Towards Distributed Communication and Control in Real-World Multi-Agent Reinforcement Learning

Jieyan Liu, Yi Liu, Zhekai Du, Ke Lu
University of Electronic Science and Technology of China
liujy@uestc.edu.cn, liuyi61612021@163.com, dzk1996411@163.com, kel@uestc.edu.cn

ICC 2022 - IEEE International Conference on Communications | 978-1-5386-8347-7/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICC45855.2022.9838553

***Abstract*—Multi-agent system investigates the problem of designing a complex system composed of multiple autonomous agents with limited ability and partial observability. As a milestone, AlphaStar has achieved remarkable success in StarCraft II, which is a significant breakthrough in the competitive environments with complex strategic spaces and real-time decisions. However, it poses new challenges for deploying these centralized control models in real-world environments because many of them in such competitive environments were not designed to accommodate the requirements of real-world communication networks, e.g., the problems of high latency and large traffic are inevitable when they are actually deployed. To alleviate this issue, we propose a distributed control paradigm that explicitly splits the control power between the centralized meta-agent and agent units through a combination of centralized and decentralized paradigms. The units can autonomously decide to follow the decisions of the meta-agent or adapt to environment variations immediately by themselves in a decentralized manner. We simulate real-world network environments based on the Mininet platform, experiments based on the StarCraft II Learning Environment (SC2LE) show that our approach achieves a better adaptation in real-world network environments.**

I. Introduction

The last decades have witnessed a resurgence of artificial intelligence which owes to the significant breakthrough of deep learning (DL) [1] techniques. DL enables machines to extract low-level perception information behind massive training data. Given its effectiveness, deep reinforcement learning (DRL) [2] combines the perceptual ability of DL with the decision-making ability of reinforcement learning, which provides a human-level control in the perceptual decision-making problems of complex systems [2]. However, unlike these games with only a single agent and environments where the states are countable and fully observable, many real-world applications require agents to compete or coordinate with each other under partial observability to achieve a certain goal, e.g., robotic teams, distributed control, collaborative and competitive decision systems, etc. Thus there is a strong incentive to extend DRL methods to multi-agent system (MAS) [3]. Conventional DRL methods fail to handle these environments due to partial observability and non-stationary environments caused by dynamic strategies of other agents [4].

To take a further step towards MAS, real-time strategy (RTS) games like StarCraft II have become an important challenge and a strong benchmark for researching MAS due to its imperfect information, combinatorial action space, real-time decision and multi-agent setting [5]–[7]. As a milestone, DeepMind's agent AlphaStar [8] has reached the grandmaster level in StarCraft II, being the first AI to reach the top esports league without simplification of the game. AlphaStar models the MAS as a set of diverse meta-agent players who control all their combat units in a centralized way. That means all combat units should input their observation data into the central model and get actions from it. Whereas, many real-world problems are not suitable to be solved in this centralized paradigm due to scalability and communication problems. Specifically, during training and inference, the observation data and action instructions between the model and the environment are transmitted through the memory copying instead of the real-world communication network, which overwhelmingly neglects the latency caused by the bandwidth limitation of networks in real-world environments (shown in the left hand of Fig. 1), thus resulting in the time shift between environment states and corresponding action instructions. It is just like a player who plays an online game with high latency, so that these centralized control models trained and tested well on a single machine may perform poorly when they are deployed in a real-world environment with a limited-bandwidth channel.

