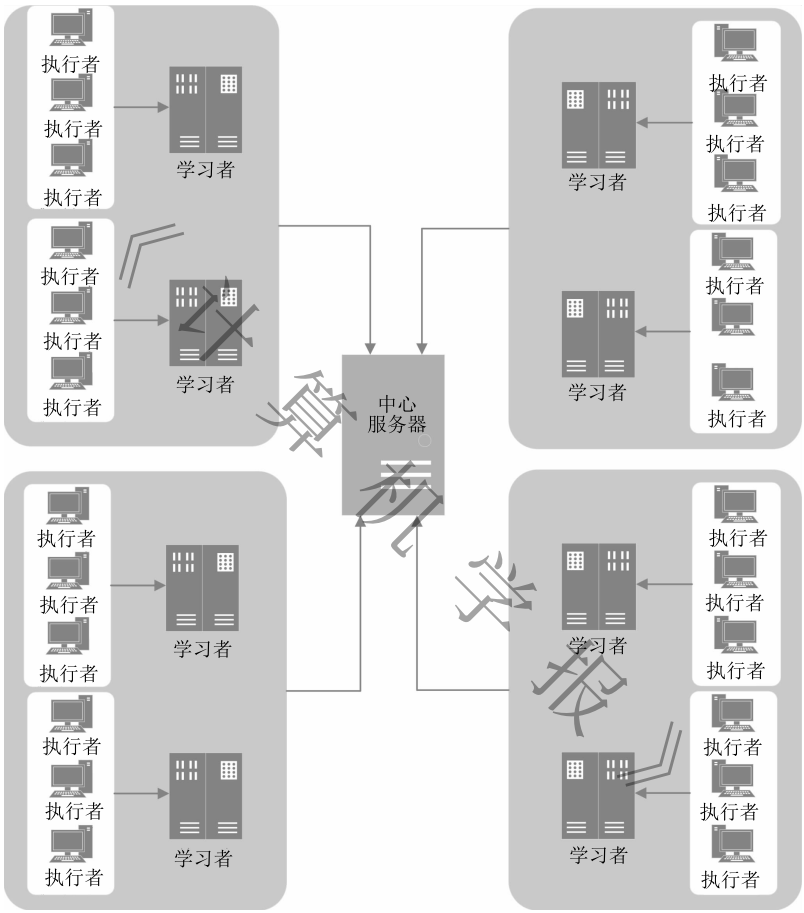


图 4　PALA 架构图

算法 2.　使用 A2C 作为智能体的 PALA-A2C.

输入：初始化可训练的参数 $x_i=(\omega_i,\theta_i)$，存储于服务器中的数据代理 S 以及连接拓扑 T

1. 从拓扑图 T 中得到消息传入节点的索引 $\mathcal{N}_i^{\text{in}}$ 以及消息传出节点的索引 $\mathcal{N}_i^{\text{out}}$
2. FOR $t=0,1,2,\cdots$, DO
3. 使用策略 π_{ω_i}，执行 N 步动作 $\{a_t\}$ 并存储 (s_t,a_t,r_t,s_{t+1})
4. 计算 $G_t^N=\sum_{i=0}^{N-1}\gamma^i r_{t+i}+\gamma^N V(s_{t+n},\theta_i)$ 的返回值以及优势函数 $A_t=\sum_{i=0}^{N-1}\gamma^i r_{t+i}+\gamma^N V(s_{t+n},\theta_i)-V(s_t,\theta_i)$
5. 执行 $\nabla_\theta(G_t^N-V(s_t;\theta))^2$ 来优化 θ 并且执行大数据批的梯度下降来优化 ω
6. 将模型参数 x_i 发送给 S 并插入到数据代理 $S[\mathcal{N}_i^{\text{out}}]$ 中
7. IF 数据代理 S 包含来自传入节点 v_j 的模型 m_j THEN
8. $\quad x_i=\frac{1}{1+|\mathcal{N}_i^{\text{in}}|}\left(x_i+\sum_j m_j\right)$
9. END IF
10. END FOR

输出：训练完成的参数 $x_i'=(\omega_i',\theta_i')$

证明 PALA 计算收敛的关键在于证明 PALA 在经过 k 步迭代后，所有智能体的参数堆叠而成向量 $\boldsymbol{X}$ 和全局智能体参数的平均值堆叠而成的向量 $\overline{\boldsymbol{X}}^k$ 的差距能够缩减在一个范围内. 如果他们的差异过大，那么就不能收敛.

PALA 和 GALA 均使用 Gossip 方法进行参数传递和共享. PALA 使用跨节点进程组织训练并使用数据代理在进程之间传递消息,由于通过网络,因此节点之间变为了消息传递模式而非共享内存模式,切换并行编程模型后 PALA 仍然符合 GALA 证明中的收敛条件.

在 GALA 的收敛性证明中有以下几个条件:首先,连接各个节点的图是有向且连续的,节点之间按照有向图方向存在收发顺序;其次,消息的传递存在延迟,在第 k 次迭代发送出来的参数信息会在第 $k'(k'>k)$ 次迭代被接受;再次,用于定义节点之间通信拓扑图的聚合矩阵 $\boldsymbol{P}$ 是行随机的(每行元素非负且求和值为 1). 设所有节点的参数堆叠而成向量 $\boldsymbol{X}$,所有的参数更新量堆叠成向量 $\boldsymbol{G}$,α 代表学习率,$\beta(0\leq\beta\leq1)$ 代表迭代过程中通信拓扑图的联合谱半径. 在这些条件限制下,GALA 证明得到 $\|X^{(k+1)}-\overline{X}^{(k+1)}\|\leq\alpha\sum_{s=0}^{k}\beta^{k+1-s}\|G^{(s)}\|$,$X^{(k+1)}$ 表示所有智能体在 $k+1$ 步时的参数值,$\overline{X}^{(k+1)}$ 表示所有智能体在 $k+1$ 步时的参数平均值. 这个结果表明,随着训练不断开展,各个智能体的差距可以缩减到一个越来越小的边界内.

