Accelerating Convergence in Distributed Reinforcement Learning via Asynchronous PPO

Asel Nurlanbek kyzy
Future Convergence Engineering
Korea University
of Technology and Education
Cheonan, South Korea
aselbaekki@koreatech.ac.kr

Chang-Hun Ji
Future Convergence Engineering
Korea University
of Technology and Education
Cheonan, South Korea
koir5660@koreatech.ac.kr

Yohan Choi
Future Convergence Engineering
Korea University
of Technology and Education
Cheonan, South Korea
yoweif@koreatech.ac.kr

Yeong-Jun Seok
Future Convergence Engineering
Korea University
of Technology and Education
Cheonan, South Korea
dsb04163@koreatech.ac.kr

Ihsan Ullah
Department of Computer
Science and Engineering
Yuan Ze University
Taoyuan 32003,Taiwan.
ihsan@saturn.yzu.edu.tw

Youn-Hee Han
Future Convergence Engineering
Korea University
of Technology and Education
Cheonan, South Korea
yhhan@koreatech.ac.kr

***Abstract*—Asynchronous Proximal Policy Optimization (APPO) has emerged as a crucial framework for achieving scalability in distributed reinforcement learning. In this paper, we propose an enhanced APPO framework that addresses critical challenges in gradient synchronization and inter-worker communication. Our framework introduces optimized mechanisms to maintain training stability and efficiency while minimizing additional computational overhead. Through extensive experiments, we demonstrate that the proposed approach achieves superior performance and accelerates convergence compared to traditional APPO frameworks. These improvements advance the stability and scalability of distributed reinforcement learning systems, making them more suitable for large-scale applications.**

***Index Terms*—Distributed Reinforcement Learning, Asynchronous PPO, Gradient Update Optimization**

I. Introduction

Reinforcement learning (RL) has achieved remarkable progress in recent years, driven by advances in distributed learning systems. A significant breakthrough in this domain was the introduction of asynchronous frameworks, pioneered by the Asynchronous Advantage Actor-Critic (A3C) algorithm [1]. A3C demonstrated that asynchronous updates from multiple agents could stabilize training and improve convergence speed, effectively decoupling updates between workers and enabling efficient scaling across devices and environments. These asynchronous frameworks laid the foundation for scalable distributed RL systems [2], addressing challenges such as synchronization bottlenecks and slow convergence.

These asynchronous frameworks paved the way for the development of more advanced policy optimization methods in distributed RL. One such method is Proximal Policy Optimization (PPO) [3], which introduced a policy optimization approach that balances exploration and exploitation while ensuring training stability. PPO's robustness and simplicity made it a widely adopted framework for both single-agent and distributed RL settings. By combining PPO's optimization framework with asynchronous updates, researchers developed the APPO algorithm—a powerful tool for large-scale RL applications. APPO leverages the strengths of both frameworks, offering significant improvements in scalability and efficiency.

Despite these advancements, traditional APPO frameworks encounter several challenges, such as synchronization bottlenecks and slow convergence, which limit their scalability in distributed environments. These challenges arise due to the reliance on frequent synchronization among workers, which introduces delays and reduces overall efficiency in large-scale systems.

In this paper, we propose an enhanced APPO framework that addresses the aforementioned challenges. Our framework optimizes gradient synchronization and worker communication, enabling faster convergence and improved sample efficiency. Unlike traditional approaches, we manually calculate gradients at the worker level, reducing the need for frequent synchronization and alleviating bottlenecks.

To evaluate the effectiveness of our framework, we conduct extensive experiments in various challenging environments. The results demonstrate that our optimized APPO outperforms traditional frameworks, achieving superior training stability, efficiency, and reduced variance across multiple runs.

The key contributions of this paper are:

1) A novel enhanced APPO framework with optimized gradient updates and asynchronous communication strategies.
2) A comprehensive evaluation of the proposed framework, demonstrating enhanced training stability and efficiency.

3) Insights into the advantages of asynchronous updates in distributed RL systems, paving the way for more scalable solutions.

This work advances the field of distributed RL by providing a more scalable, robust, and efficient solution for training in complex and dynamic environments.