Independent RL [9] avoids the latency problem of centralized control since every unit has its local intelligence, and no data should be transmitted to the back-end model. While learning in this fully decentralized paradigm is challenging because the environment becomes non-stationary from the perspective of each agent [4]. Therefore, the current state of deploying multi-agent deep reinforcement learning (MDRL) system on real-world networks leaves an unsatisfactory choice, i.e., either (1) use a centralized approach that performs well but has constrained scalability and high latency issues; or (2) use a decentralized learning model that has low latency and better scalability but suffers from the non-stationarity issue. To tackle this problem, we propose a novel distributed control paradigm which takes the latency of real networks into consideration and takes the best of both worlds — simultaneously preserves the performance of centralized manners and the low latency of decentralized ones. We motivate our framework based on the observation that not all the decisions are equally hard to be made, some of them can be easily made by some shallow layers instead of going through the whole deep neural network (DNN) model. Without loss of generality, we divided the whole DNN hierarchically into two parts (can be extended to more parts), i.e., the local model on each agent as the local action guidance, and the centralized model that deployed in the back-end command center, which combines the observation of multiple agents and make a more effective and global action (shown in the right hand of Fig 1). The backbone of the DNN links the local and central models. Based on this, we design a schema based on the information entropy to make agents adaptively choose the policy in an effective way. Since the network environments are usually rigorous in real-world competitive environments, our approach can greatly reduce the average time required to perform an action. Besides, the decentralized agents increase the robustness and scalability of the system. In a nutshell, the main contributions of this work can be summarized as follows: (1) We propose a distributed control approach for MDRL domain by a distributed DNN architecture, which takes the network latency into consideration when it is deployed in the real-world environments. Experimental results show that our proposed approach can be more adaptable in real-world environments. (2) We construct a latency-aware MAS simulation platform for MDRL. Specifically, we integrate the Mininet simulation platform into StarCraft II Learning Environment (SC2LE) for deploying decentralized agents on different virtual network nodes and simulate communication delay between them. The interfaces of SC2LE are reconstructed to be compatible with decentralized models. (3) Our method reduces the non-stationarity issue of the environment to a certain extent. The centralized model enables agents to share information, so as to learn to cooperate implicitly. Experimental results show that the system performs better than both fully centralized or decentralized architectures.

II. Related Work

Single agent is not capable for complex scenarios due to limited ability. Recently, many methods in RL and MAS were scaled to MDRL. One earliest work of MDRL is independent Q-Learning [10], where each agent learns its own policy simultaneously. This individual policy maps the agent's own observation to its own action. Tampuu *et al.* [11] used independent learning to investigate how multiple agents learn and make different policies in both collaborative and competitive tasks. However, directly leveraging these single-agent DRL methods in multi-agent environments suffers from the non-stationarity problem due to the dynamically changing policies during the training procedure [4]. Recently, some researchers proposed to reduce the independence and stabilize the training by learning to communicate among agents [12]–[15]. For instance, Foerster *et al.* [12] propose the DDRQN to share hidden layers among agents and learn to communicate to solve riddles. Later, they proposed the Reinforced Inter-Agent Learning (RIAL) and Differentiable Inter-Agent Learning (DIAL) [13] to explicitly learn communication protocols among multiple agents to share information so as to alleviate the non-stationarity problem. For a similar purpose, Sukhbaatar *et al.* proposed CommNet [14], which learns to communicate in hidden layers by sharing the extracted features among agents. There are also some methods to deal with the experience replay buffer in MDRL by adding extra information to the experience tuple that disambiguates the age of the sampled data (e.g., fingerprint [16]), or leveraging importance sampling strategy [16] to reweight previous experiences when the environment changes. Besides, some methods adopt meta-learning [17] techniques to quickly adapt to changing environment dynamics via Model-Agnostic Meta-Learning (MAML) [18]. Although these methods have achieved some promising results, the inherent non-stationarity problem cannot be well solved in this fully decentralized paradigm, resulting in inferior performance compared with centralizated architectures.

Some notable works have been conducted that combines centralized and decentralized architectures by *centralized learning but decentralized execution* [19]–[21]. These methods follow the Actor-Critic paradigm [22] and explicitly use a centralized critic that accesses and evaluates the global state during training, thus stabilizing the training. However, the agents still perform in a fully decentralized fashion during execution, the reliability of the system cannot be guaranteed. Different from the above methods, our method can be regarded as a new way to reduce the non-stationary of the environment by a distributed control paradigm. In addition, all of the above methods are either fully centralized or decentralized during execution, while our method organically combines two paradigms in both training and execution stages, thus combining their advantages more effectively. Furthermore, our method is more suitable for multi-agent systems in real-world scenarios where the network bandwidth is limited.

