Received 1 April 2024, accepted 14 May 2024, date of publication 20 May 2024, date of current version 28 May 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3402835

RESEARCH ARTICLE

An Efficient Distributed Reinforcement Learning Architecture for Long-Haul Communication Between Actors and Learner

SHIN MORISHIMA[1] **AND HIROKI MATSUTANI**[2], **(Member, IEEE)**
[1]Faculty of Engineering, Toyama Prefectural University, Toyama 939-0398, Japan
[2]Graduate School of Science and Technology, Keio University, Yokohama 223-8522, Japan

Corresponding author: Shin Morishima (morisima@pu-toyama.ac.jp)

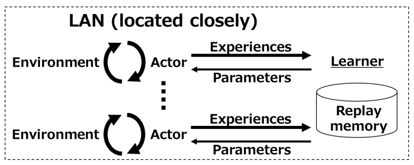
ABSTRACT A computing cluster that interconnects multiple compute nodes is used to accelerate distributed reinforcement learning that uses DQN (Deep Q-Network). In distributed reinforcement learning, actor nodes acquire experiences by interacting with a given environment and a learner node optimizes the DQN model. When distributed reinforcement learning is used in practical applications such as robotics, we can assume that actor nodes are located in edge side while the learner node is located in cloud side. In this case, the long-haul communication between them imposes significant communication overheads. However, most prior works simply assume that actors and learner are located closely, and do not take the overheads into account. In this paper, we focus on the practical environment where the actors and learner are located remotely, and they interact via a buffer node that collects information from multiple actor nodes. We implement a prototype system in which the buffer and learner nodes are connected via a 25GbE (Gigabit Ethernet) switch and a 10km optical fiber cable. Although a replay memory functionality is closely associated with the learner side, in this paper we propose to combine the replay memory into the buffer node. In our experiments using the prototype system, the proposed approach is compared with an existing approach in terms of the training efficiency (i.e., training loss) and the transfer efficiency over the long-haul communication (i.e., average priority of transferred experiences). As a result, the training loss of the proposed approach is reduced to 26% of the existing approach, and the average priority is 3.92 times higher than the existing approach after the training loss is converged. These results demonstrate that the proposed approach can improve the training/communication efficiency compared with the existing approach in a practical system that imposes long-haul communication between the actors and learner.

INDEX TERMS Distributed deep reinforcement learning, deep Q-network, prioritized experience replay.

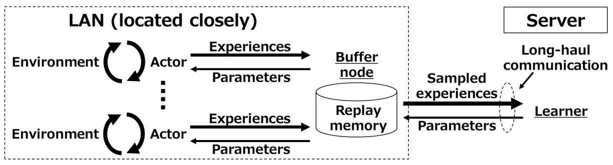
I. INTRODUCTION

Reinforcement learning is a machine learning approach to acquire an action policy that can maximize a long-term reward by repeating trial and error in action and observation at a given environment. Q-learning is a typical reinforcement learning method, where Q-value means effectiveness of an action in a given state. By taking an action based on Q-value and observing the environment, the Q-value is continuously updated in order to acquire an optimal action policy. DQN (Deep Q-Network) introduces a deep neural network called Q-network to approximate the conventional Q-learning, and it has been applied in various application domains, such as game AI [1], robotics [2], recommendation systems [3], autonomous driving [4], computer vision [5], and network control [6]. In this case, the reinforcement learning takes an action based on Q-network, observes the environment, and updates the Q-network by deep learning. Since these steps are repeated until the Q-network training is converged, it typically takes a time. In this paper, we focus on a typical case of distributed reinforcement learning, in which the first two steps (i.e., taking an action by Q-network and observing

The associate editor coordinating the review of this manuscript and approving it for publication was Fabrizio Messina.



(a) Environment assumed in prior works



(b) Environment assumed in this paper

FIGURE 1. Environment assumed in prior works (a) and that assumed in this paper (b).

the environment) are distributed over multiple edge nodes in order to accelerate acquisition of the optimal action policy.