PALA为了维持可收敛性,在训练逻辑上和GALA保持一致,主要增加了分布式实现方式,其中 PALA 使用了环状有向图,切换了并行编程模型,仍然使用 Gossip 方法作为通信机制. 环状有向图符合 GALA 论证中图是有向的这一要求,并且代表有向环的聚合矩阵 $\boldsymbol{P}$ 仍然是行随机的. 同样的,PALA 使用的并行编程模型在信息传递中也存在通信延迟. 这证明 PALA 在模型的可收敛性上保持了相同的条件,可以使用 GALA 的结论.

PALA 内含有设置 n 个 A2C 个智能体,每个智能体按照顺序进行通信并组织成环状,智能体之间使用 Gossip 算法进行通信,使用模型融合方法求整体梯度.

智能体直接面向数据代理收发信息. 在实现上,数据代理以全局可访问的列表进行实现,每个智能体会根据相应的索引在列表中查询信息. 由于每次传递的数据量较小,因此使用一个中心服务器对消息进行管理是可行的. 数据代理位于中心服务器中,使用数据代理能够简化消息传递模型的实现,保证多机之间构建通信的简便性. 智能体使用存储在 $\mathcal{N}_i^{\text{in}}$ 与 $\mathcal{N}_i^{\text{out}}$ 中的节点索引来找到执行模型聚合的节点. 通过使用消息传递模型,PALA 能够扩展到整个集群并且部署更多智能体从而加速训练. 为了在收敛上得到理论保证,除了并行模型外,PALA 在 Gossip 算法等方面与 GALA 在逻辑上等价.

在实现细节上,中心服务器中的数据代理的构建使用 Python 库中多线程库进行实现,多进程库中存在以列表形式存储的数据代理 ListProxy,这种数据代理可以在多机传递信息的同时提供数据操作. 多进程库可以直接兼容数据列表并进行读写. 同时,数据代理对于数据格式有着比较好的兼容性,PALA 通过在数据代理上自定义了一套锁方法实现了消息传递的有序性. 当前的一些分布式库如 Redis 等在多机之间难以定义进程锁,而锁在维持模型聚合顺序上是必须的,如果没有锁维持模型聚合顺序,PALA 的收敛条件之一就会被打破从而无法收敛. 因此本文没有采用 Redis 等分布式数据库.

PALA 会在智能体发送与接受模型时对模型参数与锁进行序列化和解序列化. 实现中使用 Pickle 和 Base64[26] 进行编码和解码. 序列化的目的在于使其能够通过网络进行传输并且符合 CUDA 的要求. 在发送模型参数时,参数需要从 GPU 中导出并且进行序列化,在接受时则需要解序列化参数并将其导入 GPU. 序列化过程的本质是将模型参数转换为网络内部可以传输的字符串,在其他节点接受到序列化后的模型参数后再对其进行解序列化并加载进入 GPU 中.

锁用于维持智能体的沟通顺序和通信延迟,即智能体在其他节点读取完成后才能对其发送信息,在本身的聚合操作完成后才能读取新一轮信息. 原始的锁实现形式无法满足网络传输的需求,我们重新设计了可以在网络之间进行传输的锁,用于触发进程的行为. 锁使用循环等待来维持先后顺序,每一个智能体使用布尔类型的标志触发锁. 采用这一方式的主要原因是传统的锁会由于线程不安全等问题无法使用网络进行传输,使用循环等待在保证了安全的同时使用了最简单的基础数据类型来保证符合数据代理要求.

实验结果表明,通过将智能体分散到集群上降低单个节点负载,并且增加智能体数量,PALA 达到了更好的时钟表现,在相同训练步数下得到了更高的得分.

算法 2 中存在智能体 v_i 对 $\mathcal{N}_i^{\text{in}}$ 和 $\mathcal{N}_i^{\text{out}}$ 的隐式等待. 对于智能体 v_i,它会在 $\mathcal{N}_i^{\text{in}}$ 中的所有智能体完成写入之后再进行模型平均,而对于在 $\mathcal{N}_i^{\text{out}}$ 中的节点 v_j,v_i 会在 v_j 读取完上一轮所有的信息后再发送新的信息. 在 PALA 中,读写顺序必须存在,没有这些

顺序，Gossip 过程以及更多的智能体就不能带来增益. 因此，为了维持这一顺序并且实现高效的训练，隐式等待是必须的.

5　多个智能体并行的分布式强化学习框架

5.1　实验设置

实验在含有 4 个计算节点的集群上进行，每个计算节点含有两个 Intel(R) XEON(R) 5220 CPU，共 72 个 CPU 核，一块 NVIDIA Tesla V100 GPU，256 GB 内存. 节点之间以千兆网互联. 节点运行的操作系统为 Ubuntu 16.04.4. NVIDIA GPU 驱动版本为 440.33.1. CUDA 计算库的版本为 10.2. 使用 PyTorch 1.3.1[27] 实现 PALA 以及要对比的方法.