II. Preliminaries

In this section, we describe the PPO algorithm [3], which serves as the foundation for the proposed framework, and provide an overview of its core components and principles. PPO is one of the most widely used RL algorithms due to its simplicity, robustness, and effectiveness in both single-agent and distributed settings.

A. Actor-Critic PPO Algorithm

The PPO algorithm is an on-policy RL algorithm that iteratively improves a stochastic policy. As a member of the **policy gradient frameworks** class, PPO directly optimizes the policy by adjusting the probabilities of actions to maximize cumulative rewards. Unlike traditional policy gradient frameworks, PPO introduces mechanisms to constrain updates, improving stability and preventing drastic changes to the policy. This characteristic makes it a popular choice for training robust policies in various RL applications [4]. PPO builds on previous work, notably Trust Region Policy Optimization (TRPO), which also constrains policy updates but uses a more computationally expensive constraint based on the natural gradient [5]. The improvements in PPO allow for more efficient learning and scalability compared to TRPO.

PPO is implemented within the **actor-critic framework**, where learning is facilitated by two separate neural networks:

- **Actor network**: Responsible for selecting actions based on the policy $\pi_\theta(a_t|s_t)$ parameterized by θ, where s_t and a_t represent the state and the selected action, respectively, at time step t.
- **Critic network**: The value of the state is evaluated using the value function $V_\omega(s_t)$ parameterized by ω, which guides policy updates by estimating the advantage of each action.

This architecture enables the separation of action selection and value estimation, promoting more stable learning by reducing the variance of gradient estimates through the advantage function.

The interaction process involves workers using the Actor network to sample actions, observing the resulting state transitions and rewards. These interactions are stored as trajectories, which are later used to compute policy updates. The Critic network evaluates the state values, providing feedback to refine the Actor's policy. This feedback loop ensures that the policy adapts to maximize cumulative rewards over time.

B. Mathematical Foundation of PPO

The PPO algorithm introduces several key innovations to stabilize training and improve efficiency:

Policy Gradient Objective: The primary objective in PPO is to maximize the expected cumulative rewards by optimizing the policy π_θ. The traditional policy gradient objective J^P is defined as:

$$J^P(\theta) = \widehat{E}\left[\log \pi_\theta\left(a_t|s_t\right)\widehat{A}_t\right], \tag{1}$$

where $\widehat{A}_t$ is the advantage, representing the relative value of an action compared to the baseline. This advantage is typically computed using the Generalized Advantage Estimator (GAE) [6]:

$$\widehat{A}_t = \delta_t + (\gamma\lambda)\,\delta_{t+1} + \cdots + (\gamma\lambda)^{U-t+1}\,\delta_{U-1}. \tag{2}$$

where:

- $\gamma \in [0,1]$ is the discount factor that balances immediate and future rewards.
- $\lambda \in [0,1]$ is the GAE decay parameter, balancing bias and variance in advantage estimation.
- δ_t is the Temporal-Difference (TD) residual at time step t, defined as:

$$\delta_t = R(s_t) + \gamma V_\omega(s_{t+1}) - V_\omega(s_t),$$

  where $R(s)$ is the reward function defined for any state s.
- U is the horizon or the last time step in the trajectory being considered.

1) Value Function Objective: The critic network is trained to minimize the error between the predicted state value and the target state value, each obtained from $V_\omega(s_t)$ and $V_{\bar{\omega}}^{target}(s_t)$, respectively. Formally, this objective $L^V(\omega)$ is defined as:

$$L^V(\omega) = \widehat{E}\left[\left(V_{\bar{\omega}}^{target}(s_t) - V_\omega(s_t)\right)^2\right], \tag{3}$$

where the target critic network, $V_{\bar{\omega}}^{target}(s_t)$ with frozen parameters $\bar{\omega}$, is:

$$V_{\bar{\omega}}^{target}(s_t) = R(s_t) + \gamma V_{\bar{\omega}}(s_{t+1}). \tag{4}$$