In distributed reinforcement learning systems using DQN [7], [8], actor processes are in charge of the first two steps and a learner process is in charge of the last step (i.e., updating Q-network by deep learning). State transitions and rewards experienced by the actor processes are accumulated in an experience buffer, called experience replay memory, and the learner process samples a batch of experiences from the memory in order to train and update Q-network. Since data transfer between actor and learner nodes increases depending on the number of actor processes, experience size, and Q-network model size, their communication overhead is one of major performance bottlenecks in such distributed reinforcement learning systems.

In a practical assumption of the distributed reinforcement learning for the robot and IoT device control, actor nodes are located close to these target devices in the field, and a learner node is operated as a server in data centers. In this case, actor and learner nodes are located different places, and the communication overhead is significant due to an increased latency between the actors and learner. However, most of existing works did not consider such a practical situation and evaluated with an ideal environment where actor and learner nodes are located in the same place.

Figure 1 illustrates the environment assumed in prior works [7], [9], [10], [11] and that assumed in this paper. Figure 1 (a) corresponds to the existing work, where actors and learner are located in the same LAN (Local Area Network). A replay memory is closely associated with the learner. In contrast, Figure 1 (b) shows our proposed system that consists of actors, buffer, and learner. The actor and buffer nodes are located in the same LAN, while the learner node is operated at the cloud side. The buffer node receives experiences from the actors and sends them to the learner. In the latter environment, the communication overhead tends to be high, and this overhead reduces the training efficiency. To address this issue, in this paper we improve the transfer efficiency, which intends to transfer more experiences which are useful for the model training.

There are broadly two approaches to improve the transfer efficiency. The first approach is accelerating the communication by network optimization techniques. Our previous work adopts this approach using DPDK and F-Stack [12]. The second approach is sending useful experiences preferentially for the training. This paper adopts the second approach. We propose to combine a replay memory into the buffer node and locate it in the actor (or edge) side. Our approach can preferentially extract and transfer experiences that are useful for the model training by the learner; thus, it can improve the training efficiency even in environments where the amount of experiences which can be transferred is limited.[1]

Our contributions are summarized as follows.

- We implement a prototype system in which the buffer and learner nodes are connected via a 25GbE (Gigabit Ethernet) switch and a 10km optical fiber cable to evaluate distributed reinforcement learning in a practical assumption.
- We propose a new architecture that combines the replay memory with the buffer node and locate it in the actor (or edge) side to improve the transfer efficiency. In the proposed approach, only the experiences which are really used for the model training are transferred.
- We evaluate the proposed approach by comparing it with the baseline approach, which is assumed in most prior works. The evaluation results show that the proposed approach can improve both the transfer and training efficiency.

This paper is organized as follows. Section II overviews background knowledge on distributed deep reinforcement learning and prioritized experience replay and introduces related work on acceleration of distributed deep reinforcement learning. Section III illustrates the baseline approach, while Section IV proposes our proposed approach. Section V evaluates the proposed approach by comparing with the baseline approach, and Section VII concludes this paper.

II. RELATED WORK

A. DISTRIBUTED DEEP REINFORCEMENT LEARNING

High-performance distributed deep reinforcement learning systems have been widely studied recently. Distributed deep reinforcement learning approaches are classified into the model-based and model-free approaches. The model-based distributed reinforcement learning is widely used for game AI [13], [14]. In the edge-cloud computing settings, it is assumed that the edge interacts with the environment in the field; however, it is difficult to build models at the edge, so the model-based approach is not considered in this paper.

The model-free approaches can be further divided into two communication styles: those sending experiences

[1]This paper is an extended version of our conference paper [12] by adding implementations of the proposed and existing architectures in a practical assumption and evaluations of the proposed approach in terms of training and transfer efficiency compared with the existing approach.