在实验安排上，本文以 PALA-A2C 与 GALA-A2C、A2C、IMPALA 进行比较. IMPALA 与 A2C 通过 Ray 1.1.0[28]、RLlib[29] 和 Tune[30] 实现，这些方法作为基线用于对比. RLlib 是一个开源的强化学习库，有较高的可扩展性. 比较性能表现的环境为六个常用的 Atari 2600 游戏. 六个游戏分别为 BeamRider、Breakout、Pong、Qbert、Seaquest、SpaceInvaders. 由于其自身的设计，GALA-A2C 与 A2C 将在单节点上对其进行训练；PALA-A2C 与 IMPALA 使用集群上的所有节点进行训练. PALA-A2C 训练时首先设定好本次训练的节点数量，节点编号等参数，启动中心服务器，再启动其他分布于集群上各个计算节点的训练节点，此时训练开始正常进行. 实验结果图的曲线取三次实验的平均结果，实验结果表中的结果为 10 次实验的平均结果及其标准差，实验中的步数指代训练时间步.

在超参数设置中，PALA-A2C 与 GALA-A2C 使用相同的配置.

通过实验表明，大的数据批(batch size)能够有效稳定训练并且提升计算效率[31]. 因此在超参数设置中对 PALA-A2C 与 GALA-A2C 都使用了较大的数据批. 在智能体的设置方面，PALA-A2C 与 GALA-A2C 保持相同，每个智能体含有 1 个学习者与 16 个执行者. 优化器均为 RMSProp. A2C 与 IMPALA 的超参数设置与 RLlib 中的默认设置相同.

为了探究不同超参数对于训练效果的影响，本文设计了一系列消融实验. 消融实验主要由两部分组成：计算负载的影响以及增加智能体数量的影响.

对比实验主要对比了不同方法之间的时间效率以及最终产出策略的得分. PALA-A2C 以及对比方法将被训练 5.5 h 并分析时间效率. 在训练 4 千万步之后，本文将取出每种方法最后产出的策略进行对比.

5.2　消融实验

本节主要分为两部分：第一部分将展示随着智能体数量增加计算负载提升所带来的影响，实验结果表明高工作负载会降低计算效率；第二部分将展示智能体数量增加对于训练效果带来的提升. 本节也将分析在增加智能体数量时，隐式等待所带来的影响.

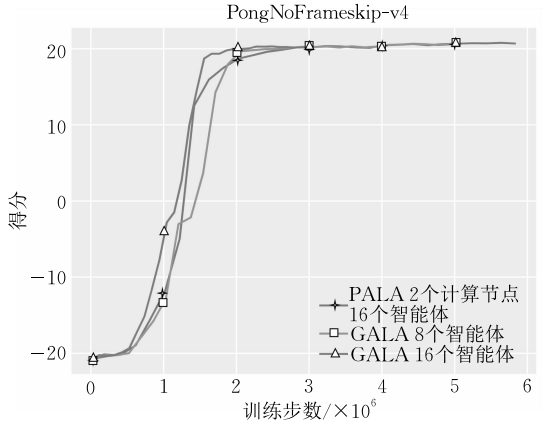
5.2.1　计算负载的影响

更多的智能体并行地探索环境，收集更多不同类型的样本对于训练是有益的，但是过高的计算负载会影响计算效率. GALA 使用的共享内存模式导致其难以扩展，增加更多智能体带来的负载无法被均衡给其他节点，从而导致训练时间增加并且抵消一部分智能体增加带来的效果. 由于 PALA 使用可扩展性更强的消息传递模式，因此增加节点数量时带来的负载，能够均衡地分散到集群上的各个节点上.

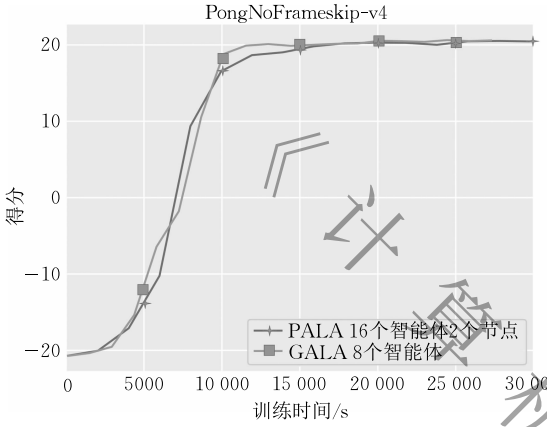
为了分析计算负载的边界，本文设计了 PALA-A2C 使用 144 个 CPU 核与 2 个 GPU 在游戏 Pong 中训练智能体的实验，通过使用三种不同的情况来对比分析计算负载带来的影响. 三种情况分别是：GALA-A2C 的计算负载不同；PALA-A2C 和 GALA-A2C 负载相同且 PALA-A2C 不使用负载均衡；PALA-A2C 和 GALA-A2C 框架内总的负载相同但是 PALA-A2C 使用负载均衡.

在图 5(a)中可以发现，随着智能体数量的增加，收敛所需要的训练步数更少，意味着更多的智能体对环境进行探索可以为训练带来正向收益.