III. Preliminaries

We begin by briefly revisiting the classic Deep Q-Network [2], which is used as our basic formulation component. DQN uses a neural network to approximate the state-action value function $Q(s,a) = \mathbb{E}\left[R_t|s_t=s,a\right]$, where $R_\iota = \sum_{k=0}^{\infty}\gamma^k r_{t+k}$ and r_t is the immediate reward received at time t. Compared with Q-learning, it makes two innovations: maintains an experience buffer $\mathcal{D}_t = \{e_1,\ldots,e_t\}$ to store interactive information to break the strong correlation between training data and keeps two version parameters of the neural networks: the target network and the evaluation network to alleviate the non-stationary data distribution.

The experience $e_t = (s_t, a_t, r_t, s_{t+1})$ consists of the state s_t at time t, the aciton a_t adopted by the agent, the immediate return r_t it receives and the next state s_{t+1}. In the partially observable environment, the state s_t is replaced by observations o_t. DQN uses a temporal-difference (TD) [2] to update at each iteration i with loss function:

$$L_t\left(\theta\right) = \mathbb{E}(\underbrace{r_t + \gamma \max_{a_{t+1}} Q\left(s_{t+1}, a_{t+1}; \theta^-\right)}_{\text{Target Q value}} - Q\left(s, a; \theta\right))^2 \quad (1)$$

where θ and θ^- are the parameters of the evaluation network and the target network at that moment, respectively. They synchronize every certain steps.

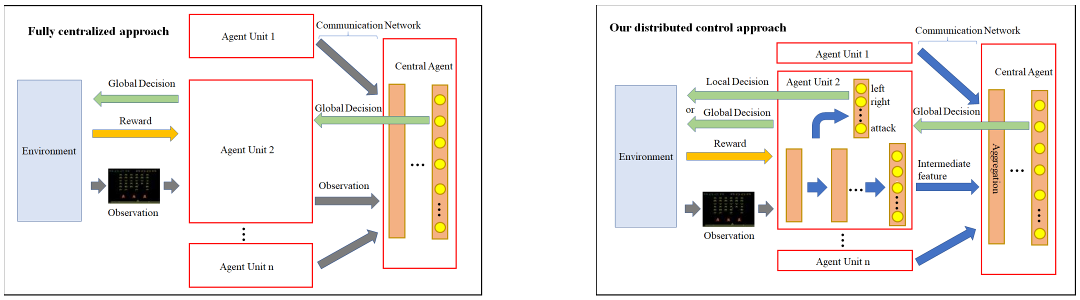


Fig. 1. (a) Centralized architecture. (b) The main idea of our work. The control of agent units is divided into two parts; Local units make a preliminary judgment based on local information without any communication latency, and the central agent makes a global command by integrating the observation of all units. The flows in gray denote the raw observation data without processing. Blue flows mean the intermediate calculation data generated by neural computing. Yellow and Green flows represent rewards and decision results, respectively.

IV. Proposed Distributed Control Architecture

A. *Overview of Distributed Control in MDRL*

Existing multi-agent algorithms either use a fully centralized or decentralized model to control agents during execution. The former will introduce network latency in practice, while the latter limits the agents to only observe locally, thereby the performance is inferior. Fig. 1 provides an overview of the comparison between our distributed control architecture and the fully centralized architecture. In real-world scenes, the agent units are usually exposed to the environment and interact with the environment directly, i.e., obtain the observation of the environment through the sensor devices and make actions to act on the environment. In the centralized architecture, the agent units have no local intelligence, namely they need to transmit the observations to the central model in the back-end, waiting for the commands from the central model. The transmission of corresponding information through real-world communication networks will introduce a non-trivial latency.

In our method, agent units have their local models. After making a local decision, each agent adaptively chooses to use the local decision or transmit the extracted features to the back-end for aggregation and further inference. Intuitively, when it is confident about what to do, it should take the action output by itself immediately, while if the agent is confused, it should query the back-end central model and wait for a globally optimal decision. On the one hand, our method reduces the overall time to ask a centralized model by a distributed DNN architecture. On the other hand, the transmission amount of intermediate features is much lower than transmitting raw observations, making our model performs quickly and effectively.