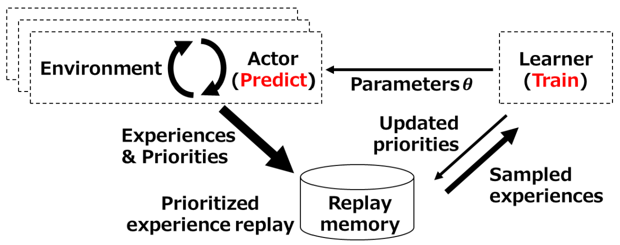


FIGURE 2. **Ape-X architecture.**

(e.g., transitions) and those sending gradients from actors to learner [15], [16]. In the case of those sending experiences, actors take actions and observe the environment to generate experiences, and then they send the experiences to the learner. The model is trained only by the learner, and then the trained model parameters are shared by the actors. Distributed Prioritized Experience Replay (Ape-X) [7] is a typical example of this communication style.

In the case of those sending gradients, on the other hand, each actor trains a local model and sends the gradients of the model to the learner. Then, the learner builds the global model based on the received gradients. Asynchronous Advantage Actor-Critic (A3C) [17] and Importance Weighted Actor-Learner Architecture (IMPALA) [18] are typical examples of this communication style.

In edge-cloud environments, the communication style of sending experiences has advantages in terms of the communication amount between actors and learner and the computation cost of actors compared with that of sending gradients. Thus, Ape-X is assumed as a baseline architecture of distributed deep reinforcement learning in this paper. Ape-X architecture is described in the following section.

B. DISTRIBUTED PRIORITIZED EXPERIENCE REPLAY (APE-X)

Figure 2 illustrates Ape-X architecture. Ape-X introduces a prioritized experience replay for large-scale distributed reinforcement learning systems that consist of actor processes, experience replay memory, and learner process. The actor processes take actions based on Q-network inference results and observe state transitions and rewards by the selected actions. The state transitions and rewards (i.e., experiences) are stored in an experience replay memory, and the Q-network model is trained using sampled experiences by the learner process.

Several improvements based on Ape-X have been proposed. In Recurrent Replay Distributed DQN (R2D2) [9], RNN (Recurrent Neural Network) based reinforcement learning agents are trained via distributed prioritized experience replay. Never Give Up (NGU) [10] improves R2D2 by introducing an episodic memory-based intrinsic reward to avoid the same state in the same episode. Agent 57 [11] improves NGU by selecting efficient internal reward reflection rates. All these prior works use distributed prioritized experience

Algorithm 1 Actor

```
1:  Pull parameters $\boldsymbol{\theta}_0$
2:  for $t = 1$ to $T$ do
3:      $a_t \leftarrow \epsilon$-greedy$(A)$
4:      $(r_t, \boldsymbol{s}_{tC1}) \leftarrow$ Environment$(a_t, \boldsymbol{s}_t)$
5:      LocalBuffer.Add$(\boldsymbol{s}_t, a_t, r_t, \boldsymbol{s}_{tC1})$
6:      if LocalBuffer.Size() $\geq$ Batch_Size then
7:          $\boldsymbol{\tau} \leftarrow$ LocalBuffer.Get(Batch_Size)
8:          $\boldsymbol{p} \leftarrow$ ComputePriorities$(\boldsymbol{\tau})$
9:          Push$(\boldsymbol{\tau}, \boldsymbol{p})$
10:     end if
11:     Pull $\boldsymbol{\theta}_t$ every $N_{pull}$ steps
12: end for
```

replay originally proposed in Ape-X; thus, experiences are sent from actors to learner in these prior works. Since we also follow this direction, our proposed approach can be applied to these Ape-X style distributed reinforcement learning systems.

In the following, the actor, experience replay memory, and learner processes in the Ape-X style architecture are explained.

1) ACTOR

Actor is a process that takes actions based on DQN inference results and observes the environment to generate experiences each of which consists of the original state, action, reward, and next state. Algorithm 1 shows actor's behavior. An actor makes inferences using model parameters $\boldsymbol{\theta}$ obtained from learner to select an action a_t at current time t (Line 3). ϵ-greedy is a well-known algorithm to select an action a_t from a set of possible actions. It can increase the diversity of action search by randomly selecting an action with a probability of ϵ. A different ϵ value is set to each actor. The actor then takes the selected action a_t and observes reward r_t and next state $\boldsymbol{s}_{tC1}$ from the environment so that an experience (a_t, $\boldsymbol{s}_t$, r_t, $\boldsymbol{s}_{tC1}$) is generated (Line 4). The generated experiences are temporarily stored in a local buffer (Line 5), and then those of a predefined batch size are transferred to an experience replay memory (Lines 6-10). $\boldsymbol{\tau}$ is a batch of experiences, and $\boldsymbol{p}$ is a batch of their priorities. A priority is assigned to each experience by the actor so that experiences that can accelerate the DQN training are preferentially trained by the learner (Line 8). A difference between estimated and target Q-values, called TD (Temporal Difference) error, is used as the priority of an experience.