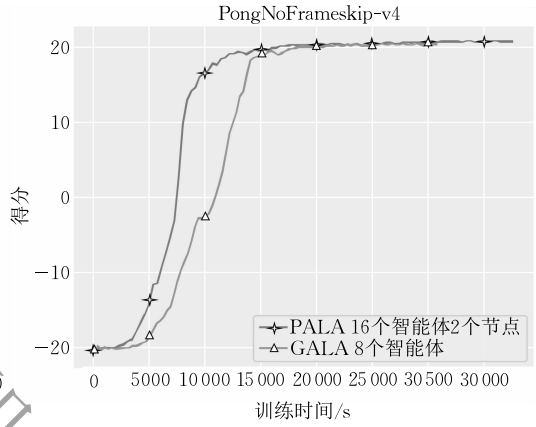
图 5(b)表示，当节点负载相同时，PALA-A2C 与 GALA-A2C 的时间表现基本一致. 即，在单机低负载的情况下，PALA-A2C 和 GALA-A2C 的时间表现没有很大差距. 在节点负载不同，但总的智能体数量相同时，通过图 5(c)可以发现，同样拥有 16 个智能体的 PALA-A2C 与 GALA-A2C 的时间表现出现了差距，PALA-A2C 的训练速度更快，耗时更短. GALA-A2C 由于无法利用额外的计算节点，因此在使用 16 个智能体时，单个节点承载了更高的工作负载，降低了时间表现. PALA-A2C 通过将负载均衡地分布到多个节点上降低了单个节点的工作负载，提升了训练效率. 当训练结果的得分达到 20 时，PALA-A2C 相较于 GALA-A2C 能够加速 25%.



(a) 训练步数-得分图



(b) PALA与含有8个智能体的GALA的对比训练时间-得分图



(c) PALA与含有16个智能体的GALA的对比训练时间-得分图

图 5 不同计算负载的结果

实验结果表明过重的工作负载对时间效率有负向的影响，而 PALA 则通过将整体负载均衡分散到整个集群上提升了时间表现. 负载过重导致效率降低的原因是高负载情况下会需要更多时间完成 Gossip 流程，从而使隐式同步的时间变长，直观的表现是在循环等待锁上消耗的时间更长. 如果将所有智能体部署在单个节点上，那么智能体需要等待其他智能体释放资源之后再占用，从而需要更长的时间完成 Gossip 流程，而 PALA 通过提升可扩展性得到了更多可用的计算资源，使得智能体不需要再耗时等待资源释放. 下一节将分析负载过重带来的具体影响.

5.2.2 智能体数量增加带来的影响

在取得良好的时间性能的同时，PALA 可以在框架内部署更多的智能体，实验表明更多的智能体可以保证 PALA 在相同的训练步数下取得更好的训练结果.

这一实验使用 288 个 CPU 核与 4 个 GPU 来观测增加智能体所带来的影响，在游戏 BeamRider 上进行训练. 使用更多的结算节点可以保证框架在部署更多的智能体时能够维持较低的负载. 在本节的实验中，整个框架的智能体从 8 到 32 个不等.

图 6 表明，当集群中智能体数量增加时，训练的效果也可以得到相应的增益. 在图 6 中可以发现，当框架中智能体的数量增加，相同步数下的得分也会增长.

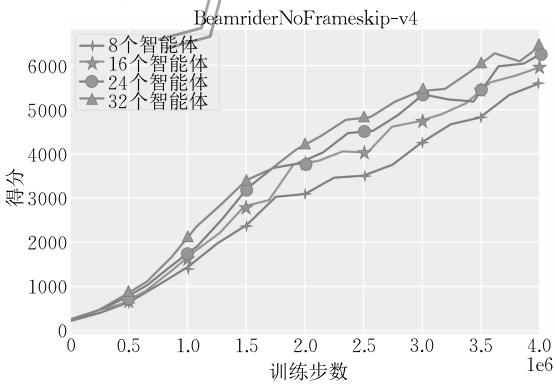


图 6 增加智能体数量后的训练步数-得分图

PALA-A2C 中的智能体设置与 GALA-A2C 保持一致，都含有一个学习者和 16 个执行者. 框架内的智能体增多会导致探索环境的执行者数量增多，从而采样出来更加丰富的轨迹. 使用不同情况下的轨迹可以让智能体掌握多种不同的能力，让其更好地适应环境. 同时，并行更多的执行者可以极大提升采样效率，满足训练需求.

在Gossip过程方面，PALA和GALA是逻辑上等价的，PALA也需要加入隐式等待. 框架内含16个智能体与12个智能体时的时间表现基本一致，但是当节点数量增加到24个时，时间效率出现了衰减. 如图7所示.

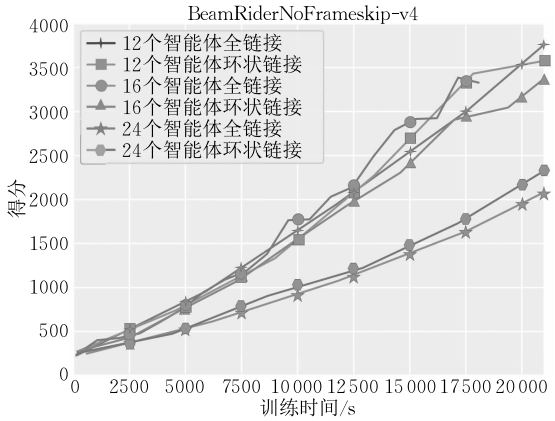


图 7　在不同拓扑结构下增加智能体的时间效率图

智能体数量增加导致时间效率衰减的原因如图8所示. 实验表明，当框架内设置了超过24个智能体时，进行模型聚合所需的时间相比部署16或12个智能体几乎翻倍. 在互联的网络环境限制下，一次迭代中，由于没有显式同步，当智能体数量过多时，消息到达的时间方差就会变大. 如前文所述，Gossip过程需要使用锁来维持一定的顺序，否则智能体数量的增加就不会产生增益. 智能体v_i需要等待$\mathcal{N}_i^{\text{in}}$集合中的智能体消息全部到达后才能开始聚合. 大的到达时间方差会导致智能体v_i等待时间变长，直观表现就是锁的占用时间更长.