Without loss of generality, we choose single-agent DQN and independent DQN algorithm as the centralized and decentralized components in our method. It can also be easily extended to scenarios with a large number of agents and various MDRL methods. In terms of the scalability of the model, we can learn from some techniques, such as parameter sharing [23], which factors the action space of centralized multi-agent systems from $|\mathcal{A}|^n$ to $n|\mathcal{A}|$.

B. *Local Model and Central Model*

We trained the local model of the agents through independent Q-learning, i.e., each agent independently learns through the standard DQN algorithm. Formally, the target Q value of the local model is

$$Q_t^{\text{Local}} = r_t + \gamma Q\left(s_{t+1}, \underset{a}{\operatorname{argmax}} Q\left(s_{t+1}, a; \theta^{-}\right); \theta^{-}\right). \quad (2)$$

For each iteration, a batch of samples will be sampled from the experience replay buffer, then the target Q value will be calculated and the model parameters θ will be updated through mean square error like Eq. (1).

Standard DQN uses the same value to select and evaluate an action, which leads to a high probability of selecting an overestimated value. To prevent this problem in the more challenging central model, we use the double DQN [24] in order to alleviate the over estimation of the Q value. Double DQN decouples the evaluation and selection. The target Q value of the central model can be expressed as

$$Q_t^{\text{Central}} = r_t + \gamma Q\left(s_{t+1}, \underset{a}{\operatorname{argmax}} Q\left(s_{t+1}, a; \theta\right); \theta^{-}\right), \quad (3)$$

in each time step the samples are randomly used to update the evaluation network. For each update, the evaluation network is used to determine the greedy strategy (select action), and the target network is used to determine the corresponding target Q value (evaluate action).

In addition, learning efficiency is low in DQN because experiences are sampled uniformly in memory. Thus we apply the prioritized replay [25] with both the local model and the central model. Prioritized DQN makes experiences that are new or with large TD error to be preferentially used for updating. We denote the TD error of samples i by δ_i, then the probability $P(i)$ of the sample being sampled is $P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$, where α is an adjustable hyper-parameter and $p_i = \frac{1}{\text{rank(i)}}$ with rank(i) the sort of δ_i. Since the distribution of the original samples is different from that of the priority samples, a importance sampling weight $\omega_i = (\frac{1}{N} \cdot \frac{1}{P(i)})^\beta$ is used with β another hyper-parameter. To update the parameters of the models, we employ a soft update strategy, which synchronizes the parameters between target and evaluation network by moving average instead of directly copying. Since the agents are homogeneous, we used the parameter sharing scheme [23] to learn the local model, which is effective in partially observable environments.

Algorithm 1 Inference of Distributed Control Algorithm

```
Require: Trained local model $f_{\text{local}}$ and central model $f_{\text{central}}$
1: $t \leftarrow 0$
2: while $t < h$ and $S_t \notin$ Termination states do
3:     for each agent $i$ do
4:         Get the output of local model $\mathbf{y} = f_{\text{local}}\ (o_t^i)$
5:         $\mathbf{z} = \text{softmax}(\mathbf{y})$
6:         Calculate the entropy $e = Entropy(\mathbf{z})$
7:         if $e \leq T$ then
8:             Excute a local decision $a_t = \underset{a}{\text{argmax}}\{\mathbf{y}\}$
9:         else
10:            Waiting for global decision $a_c = f_{\text{central}}\ (O_t; i)$
11:        end if
12:    end for
13:    $t = t + 1$
14: end while
```

C. Observation Aggregation

In the multi-agent environment, there exist multiple local models on agent units, while only one central model in the back-end. It is necessary to make an information aggregation to be compatible with MAS and have a more accurate understanding of the real environment state. In this paper, we sum up the intermediate tensors output by all local models as the observation input O_t to the central model.

D. Confidence Metric

During inference stage, the agent unit needs to autonomously determine whether it is necessary to query the central model for instructions. In this paper, we use the information entropy of the output of the local model to measure the confidence in what it should do. Formally, the information entropy is defined as $\eta(\mathbf{y}) =$ entropy $(\boldsymbol{y}) = -\sum_{c\in\mathcal{C}} y_c \log y_c$, where $\boldsymbol{y} = \text{softmax}(\boldsymbol{x}) = \frac{\exp(\boldsymbol{x})}{\sum_{c\in\mathcal{C}}\exp(x_c)}$, with $\boldsymbol{x}$ the output of local model and $\mathcal{C}$ represents the size of action space. In value-based approaches, $\boldsymbol{x}$ is the Q value vector of actions, while in policy-based approaches, it can be substituted by the probability distribution of actions.