2) LEARNER

Algorithm 2 shows learner's behavior. A learner samples experiences accumulated in an experience replay memory based on their priorities assigned by actors (Line 3). The learner uses the sampled experiences of a training batch size as training data, and then it updates parameters $\boldsymbol{\theta}$ of Q-function so that a loss value by the training data is

Algorithm 2 Learner

```
1:  $\boldsymbol{\theta_0}$ ← Initialized parameters
2:  for $t = 1$ to $T$ do
3:      $\boldsymbol{id}, \boldsymbol{\tau}$ ← Sampling(Batch_Size)
4:      $l_t$ ← ComputeLoss($\boldsymbol{\tau}$;$\boldsymbol{\theta_{t-1}}$)
5:      $\boldsymbol{\theta_t}$ ← UpdateParameters($l_t$;$\boldsymbol{\theta_{t-1}}$)
6:      UpdatePriorities($\boldsymbol{id}$)
7:      if Receive parameter pull request then
8:          Send $\boldsymbol{\theta_t}$ to actor
9:      end if
10:     Update replay memory every $N_{replay}$ steps
11: end for
```

Algorithm 3 Probabilistic Experience Sampling Using SumTree

```
Require: $0 \leq s$ (Random number) $\leq \Sigma_k p_k$
1: $n$ ← root
2: if $n$ is leaf_node then
3:     return $n$
4: end if
5: if $n$.left_val $\geq s$ then then
6:     return Sampling($n$.left, $s$)
7: else
8:     return Sampling($n$.right, $s$ - $n$.left_val)
9: end if
```

minimized (Line 5). More specifically, in DQN, the model parameters of Q-function are updated so that the Q-function can predict a sum of the latest reward r_{t+1} and the maximum expected reward in the next state $s_{t\mathcal{C}1}$ as follows.

$$Q(s_t, a_t) \leftarrow r_{t+1} + \gamma \max_{a_{t+1}} Q(s_{t\mathcal{C}1}, a_{t+1}), \quad (1)$$

where γ is a discount rate of reward. Since priorities of experiences once used for the training should be decreased, the learner updates the priorities of such experiences in the experience replay memory (Line 6). The model parameters are sent to an actor when a parameter pull request is received from the actor (Lines 7-9).

3) PRIORITIZED EXPERIENCE REPLAY

Although it is expected that preferentially using the experiences or state transitions with higher priorities can improve efficiency of training, there are some issues. Specifically, experiences with lower priorities may not be used for a long time, and an overfitting which becomes sensitive to noises may occur due to a limited diversity of experiences actually used. To address these issues, a probabilistic sampling of experiences based on their priorities [19] is used in recent distributed deep reinforcement learning. A sampling probability of an experience (or state transition) i is calculated based on priority p of the experience as follows.

$$P_i = \frac{p_i^{\alpha}}{\Sigma_k p_k^{\alpha}} (p_k \neq 0), \quad (2)$$

where α is a hyper-parameter that weights the priority and $\Sigma_k p_k$ means the sum of priorities of entire replay memory.

In the prioritized experience replay, data manipulation and probabilistic sampling of experiences can be efficiently implemented by using SumTree as a data structure. Algorithm 3 shows the probabilistic sampling using SumTree. By traversing the tree structure from root to leaf as described in Algorithm 3, a probabilistic sampling based on priority is implemented without reordering the experiences. Computational complexity of the probabilistic sampling is $O(\log N)$, where N is the number of experiences in SumTree.