在实际的设置中，当我们需要把聚合时间控制到较低水平时，智能体数量的设置存在一个边界. 在当前的网络条件限制下，智能体数量带来的性能边界为16. 当超过这一边界时，聚合所需的时间会随着节点数量的增长而增长.

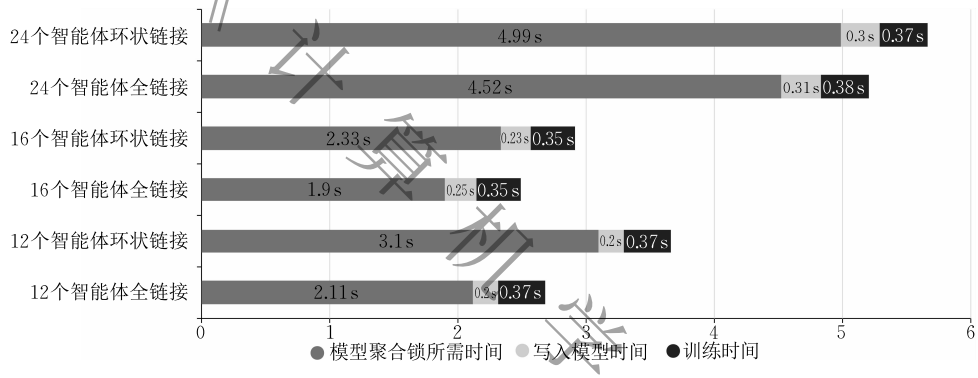
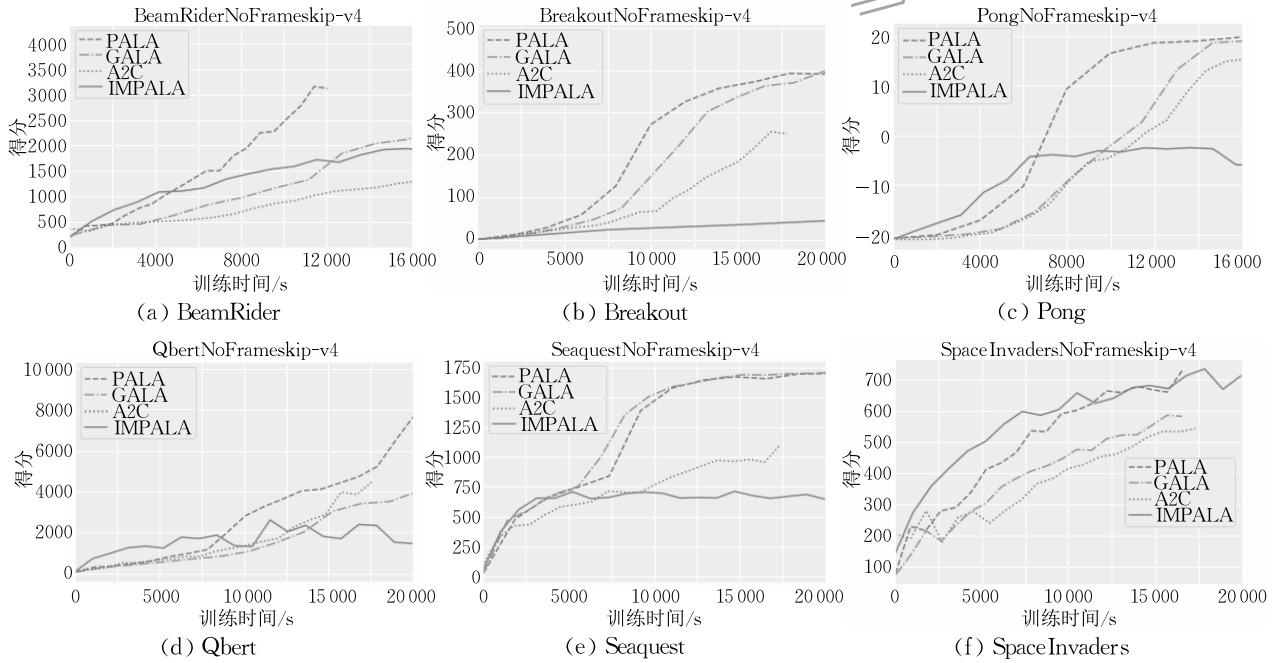


图 8　训练过程中不同步骤消耗的时间占比

5.3　与当前分布式深度强化学习方法的比较

本节将PALA-A2C的收敛效率以及训练完成后的策略表现与GALA-A2C、A2C和IMPALA进行比较. 结果如图9和表1所示.



(a) BeamRider　(b) Breakout　(c) Pong

(d) Qbert　(e) Seaquest　(f) SpaceInvaders

图 9　PALA、GALA、IMPALA 和 A2C 的训练时间效率比较

表 1 A2C、IMPALA、GALA-A2C 和 PALA-A2C 的最终策略水平对比

	Steps	BeamRider	Breakout	Pong	Qbert	Seaquest	SpaceInvaders
IMPALA	40M	7833±196	248±5	**21±0**	13796±459	1751±117	2716±267
A2C	40M	9829±1355	495±57	**21±0**	19928±99	1894±6	3021±36
GALA-A2C	25M	9500±1020	690±72	**21±0**	18810±37	1874±4	2726±189
GALA-A2C	40M	10188±1316	690±72	**21±0**	20150±28	1892±6	**3074±69**
PALA-A2C	25M	13394±4317	798±127	**21±0**	26350±353	2638±6	2985±83
PALA-A2C	40M	**14125±**3645	**848±**45	**21±0**	**26375±**133	**2720±**21	2987±117

5.3.1 时间效率

如图 9 所示，相较于 GALA-A2C，PALA-A2C 缩减了训练时间. GALA-A2C 与 A2C 在一个计算节点上进行扩展，而 PALA-A2C 和 IMPALA 则可以扩展到整个集群. 在这个实验中，并行训练框架均使用 144 个 CPU 核与 2 个 GPU 进行实验，GALA-A2C 和 A2C 使用 72 个 CPU 核与 1 个 GPU 进行实验. PALA-A2C 与 GALA-A2C 均使用 16 个智能体. 每个 Atari 2600 游戏都被训练约 5.5 h.