We use a threshold T as a criterion to measure whether it is confident enough. Specifically, η close to 0 indicates that the agent is confident about the action, thus performing the action immediately without asking the central model. While large η (i.e., $\eta > T$) means it is confused, thus asking the central model. The whole inference process can be seen in Algorithm 1. The optimal threshold T can be obtained by multiple experiments.

V. Experiments

A. Experimental Setup

1) MDRL Environment Settings: SC2LE [26] is a StarCraft II AI research environment widely used. For our research purpose, we customize the StarCraft II map. Specifically, there are two teams in our game, each with a set of agents and a base building. Each agent only observes a subarea of the map centered on itself. In each time step, we can get the observation of each agent through the SC2LE interfaces. The observation is a matrix of the vision, which takes values in [0, 4], denoting [background, self, ally, neutral, enemy] units, respectively. Each agent must select an action from a restricted set, i.e., move [position], attack [id] and act on the environment, then the game moves on to the next time step. The maximum number of steps we set for an episode is set to 4000.

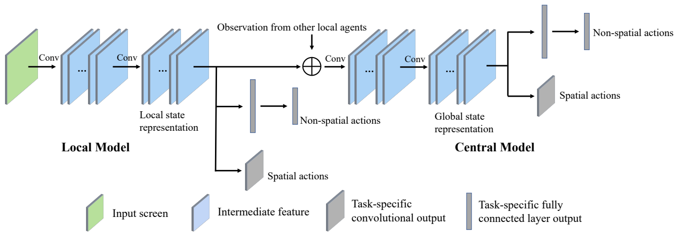
In this experiment, all agents are Terran Vikings, which only execute move and attack actions. Our experiments are based on 3 Vikings vs 3 Vikings scenario. One team is controlled by our algorithm, while the opponents are controlled by the game AI. All game AIs are able to move and attack, when they are destroyed, they will be regenerated in random positions. The game map is 48 × 48, and the field of vision of all units is 24 × 24. When our agents hurt or destroy the enemy units, they will get a positive reward, while when they are attacked or destroyed, they will get a negative reward.

2) Communication Network Settings: We conduct and analyze our experiments based on the Mininet [27] simulation platform. Mininet is a lightweight network virtualization platform, which is effective to create a custom network topology and run any user-specific programs on the network nodes. We deploy models trained by our distributed control framework on this platform.

We ran the StarCraft II game and the central model on two separate network nodes. Then another 3 nodes are used to simulate 3 local agent units. We set no delay between the game engine and local agent units, while set a certain latency between the agent units and the central model. We can change the latency arbitrarily, if desired, to observe the impact of the network latency.

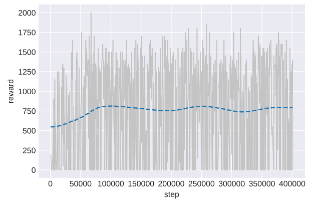
3) DNN Architecture and Training Settings: Fig. 2 describes the architecture we developed in this paper. In our DNN architecture, the Q values of spatial actions are directly output through a sequence of convolutional layers. The stride of the convolution layers we used is set to 1, thus the output dimension of the network is consistent with the observation feature of the input. The two convolution layers we use in this paper have 16, 32 filters of size 5 × 5, 3 × 3, respectively. Then the Q value of spatial actions is obtained using a 1 × 1 convolution layer of the state representation with a single output channel. The non-spatial action, i.e., attack action, is obtained using fully connected layers fed with state representations. The architecture of the local model is almost the same as the central model.

After learning, we deployed the models on the Mininet network nodes. We set the network latency between the agent units and the central model to 0ms, 5ms and 20ms respectively, then run the experiments for 5 minutes. The SGD optimizer with the learning rate 0.01 and a weight decay 0.99 was used. The batch size is 32. We synchronize the parameters of the target network with the evaluation network every 2000 iterations. For the soft update, we set the tau to 0.001. The hyper-parameters α and β are set to 1 in our experiments.