Several improvement methods of the experience replay have been proposed. There are mainly two directions for the improvement: reducing off-policy (i.e., difference between the policy on training and the replay memory) and rapid convergence. Combined Experience Replay (CER) [20] reduces the off-policy by adding an experience generated by the latest parameters to the training batch after the sampling. Attentive Experience Replay (AER) [21] also reduces the off-policy by using similarities between states of experiences in the replay memory and current agent's states. Large Batch Experience Replay (LaBER) [22] improves the convergence speed by two-step sampling. The first step makes a larger batch than training batch from replay memory by uniform sampling, and the second step makes a training batch from the large batch by sampling based on the priority.

In this paper, we use a prioritized experience replay based sampling. Since these improved sampling methods described above do not affect the actor-learner communication architecture, our proposed approach can be combined with these sampling methods.

C. IMPROVEMENT METHODS OF TRANSFER EFFICIENCY OF DISTRIBUTED REINFORCEMENT LEARNING

As mentioned in Section I, there are two approaches to improve the transfer efficiency: accelerating the communication and sending useful experiences preferentially.

As an acceleration method of communication, Li et al. [23] assume a distributed reinforcement learning architecture, in which each computer is high-performance and the experience generation and learning take place on all the computers. In this case, gradient aggregation between computers becomes a performance bottleneck; thus, they propose to accelerate this process by performing the gradient aggregation within network switches. Our past work [12] also proposes an acceleration method of communication between actors and learner in Ape-X style architecture using DPDK and F-Stack.

As an improvement method for sending more useful experiences, Liu et al. [24] propose a distributed reinforcement learning architecture in which both actors and learner have a replay memory. The experience sampling is done twice in a hierarchical manner in both actors and learner. Since the sampling is done by actors too, it can reduce communication overheads between the actors and learner. However, they assume the same environment as in [7], where actor and learner nodes are located closely; in this case, workload of actors is increased because actors have a replay memory and sample their experiences. Their work is different from our assumed distributed reinforcement learning setting, and thus it cannot be applied to ours as is.

Chen et al. [25] propose a communication efficient policy for the distributed reinforcement learning, in which gradients are transferred from actors to learner. Their method adaptively skips the gradient communication during training iterations without degrading learning performance. However, their method cannot be applied to distributed reinforcement learning that transfers experiences.

In addition, the improvement methods of experience replay introduced in Section II-B can also be used for sending more useful experiences, because these methods aim to sample more effective experiences for training. Overall, our proposed approach is classified as an improvement method for sending more useful experiences to improve the transfer efficiency in the edge-cloud settings, but the prior works mentioned above do not focus on edge-cloud implementation where actors and learner's communication is long-haul as assumed in this paper. Although our previous work [12] also focused on such an edge-cloud environment as mentioned in Section I, this paper is quite different from the previous work since the previous work proposed an acceleration method of communication by DPDK.

D. APPLICATIONS OF DISTRIBUTED REINFORCEMENT LEARNING FOR EDGE-CLOUD ENVIRONMENT

Several applications of distributed reinforcement learning in edge-cloud environments have been proposed. Please note that most of them are based on Ape-X.

Boni et al. [26] propose a distributed reinforcement learning architecture for task offloading in autonomous IoT systems. In their architecture, IoT devices correspond to actors, and a cloud node corresponds to a learner; in addition, a smart access point is installed to aggregate information from the IoT devices, while a replay memory is included in the cloud node.

Liu et al. [27] propose a mobile crowdsensing optimization of unmanned vehicles using a distributed reinforcement learning. In their method, each unmanned vehicle forms a network that uses its own replay memory. The architecture is similar to Ape-X; since there are multiple vehicle networks in the system, a single learner maintains multiple replay memories and trains an individual model for each network.

Li et al. [28] propose a multipath TCP congestion control using a distributed reinforcement learning. In their architecture, multipath TCP connections are corresponding to actors. A server called "collector" receives experiences from clients. A learner node then receives the experiences from the collector. The learner node has a replay memory, and it

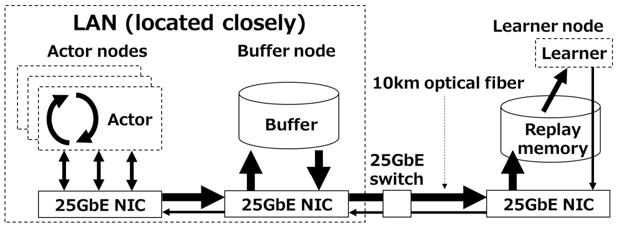


FIGURE 3. An implementation of baseline architecture.

trains the model using experiences sampled from the replay memory.