在六个游戏中，PALA-A2C 都超过了 GALA-A2C. 结果表明，在达到相同的水平下，相比于 GALA-A2C，PALA-A2C 能节约约 20% 的训练时间. 通过使用消息传递模型并利用基于 Gossip 算法的通信模式，PALA-A2C 得到了更好的计算效率.

5.3.2 最终策略表现

这一部分将比较训练完成后的策略水平.

图 9 表明，PALA-A2C 有着良好的计算性能. 在训练完成后，选择每种方法最优的策略作为最终策略，通过 10 次验证迭代来分析其最终表现. 表 1 展示了最终评估的平均得分以及在 10 次验证迭代中的标准差.

相比与其他方法，PALA-A2C 的最终策略几乎在每个游戏中都得到了最高的得分. 值得注意的是，PALA-A2C 的智能体在训练 2500 万步时就已经取得了和 GALA-A2C 训练 4000 万步相同的得分. 综合这些实验结果，得出的结论是，相比于 GALA，PALA 成功地减少了至少约 20% 的训练时间，并且相比于其他训练方法，其最终的结果表现更优.

在基础算法相同的情况下，PALA-A2C 通过突破样本局限性带来的局部最优取得了更好的效果. 原因可以参考样本图. 如图 10 所示. 以 Seaquest 环境为例，在 Seaquest 环境中，智能体控制潜艇在水下射击目标，潜艇在水下不断消耗氧气，上浮则可以补充氧气. 潜艇射中目标则得分，潜艇被攻击则死亡，同时如果潜艇氧气耗尽同样会死亡. 在 GALA-A2C 的训练过程中，由于潜艇在水下射击到目标时才会得分，而上浮会一直消耗氧气，因此智能体只会采样潜艇在水下攻击的样本，很难采集到的潜艇上浮补充氧气的样本. 这意味着智能体受限于局部最优，潜艇只会在下面射击拿到局部最优值，而不会上浮，因为上浮样本在局部会带来负向奖励. GALA-A2C 仅仅会采样到很少的上浮样本，因此这些样本无法对训练起到决定性影响，而这些样本的缺失会导致智能体受限于局部最优. 在不会上浮的情况下，智能体最高可以得到约 1800 分.

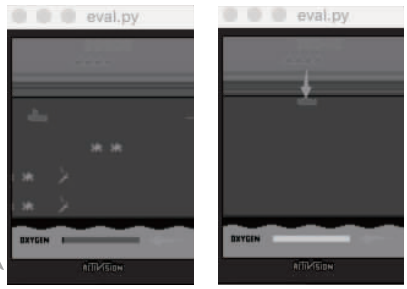


图 10 Seaquest 游戏中潜艇补充氧气样例

而 PALA-A2C 由于可以扩展到整个集群，因此 PALA-A2C 可以同时并行数量更多的智能体，这些智能体并行采样，采样到更多潜艇上浮的样本，这些样本可以帮助智能体克服局部最优，从而使得智能体控制的潜艇学会上浮. 学会上浮的潜艇可以在整个训练过程中执行多次下潜-攻击-上浮补充氧气的过程，从而使用有限的生命得到更多分数.

5.4 关于实验结果的讨论与分析

PALA 能够在提升时间效率的同时增强训练效果的原因在于，PALA 能够使用更多的计算资源，从而允许更多的智能体同时对环境进行探索. 通过使用更多的智能体与不同的环境交互，收集更多样本，智能体能够快速的提升自身的策略.

需要指出的一点是，学习者的数量需要和执行者的数量相匹配，即深度强化学习中整个框架的学习能力需要与采样的能力相匹配. 由于 PALA-A2C 在逻辑上与 GALA-A2C 等价，我们使用 GALA-A2C 来验证学习者的数量需要伴随执行者的增加

而增加.

实验采用了两种设置，分别是 256 个执行者加 4 个学习者与 128 个执行者加 8 个学习者. 使用 BeamRider 作为训练环境. 结果如图 11 所示.

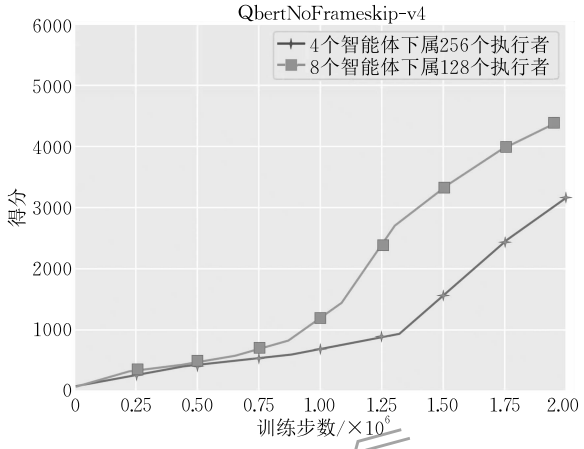


图 11　对 GALA-A2C 设置 4 个到 8 个含有不同执行者数量的智能体

从图中可以发现，更多的执行者并没有对设置 4 个智能体的框架带来更多收益. 伴随着采样能力的增加，学习能力也需要随之增加，单纯增加采样能力不会带来很高的收益.