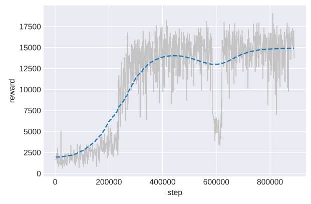


Observation from other local agents
Conv
Conv
Conv
Conv
Local state representation
Non-spatial actions
Spatial actions
Local Model
Global state representation
Central Model
Non-spatial actions
Spatial actions
Input screen
Intermediate feature
Task-specific convolutional output
Task-specific fully connected layer output

Fig. 2. DNN architecture developed in this paper.

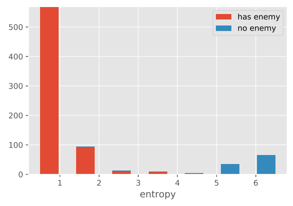


(a) Reward of local model

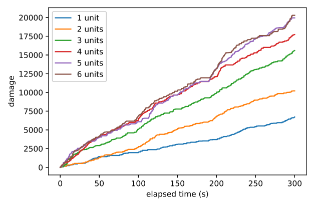


(b) Reward of central model

Fig. 3. The performance of the learning model we used in this paper. (a) and (b) both show the 3v3 scene. The x-axis represents the accumulated steps during the training stage and the y-axis represents the reward of each episode.



(a)



(b)

Fig. 4. (a) The distribution of entropy in one episode. The vertical coordinate represents the times that the corresponding entropy value appears in this experiment. The red color means that the agent unit has observed enemy units, while the blue represents that no enemy unit has been observed. (b) The performance of the MAS when the agent number changes from 1 to 6.

TABLE I

THE EFFECTS OF DIFFERENT THRESHOLD (T) SETTINGS.

T	Local decision rate (%)	Decision time (s)
0	0	29.2
1	49.2	21.1
2	70.7	16.5
3	86.7	13.4
4	87.9	13.1
5	92.9	12.6
6	93.8	11.7
7	100	9.9

B. Results

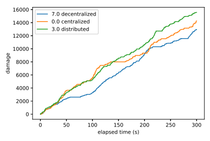
1) Training Results: Fig. 3 depicts the reward of each step during training. We can see that the reward curve fluctuates seriously. The reason may be that in reinforcement learning, especially in the environment with huge state-action space, the model will carry out a lot of exploration, and the reward is sparse in our environment, thus the learning process is relatively challenging. As a result, there will be obvious fluctuations in reward curves. For a clearer observation, we smooth it to get the average reward curve, as shown in blue. It can be seen from the curve in Fig. 3(a) that the mean reward of the local model only slightly improves during the training state. One reason is that in our experiments, learning between the local agents is completely independent, so the model is relatively more difficult to converge. Fig. 3(b) shows that the improvement of the central model is obvious, because the central model can get a more complete observation of the environment compared with the local model, and there is no non-stationary environment issue when using experience buffer, so it can quickly learn useful things.

2) Impact of Entropy Threshold: After training, we deployed the model on the Mininet nodes as described above. We set the network latency between the local model and the central model to 5ms, and run the experimental environment for 5 minutes. In order to have a more intuitive understanding of the confidence, i.e., entropy, we visualized the entropy distribution of one agent in one experiment, as shown in Fig. 4(a). Specifically, 787 steps were counted. Intuitively, when an agent observed enemy units, it is very clear about what to do, so the entropy is quite small, while if there is no enemy unit in its observation, it will be very confused, thus the entropy is large and it resorts to the central model.

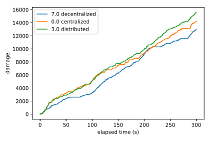
The impact of the threshold T is reported in Table I. The decision time includes the inference time of the DNN models and the transmission latency in communication network, both of them constitute the meaningless waiting time. We can see that when the threshold T increases, the ratio that the agent uses the local decision increases. When T is set to 0, the model becomes completely centralized, the agents will always ask the central model, thus introducing a lot of latency. When T is set to 7, the model becomes completely decentralized, all agents will use the local decisions. With T increasing, the central model will be asked less and the accumulated decision time will reduce. We empirically found that when the threshold T is set to 3, the performance of the system is optimal. Namely, it has a high local decision rate while the reward obtained by the system does not decline obviously. We set T=3 for following experiments unless noted otherwise.