Clipped Surrogate Objective: One of the key contributions of PPO is the use of the clipped surrogate objective to constrain policy updates. The clipped surrogate objective $J^{CLIP}(\theta)$ is defined as follows:

$$J^{CLIP}(\theta) = \widehat{E}\left[min\big(r_t\left(\theta\right), clip\left(r_t\left(\theta\right), 1-\epsilon, 1+\epsilon\right)\big)\widehat{A}_t\right], \tag{5}$$

where $r_t(\theta)$ is the ratio of new policy π_θ and old policy $\pi_{\theta_{\text{old}}}$ probabilities:

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}. \tag{6}$$

In the ratio $r_t(\theta)$, $\pi_{\theta_{\text{old}}}$ denotes the policy that generated the trajectories before the current update step. The clipping function ensures that policy updates stay within a predefined range, preventing overly large steps that could destabilize learning.

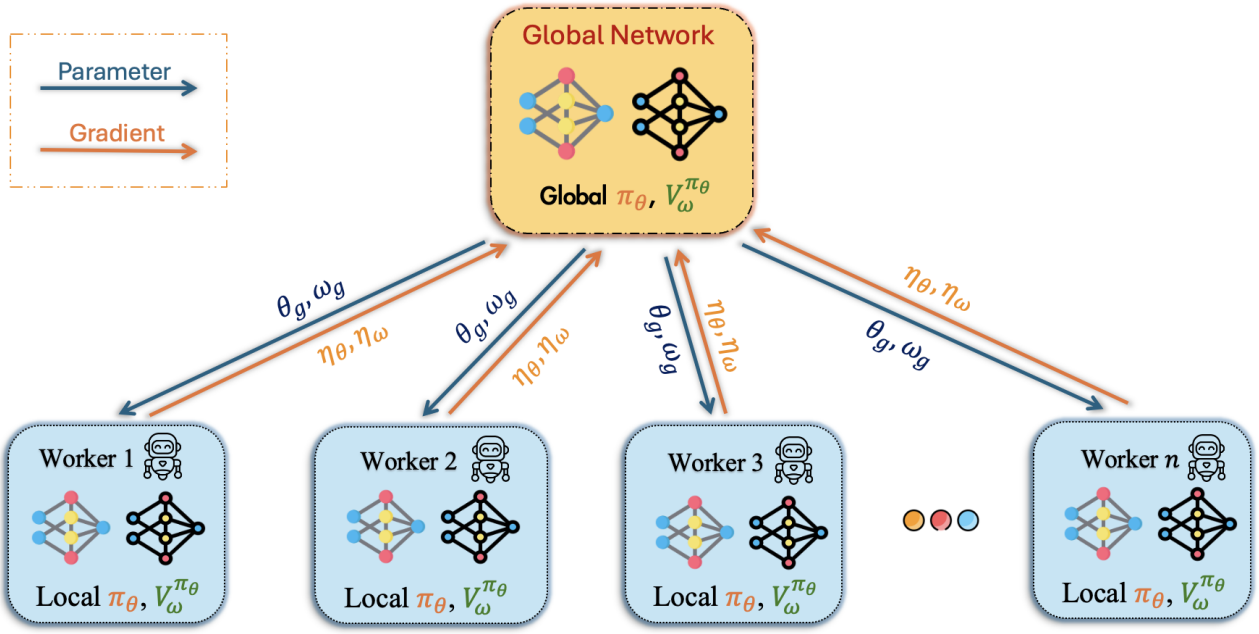


Fig. 1: Asynchronous PPO Framework: Each worker independently interacts with its environment and updates the global policy asynchronously.

C. Advantages of PPO

The simplicity, computational efficiency, and robustness of PPO make it ideal for large-scale distributed training. By combining the advantages of actor-critic architectures with stable policy updates, PPO is widely applicable across various domains, from robotics to multi-agent systems. The enhanced scalability of PPO provides the foundation for further improvements, such as the APPO algorithm proposed in this work.

III. Proposed Framework

We provide a detailed explanation of our proposed APPO framework in this section. In the proposed APPO:

- Workers send gradients η_θ, η_ω to the global network asynchronously, and the global parameters θ_g, ω_g are updated periodically.
- Updates are propagated back to workers, ensuring diverse policy exploration and avoiding delays caused by synchronization.