PALA 能够同时增加执行者与学习者的数量并保持其数量匹配. PALA 通过将负载均衡地分散到计算集群中平衡了同时增加执行者与学习者带来的大量的工作负载，提升了计算效率. 这也就是 PALA 能够使用更短的计算时间达成更好的训练效果的原因.

在消融实验中我们可以发现，随着智能体数量的增加，训练步数-得分得到了正向的影响而训练时间-得分会受到负向影响. 在不同的硬件条件限制下，我们需要设置不同的智能体数量以便平衡这两个指标. 当拥有更好的计算资源和网络条件时，框架内可以设置更多的智能体，从而在不损害时间效率的条件下提升训练步数-得分这一指标.

如图 8 所展示的，聚合时间占据了训练时间的主要部分. 聚合时间主要受到两部分影响：智能体数量以及网络环境. 聚合时间主要用于维持写入和聚合顺序并保证读写的安全. 维持写入和聚合顺序是必要的，一旦破坏这一顺序，PALA-A2C 的表现将会等同于 A2C.

聚合时间同样受到网络带宽影响. 过多的智能体数量会完全填满带宽，从而导致等待. 在未来，我们将会让 PALA 具备拓扑敏感的特性，这一特性有助于减少节点之间消息传递的代价.

在超参数的设置上，数据批的大小是影响训练结果的关键参数. 更大的数据批可以稳定训练并且提升效率. 图 12 表明，当数据批的大小小于 15 时，时间效率会随着数据批的增大而增大. 而在数据批的大小超过 15 时，更大的数据批将不会带来增益. 数据批的大小需要按照实际情况进行设定，过大的数据批不会带来更多的增益，而如果数据批过小，则需要更多训练步数来达到相同水平. 通过实验表明，最合适的数据批大小在 15.

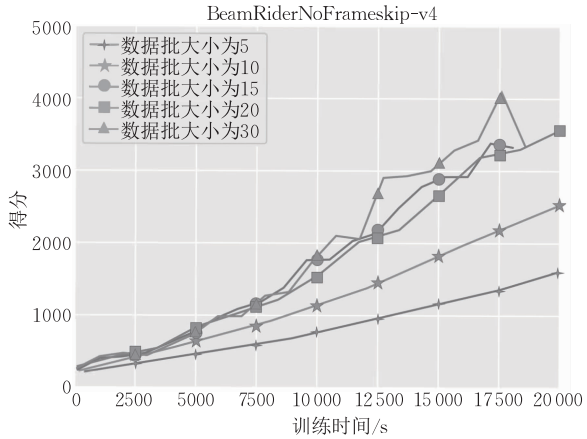


图 12　使用不同数据批大小时的时间性能表现

6　结束语

本文提出了一种并行执行-学习者强化学习训练方法. 这一方法能够有效加速深度强化学习模型的训练. 与其他同策略学习方法不同，PALA 主要集中于将并行的智能体分布到远程节点上. 我们的实验表明，PALA 能够有效地在集群内部进行扩展. PALA 通过优化并行模型，实现多节点共享的锁和模型序列化方法，让其能够使用多个计算节点平衡负载，从而在降低单节点工作负载的同时增加智能体数量，实现减少总体训练时间、提升训练效果的目标.

参　考　文　献

[1] Volodymyr M, Koray K, David S, et al. Human-level control through deep reinforcement learning. Nature, 2015, 518(7540): 529-533

[2] Vinyals O, Babuschkin I, Czarnecki W M, et al. Grandmaster level in StarCraft II using multi-agent reinforcement learning. Nature, 2019, 575(7782): 350-354

[3] Silver D, Huang A, Maddison C J, et al. Mastering the game of Go with deep neural networks and tree search. Nature,