3) Impact of Network Delay: To validate the influence of the network delay on the models, we set the network delay to 0ms, 5ms and 20ms, respectively. In each group of experiments, we test three different architectures (i.e., fully centralized, fully decentralized and distributed control architecture) by setting the threshold T to 0, 3, 7, respectively. In the execution phase, we use the accumulated damage caused by our units to measure the combat capability of the system. We run each group for 5 minutes. Fig. 5 depicts the accumulated damage caused by our team. When there is no network delay, the curves in Fig. 5(a) show that the decentralized architecture performs the worst, while the distributed control architecture is slightly better than the centralized control architecture. In Fig. 5(b) and Fig. 5(c), we can see that when the network delay increases gradually, there is almost no difference in the performance of the decentralized architecture, because it makes all decisions locally. The other two models have different degrees of performance degradation. The performance of the centralized architecture decreases faster than the distributed control one. When the network delay reaches 20 ms, it is almost the same as the decentralized architecture.

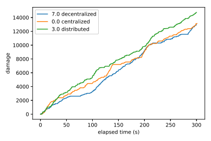
We also report the accumulated decision-making time of different architectures in Fig. 6. In all cases, the centralized



(a) with no latency

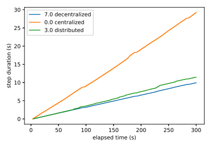


(b) with 5ms latency

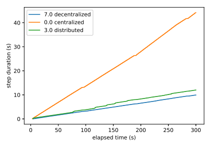


(c) with 20ms latency

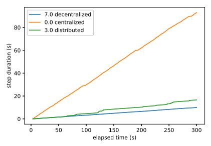
Fig. 5. The performance of the system under varying network latency settings.



(a) with no latency



(b) with 5ms latency



(c) with 20ms latency

Fig. 6. The accumulated decision making time under varying network latency.

architecture takes more time to make decisions than the other two paradigms since the centralized architecture needs more inputs and is more complex. The distributed control architecture takes a little longer time than the decentralized one. With the network delay increasing, the decision-making time increases rapidly while the decentralized architecture has no change. Although the time of distributed control architecture has increased, it is still similar to the fully decentralized architecture. It indicates that our distributed control architecture is more suitable for being deployed in real-world environments.

4) Impact of Scaling Agent Units: To validate the scalability of our method, we tested the performance of the system when the number of agents ranged from 1 to 7. Fig. 4(b) shows the experimental results. It can be seen that as the number of agents increases, the overall performance of the system grows up. When we have five agents, we can almost reach the bottleneck of the game reward. It is unnecessary to continue to add more agents. In theory, the number of agents can increase all the time if the computing power is sufficient.

VI. Conclusion

We propose a distributed control paradigm in MDRL, which combines the centralized and decentralized paradigms to control multiple agents flexibly. Based on the Mininet and SC2LE, we conducted many experiments in the StarCraft II game to validate our approach. The experimental results suggest that a properly trained centralized model and a decentralized model can achieve a better performance jointly with our framework. Moreover, when we put it into the real-world environment with network latency, our framework shows a better adaptability.

Acknowledgement

This work was supported by National Natural Science Foundation of China under Grant 62173066 and Grant 62073059. This work was supported in part by Joint Key Research and Development Project between Sichuan and Chongqing (No. cstc2020jscx- cylhX0004).

References

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[3] T. T. Nguyen, N. D. Nguyen, and S. Nahavandi, "Deep reinforcement learning for multiagent systems: A review of challenges, solutions, and applications," *TYCB*, vol. 50, no. 9, pp. 3826–3839, 2020.

[4] G. Papoudakis, F. Christianos, A. Rahman, and S. V. Albrecht, "Dealing with non-stationarity in multi-agent deep reinforcement learning," *arXiv preprint arXiv:1906.04737*, 2019.