Our APPO leverages asynchronous updates to improve scalability and reduce communication overhead. Each worker optimizes its local policy using J^{CLIP} and L^V, and the global network aggregates their contributions.

The distributed asynchronous approach enhances learning efficiency, enabling robust training across multiple environments. It is well-suited for large-scale RL applications requiring decentralized execution and high throughput.

A. The Architecture of the Proposed APPO Framework

The proposed APPO framework, a distributed RL system illustrated in Figure 1, addresses the challenges of scalability and efficiency by employing a decentralized, asynchronous optimization paradigm. This framework consists of a centralized Global Network and multiple distributed Worker Nodes. Each Worker Node interacts with its respective environment, collects trajectories, and computes gradient, η_θ for the policy and η_ω for the value function, based on updates of local parameters θ_l and ω_l. These gradients are then asynchronously aggregated by the Global Network to periodically update the global parameters, θ_g and ω_g. By enabling Worker Nodes to operate independently, the APPO framework ensures efficient utilization of computational resources and scalability, overcoming the bottlenecks often faced by traditional synchronous federated systems that rely on strict synchronization. We provide a detailed explanation of the proposed APPO update process in the following sections.

1) Worker Nodes Training Process: Each Worker Node operates independently, as detailed in Algorithm 2. The training process begins by synchronizing the local parameters with the global network:

$$\theta_l \leftarrow \theta_g, \quad \omega_l \leftarrow \omega_g. \tag{7}$$

Worker Nodes interact with their environments to collect trajectories, storing N transitions in a local buffer D_N. For each trajectory, the following computations are performed:

- The advantage is estimated using $A_{\omega_l}^{\pi_{\theta_{\text{old}}}}(s_t, a_t)$.
- The target state value is estimated using $V_{\text{target}}^{\pi_{\theta_{\text{old}}}}(s_t, a_t)$.

It is important to note that $A_{\omega_l}^{\pi_{\theta_{\text{old}}}}(s_t, a_t)$ and $V_{\text{target}}^{\pi_{\theta_{\text{old}}}}(s_t)$ are derived using trajectories that were generated by $\pi_{\theta_{\text{old}}}$ prior to the current update step.

Subsequently, the local policy and value function are updated over E epochs. The policy is optimized using the clipped

surrogate objective $J_t^{\text{CLIP}}(\theta_l)$ to prevent excessive updates as shown in Eq. **(5)**. The value function is updated to minimize the loss $L_t(\omega_l)$, thereby improving the accuracy of state value predictions:

$$L_t(\omega_l) = \left(V_{\omega_l}^{\pi_\theta}(s_t) - V_{\text{target}}^{\pi_{\theta_{\text{old}}}}(s_t)\right)^2. \tag{8}$$

The $L_t(\omega_l)$ values computed for each transition are aggregated to L^V , which is subsequently used for the update:

$$L^V = \sum_{t=0}^{N-1} L_t(\omega_l). \tag{9}$$

After completing the local updates, the scaled gradients η_θ and η_ω are computed and sent asynchronously to the global network. The update process for θ_l and ω_l during local training can be detailed as:

$$\begin{aligned}
&\theta_{\text{init}} \leftarrow \theta_l^0, \\
&\theta_l^1 = \theta_l^0 + \alpha_l \nabla J_0^{\text{CLIP}}, \\
&\theta_l^2 = \theta_l^1 + \alpha_l \nabla J_1^{\text{CLIP}} = \theta_l^0 + \alpha_l (\nabla J_0^{\text{CLIP}} + \nabla J_1^{\text{CLIP}}), \\
&\quad \vdots \\
&\theta_l^n = \theta_l^0 + \alpha_l \sum_{t=0}^{n-1} \nabla J_t^{\text{CLIP}}.
\end{aligned} \tag{10}$$