2016, 529(7587): 484-489
[4] Silver D, Hubert T, Schrittwieser J, et al. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. Science, 2018, 362(6419): 1140-1144
[5] Tebbe J, Krauch L, Gao Y, et al. Sample-efficient reinforcement learning in robotic table tennis//Proceedings of the 2021 IEEE International Conference on Robotics and Automation (ICRA). Xi'an, China, 2021: 4171-4178
[6] Zhang Y, Clavera I, Tsai B, et al. Asynchronous methods for model-based reinforcement learning. arXiv preprint arXiv: 1910.12453, 2019
[7] Horgan D, Quan J, Budden D, et al. Distributed prioritized experience replay. arXiv preprint arXiv:1803.00933, 2018
[8] Assran M, Romoff J, Ballas N, et al. Gossip-based actor-learner architectures for deep reinforcement learning//Proceedings of the 33rd International Conference on Neural Information Processing Systems. Vancouver, Canada, 2019: 13320-13330
[9] Yu H, Yang S, Zhu S. Parallel restarted SGD with faster convergence and less communication: Demystifying why model averaging works for deep learning//Proceedings of the AAAI Conference on Artificial Intelligence. Hawaii, USA, 2019, 33(1): 5693-5700
[10] Fujimoto S, Hoof H, Meger D. Addressing function approximation error in actor-critic methods//Proceedings of the International Conference on Machine Learning. Stockholm, Sweden, 2018: 1587-1596
[11] Espeholt L, Soyer H, Munos R, et al. IMPALA: Scalable distributed deep-RL with importance weighted actor-learner architectures//Proceedings of the International Conference on Machine Learning. Stockholm, Sweden, 2018: 1407-1416
[12] Machado M C, Bellemare M G, Talvitie E, et al. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. Journal of Artificial Intelligence Research, 2018, 61: 523-562
[13] Konda V R. Actor-critic algorithms. Advances in Neural Information Processing Systems, 2000, 12: 1008-1014
[14] Espeholt L, Marinier R, Stanczyk P, et al. Seed RL: Scalable and efficient deep-RL with accelerated central inference. arXiv preprint arXiv:1910.06591, 2019
[15] Zahavy T, Xu Z, Veeriah V, et al. A self-tuning actor-critic algorithm. Advances in Neural Information Processing Systems, 2020, 33: 20913-20924
[16] Sutton R, Barto A. Reinforcement Learning: An Introduction. Cambridge, UK: MIT Press, 1998
[17] Sutton R S. Policy gradient methods for reinforcement learning with function approximation. Advances in Neural Information Processing Systems, 2000, 12: 1057-1063
[18] Mcbryan O A. An overview of message passing environments. Parallel Computing, 1994, 20(4): 417-444
[19] Skillicorn D B, Talia D. Models and languages for parallel computation. ACM Computing Surveys, 1998, 30(2): 123-169
[20] Graham R L. Bounds on multiprocessing timing anomalies. SIAM Journal on Applied Mathematics, 1969, 17(2): 416-429
[21] Hillis W D, Steele G. Data parallel algorithms. Communications of the ACM, 1986, 29: 12(12): 1170-1183
[22] Dagum L, Menon R, et al. OpenMP: An industry standard API for shared-memory programming. Computational Science & Engineering, 1998, 5(1): 46-55
[23] Shah D. Network gossip algorithms//Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2009). Taipei, China, 2009: 3673-3676
[24] Boyd S, Ghosh A, Prabhakar B, et al. Gossip algorithms: Design, analysis and applications//Proceedings of the IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Miani, USA, 2005, 3: 1653-1664
[25] Dhariwal P, Hesse C, Klimov O, et al. OpenAI baselines. https://github.com/openai/baselines, 2017
[26] Muła W, Lemire D. Faster Base64 encoding and decoding using AVX2 instructions. ACM Transactions on the Web (TWEB), 2018, 12(3): 1-26
[27] Paszke A, Gross S, Chintala S, et al. Automatic differentiation in PyTorch//Proceedings of the International Conference on Neural Information Processing Systems Workshop AutoDiff Submission. Long Beach, USA, 2017: 1-4
[28] Moritz P, Nishihara R, Wang S, et al. Ray: A distributed framework for emerging AI applications//Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18). Carlsbad, USA, 2018: 561-577
[29] Liang E, Liaw R, Nishihara R, et al. RLlib: Abstractions for distributed reinforcement learning//Proceedings of the International Conference on Machine Learning. Stockholm, Sweden, 2018: 3053-3062
[30] Liaw R, Liang E, Nishihara R, et al. Tune: A research platform for distributed model selection and training. arXiv preprint arXiv:1807.05118, 2018
[31] Stooke A, Abbeel P. Accelerated methods for deep reinforcement learning. arXiv preprint arXiv:1803.02811, 2018

附录 1.

A2C 和 IMPALA 使用的关键超参数为：A2C：学习率为 0.0001，使用 reward 裁剪，每个环境使用 16 个 worker，数据批大小为 200. IMPALA：学习率为 0.0005，每个环境使用 16 个 worker，数据批大小为 500，使用 adam 作为优化器，学习率衰减率为 0.99，不使用动量. 其他参数设置与 1.0.0 版本相同.

SUN Zheng-Lun, Ph. D. candidate. His research areas are intelligent computing (reinforcement learning, deep learning, etc.), high-performance computing (parallel computing, etc.).

QIAO Peng, Ph. D., assistant researcher. His research areas are high-performance computing (parallel computing, heterogeneous computing, etc.), intelligent computing (reinforcement learning, distributed machine learning, etc.), artificial intelligence applications (computer vision).

DOU Yong, Ph. D., researcher. His research focuses on high-performance computing, intelligent computing (machine learning, deep learning, etc.).

LI Qing-Qing, M. S. candidate. Her research interests include artificial intelligence applications (natural language processing), intelligent computing (reinforcement learning).

LI Rong-Chun, Ph. D., associate researcher. His research interests focus on high-performance computing, intelligent computing (deep learning), artificial intelligence applications (face recognition, video understanding, etc).

Background

Deep Reinforcement Learning (DRL) has achieved great success in recent years and even outperforms humans in several domains. But it's a computational resource-consuming and time-consuming task.

To speed up the training of DRL agents, many distributed DRL architectures are proposed.

However, synchronize, policy consistency, and scalability remains a problem.

One of the previous DRL architecture GALA (Gossip-based Actor-learner Architecture) provides a way to synchronize and policy consistency problems but only scale on a single node.

To improve the scalability, we proposed parallel actor-learner architecture (PALA).

By using the Gossip algorithm and message-passing programming model, PALA can reach all nodes of the cluster when there is no shared memory connecting each other.

We also proposed a way to transfer model parameters and event through the network which reduce the complexity of building a distributed DRL framework.

Like GALA, PALA allows the policy of each agent to remain close but not the same, removing the limit of frequent synchronize while distributing agents in remote nodes.

As a practical algorithm, we use A2C(Advanced actor-critic algorithm) as the agent in PALA and proposed PALA-A2C.

A series of experiments are designed to evaluate the scalability and performance of PALA.

Compared with previous work, PALA reduce the training time by more than 20% and reach the best final policy in almost every Atari 2600 environment.

This paper is supported by the National Natural Science Foundation of China under Grant No. 61732018, 61902415, 61972409 and the Open Fund of Science and Technology on Parallel and Distributed Processing Laboratory (PDL) under Grant No. WDZC20205500104.