[5] D. W. Aha, M. Molineaux, and M. Ponsen, "Learning to win: Case-based plan selection in a real-time strategy game," in *ICCBR*, 2005, pp. 5–20.

[6] S. Ontanón, G. Synnaeve, A. Uriarte, F. Richoux, D. Churchill, and M. Preuss, "A survey of real-time strategy game ai research and competition in starcraft," *T-CIAIG*, vol. 5, no. 4, pp. 293–311, 2013.

[7] M. Buro and T. M. Furtak, "Rts games and real-time ai research," in *BRIMS*, vol. 6370, 2004.

[8] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev *et al.*, "Grandmaster level in starcraft ii using multi-agent reinforcement learning," *Nature*, pp. 1–5, 2019.

[9] L. Matignon, G. J. Laurent, and N. Le Fort-Piat, "Independent reinforcement learners in cooperative markov games: a survey regarding coordination problems," *The Knowledge Engineering Review*, vol. 27, no. 1, pp. 1–31, 2012.

[10] M. Tan, "Multi-agent reinforcement learning: Independent vs. cooperative agents," in *ICML*, 1993, pp. 330–337.

[11] A. Tampuu, T. Matiisen, D. Kodelja, I. Kuzovkin, K. Korjus, J. Aru, J. Aru, and R. Vicente, "Multiagent cooperation and competition with deep reinforcement learning," *PloS one*, vol. 12, no. 4, p. e0172395, 2017.

[12] J. N. Foerster, Y. M. Assael, N. de Freitas, and S. Whiteson, "Learning to communicate to solve riddles with deep distributed recurrent q-networks," *arXiv preprint arXiv:1602.02672*, 2016.

[13] J. Foerster, I. A. Assael, N. de Freitas, and S. Whiteson, "Learning to communicate with deep multi-agent reinforcement learning," in *NIPS*, 2016, pp. 2137–2145.

[14] S. Sukhbaatar, A. Szlam, and R. Fergus, "Learning multiagent communication with backpropagation," in *NIPS*, 2016.

[15] A. Singh, T. Jain, and S. Sukhbaatar, "Learning when to communicate at scale in multiagent cooperative and competitive tasks," in *ICLR*, 2019.

[16] J. Foerster, N. Nardelli, G. Farquhar, T. Afouras, P. H. Torr, P. Kohli, and S. Whiteson, "Stabilising experience replay for deep multi-agent reinforcement learning," in *ICML*, 2017, pp. 1146–1155.

[17] M. Al-Shedivat, T. Bansal, Y. Burda, I. Sutskever, I. Mordatch, and P. Abbeel, "Continuous adaptation via meta-learning in nonstationary and competitive environments," *ICLR*, 2018.

[18] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *ICML*, 2017, pp. 1126–1135.

[19] J. N. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson, "Counterfactual multi-agent policy gradients," in *AAAI*, 2018.

[20] R. Lowe, Y. Wu, A. Tamar, J. Harb, O. P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," in *NIPS*, 2017, pp. 6379–6390.

[21] S. Li, Y. Wu, X. Cui, H. Dong, F. Fang, and S. Russell, "Robust multi-agent reinforcement learning via minimax deep deterministic policy gradient," in *AAAI*, 2019, pp. 4213–4220.

[22] S. Fujimoto, H. Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *ICML*, 2018, pp. 1587–1596.

[23] J. K. Gupta, M. Egorov, and M. Kochenderfer, "Cooperative multi-agent control using deep reinforcement learning," in *AAMAS*, 2017, pp. 66–83.

[24] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *AAAI*, 2016.

[25] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," in *ICLR*, 2016.

[26] O. Vinyals, T. Ewalds, S. Bartunov, P. Georgiev, A. S. Vezhnevets, M. Yeo, A. Makhzani, H. Küttler, J. Agapiou, J. Schrittwieser *et al.*, "Starcraft ii: A new challenge for reinforcement learning," *arXiv preprint arXiv:1708.04782*, 2017.

[27] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," in *SIGCOMM Workshop on Hot Topics in Networks*. ACM, 2010, p. 19.