Similarly, for the value function:

$$\begin{aligned}
&\omega_{\text{init}} \leftarrow \omega_l^0, \\
&\omega_l^1 = \omega_l^0 - \alpha_l \nabla L_0, \\
&\omega_l^2 = \omega_l^1 - \alpha_l \nabla L_1 = \omega_l^0 - \alpha_l (\nabla L_0 + \nabla L_1), \\
&\quad \vdots \\
&\omega_l^n = \omega_l^0 - \alpha_l \sum_{t=0}^{n-1} \nabla L_t.
\end{aligned} \tag{11}$$

Unlike traditional synchronous PPO frameworks, the APPO framework introduces a key novelty: global parameter updates occur less frequently, and gradients are computed manually at the worker level. Specifically, after performing n local updates, each worker calculates the aggregate gradients η_θ and η_ω using the following formulas:

$$\eta_\theta = \frac{\theta_l^n - \theta_{\text{init}}}{\alpha_l}, \quad \eta_\omega = \frac{\omega_{\text{init}} - \omega_l^n}{\alpha_l}, \tag{12}$$

where α_l is the learning rate, and θ_{init} and ω_{init} represent the initial local parameters before updates. These gradients, normalized by the local learning rate, ensure that the contributions to the global network are proportional to the local progress made by the worker nodes.

This manual computation of gradients aligns with the theoretical foundations of asynchronous distributed optimization, ensuring correctness and stability during the training process. As described in Algorithm 1, the global network updates the shared parameters θ_g and ω_g using a predefined global learning rate α_g:

$$\theta_g \leftarrow \theta_g + \alpha_g \eta_\theta, \quad \omega_g \leftarrow \omega_g - \alpha_g \eta_\omega. \tag{13}$$

Algorithm 1 Master Training Pseudocode

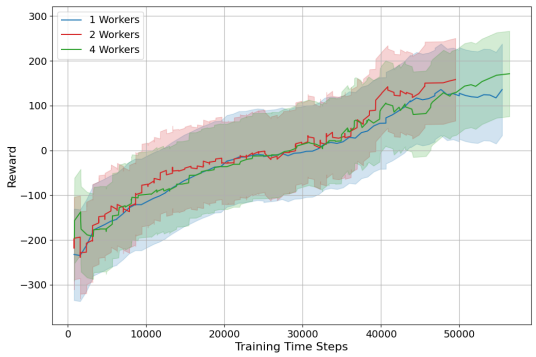
ω_g: global value function parameters
θ_g: global policy parameters
η_ω: the scaled difference between ω_{init} and ω_l
η_θ: the scaled difference between θ_{init} and θ_l
α_g: global learning rate
Initialize the network parameters: ω_g, θ_g
for $k = 0, 1, 2, \ldots$ **do**
    wait and get a gradient η_ω,η_θ from a worker
    $\omega_g \leftarrow \omega_g - \alpha_g \eta_\omega$
    $\theta_g \leftarrow \theta_g + \alpha_g \eta\theta$

Algorithm 2 Worker Training Pseudocode

ω_l: local value function parameters
θ_l: local policy parameters
ω_{init}: initial local value function parameters
θ_{init}: initial local policy parameters
$\pi_{\theta_{\text{old}}}$: policy under previous parameters θ_{old}
D_N: buffer holding N sampled transitions
α_l: local learning rate
Initialize the local network parameters: ω_l, θ_l
for $k = 0, 1, 2, \ldots$ **do**
    assign global parameters to local: $\omega_l \leftarrow \omega_g, \theta_l \leftarrow \theta_g$
    $\pi_{\theta_{\text{old}}} \leftarrow \theta_l$
    using $\pi_{\theta_{\text{old}}}$, sample N step transitions into D_N
    for each transition $(a_t, s_t, R(s_t), s_{t+1})$ in D_N **do**
        $A_{\omega_l}^{\pi_{\theta_{\text{old}}}}(s_t, a_t) = R(s_t) + \gamma V_{\omega_l}^{\pi_{\theta_{\text{old}}}}(s_{t+1}) - V_{\omega_l}^{\pi_{\theta_{\text{old}}}}(s_t)$
        $V_{\text{target}}^{\pi_{\theta_{\text{old}}}}(s_t) = R(s_t) + V_{\omega_l}^{\pi_{\theta_{\text{old}}}}(s_{t+1})$
    $\theta_{\text{init}} \leftarrow \theta_l, \omega_{\text{init}} \leftarrow \omega_l$
    for $h = 0, 1, 2, \ldots E-1$ **do**
        for each transition $(a_t, s_{t,t+1}, s_{t+1})$ in D_N **do**
            $L_t(\omega_l) \leftarrow V_{\text{target}}^{\pi_{\theta_{\text{old}}}}(s_t) - V_{\omega_l}^{\pi_{\theta_l}}(s_t)$
            $r_t(\theta_l) = \frac{\pi_{\theta_l}(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$
            compute $J_t^{\text{CLIP}}(\theta_l)$ by **Eq. (5)**
        $\omega_l \leftarrow \omega_l - \alpha_l \nabla_{\omega_l} \sum_{t=0}^{N-1} L_t(\omega_l)$ by **Eq. (11)**
        $\theta_l \leftarrow \theta_l + \alpha_l \nabla_{\theta_l} \sum_{t=0}^{N-1} J_t^{\text{CLIP}}(\theta_l)$ by **Eq. (10)**
    $\eta_\omega \leftarrow (\omega_{\text{init}} - \omega_l)/\alpha_l, \eta_\theta \leftarrow (\theta_l - \theta_{\text{init}})/\alpha_l$ by **Eq. (12)**
    send η_ω, η_θ to master

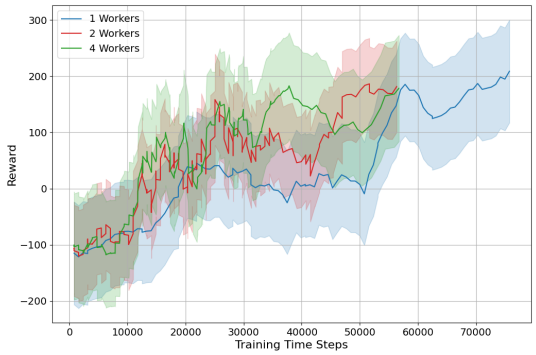
The proposed APPO framework offers several advantages over traditional PPO and synchronous distributed systems:

1) **Scalability:** The asynchronous architecture eliminates the need for global synchronization, enabling efficient scaling across numerous worker nodes.
2) **Reduced Latency:** Workers perform updates independently, ensuring that straggling nodes do not delay the training process.
3) **Improved Resource Utilization:** By distributing the computational workload across workers, the framework maximizes the utilization of available resources.
4) **Faster Convergence:** Frequent local updates and asynchronous gradient aggregation allow the system to adapt quickly to the dynamic training environment.



(a) LunarLanderContinuous-v3 Time Steps

(b) LunarLanderContinuous-v3 Process Time

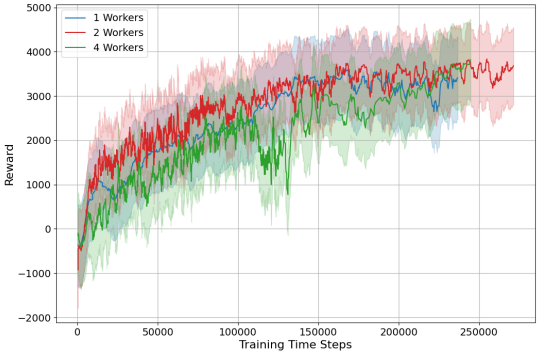


(c) BipedalWalker-v3 Time Steps

(d) BipedalWalker-v3 Process Time

(e) Hopper-v5 Time Steps

(f) Hopper-v5 Process Time



(g) HalfCheetah-v5 Time Steps

(h) HalfCheetah-v5 Process Time

Fig. 2: Comparison of rewards over training time steps and process time (in seconds) for different numbers of workers.

IV. Experiments

This paper proposes an enhanced APPO framework and evaluates its performance across various continuous control environments. The goal of these experiments is to demonstrate the scalability and stability of the APPO framework when using a different number of workers in distributed training.

A. Environments

For our experiments, we employed multiple OpenAI Gym [7] environments, several of which utilize the MuJoCo physics engine [8]. Specifically, we considered the following environments:

- **LunarLanderContinuous-v3:** A continuous control task where the goal is to land a spacecraft safely on a platform by controlling the thrust of the main and side engines.
- **BipedalWalker-v3:** A task where a two-legged robot must learn to walk across uneven terrain.
- **Hopper-v5:** A one-legged robot must learn to hop forward efficiently while maintaining balance.
- **HalfCheetah-v5:** A two-dimensional cheetah-like robot must maximize forward velocity by learning efficient limb coordination.

Each of these environments presents unique challenges in terms of control complexity and reward dynamics, providing a robust benchmark for evaluating the performance of RL algorithms.

B. Experimental Setup

To evaluate the effectiveness of the proposed APPO framework, we conducted experiments with 1, 2, and 4 workers. For each configuration, training was performed 5 times, and the average results were plotted to assess performance and stability. All experiments were conducted using CPUs to ensure the scalability of the approach to low-resource environments.

The experiments aimed to analyze:

1) **Training Time Efficiency:** How quickly the algorithm achieves a stable reward.
2) **Performance:** The maximum reward obtained during training.
3) **Stability:** The variance in reward across runs.

C. Experimental Results and Analysis

Figure 2 presents the mean performance values and standard deviations across 5 runs for four environments: LunarLanderContinuous-v3, BipedalWalker-v3, Hopper-v5, and HalfCheetah-v5, with 1, 2, and 4 workers used in the experiments.

In all tested environments, the use of 2 and 4 workers consistently results in improved training performance compared to the 1-worker setup. The configurations with more workers tend to converge faster and achieve higher rewards. Moreover, the standard deviation of rewards is generally smaller with 2 and 4 workers, indicating more stable and reliable training. These findings suggest that the APPO framework weffectively scales with the number of workers, improving both training speed and stability across multiple environments.

Further experiments with Hopper-v5 and HalfCheetah-v5 will provide additional insights into the scalability of APPO across more challenging environments.

V. Conclusion

In this paper, we introduced a new APPO framework, which combines asynchronous optimization and multi-worker setups to enhance the scalability, stability, and efficiency of RL. Through experiments on environments such as LunarLanderContinuous-v3, BipedalWalker-v3, Hopper-v5, and HalfCheetah-v5, we showed that increasing the number of workers from 1 to 2 and 4 results in improved training speed and performance. Furthermore, the manual gradient update optimization in APPO contributes to more efficient convergence, as it allows each worker to independently compute gradients and reduce synchronization bottlenecks. The results demonstrate that APPO accelerates convergence while maintaining consistent reward performance. These findings highlight the potential of APPO as a flexible and efficient framework for large-scale distributed RL tasks. Future work could further optimize the gradient update process and explore its application in more complex environments.

Acknowledgment

This work was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (No. NRF-2018R1A6A1A03025526) and the National Science and Technology Council of Taiwan (NSTC) (No. NSTC113-2222-E-155-007).

References

[1] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Harley, T. Lillicrap, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," *International Conference on Machine Learning (ICML)*, pp. 1928–1937, 2016.

[2] H.-K. Lim, J.-B. Kim, I. Ullah, J.-S. Heo, and Y.-H. Han, "Federated reinforcement learning acceleration method for precise control of multiple devices," *IEEE Access*, vol. 9, pp. 76 296–76 309, May 2021.

[3] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[4] D. Quang Tran and S.-H. Bae, "Proximal policy optimization through a deep reinforcement learning framework for multiple autonomous vehicles at a non-signalized intersection," *Applied Sciences*, vol. 10, no. 16, 2020.

[5] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Trust region policy optimization," *arXiv preprint arXiv:1502.05477*, 2015.

[6] J. Schulman, P. Moritz, S. Levine, M. I. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," *arXiv*, vol. 1506.02438, 2015, [online] Available: https://arxiv.org/abs/1506.02438.

[7] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, W. Tang, and W. Zaremba, "OpenAI Gym," *arXiv preprint arXiv:1606.01540*, 2016.

[8] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033, 2012.