Geng et al. [29] propose a distributed reinforcement learning based computation offloading in vehicular edge computing networks. There are three components in their assumed environment: vehicles which interact with environment, mobile-edge servers which collect data from nearby vehicles, and a cloud server. The distributed reinforcement learning is based on Ape-X. The vehicles are corresponding to actors, and the cloud server is corresponding to a learner with replay memory. Mobile-edge servers collect experiences from the vehicles and send them to the replay memory of the cloud server.

Qiu et al. [30] propose a distributed reinforcement learning based computation offloading for smart home devices. This method is also based on Ape-X. Each device in a smart home is corresponding to an actor, and a cloud node is corresponding to a learner with replay memory. Management hubs of smart home devices collect experiences from these devices and send them to the replay memory of the cloud node.

These applications adopt a prioritized experience replay, and the replay memory is included in the learner node operated at the cloud side. Therefore, we use this architecture as a baseline to compare with our approach. The implementation of the baseline architecture in this paper is illustrated in Section III.

III. BASELINE DISTRIBUTED DEEP REINFORCEMENT LEARNING ARCHITECTURE

In distributed reinforcement learning for edge-cloud environment, there are three types of nodes; actor nodes interact with environment and generate experiences, a buffer node collects the experiences from the actor nodes, and a learner node trains model parameters. As mentioned in Section II-D, in the Ape-X based architecture, a replay memory is typically implemented in the learner side. This architecture is used as baseline in this paper.

Figure 3 shows an implementation of the baseline architecture that consists of actor node, buffer node, and learner node. Although multiple actors are running on a single actor node in this figure for simplicity, we can assume that they are running on multiple edge computers in practical situations. These nodes are connected via 25GbE. The actor and buffer nodes are connected via the same LAN, while the buffer and

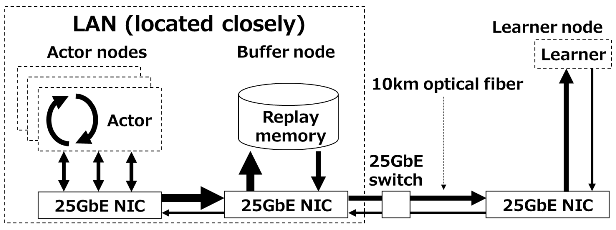


FIGURE 4. An implementation of proposed architecture.

learner nodes are connected via a 10km optical fiber cable since we assume that the learner is running at the cloud side.

The behaviors of the actor and learner nodes are same as those in Algorithms 1 and 2. The buffer node acts as a buffer between the actor and learner nodes. Specifically, the buffer node receives experiences from actors, stores them in a local buffer, and then transfers the buffered experiences to the learner when it receives a send request from the learner during the periodic update of replay memory in Algorithm 2. The learner then updates replay memory using the received experiences.

Figure 3 shows the communication flow over the actor, buffer, and learner nodes. The data transfer amount from actor to buffer is determined by given hyperparameters, that from buffer to learner depends on the replay memory size, and that from learner to actor via buffer depends on the model parameter size. Especially, data transfer amount from buffer to learner is large as shown in a big arrow in Figure 3.

Algorithm 4 Receiving Process of Buffer From Actors
```
1: while Replay memory is not full do
2:     Receive $\boldsymbol{\tau}, \boldsymbol{p}$ from actor
3:     Replay.Add($\boldsymbol{\tau}, \boldsymbol{p}$)
4: end while
5: for $t = 1$ to $T$ do
6:     Receive $\boldsymbol{\tau}, \boldsymbol{p}$ from actor
7:     Replay.Update($\boldsymbol{\tau}, \boldsymbol{p}$)
8:     if Receive parameter pull request then
9:         Send $\boldsymbol{\theta}_t$ to actor
10:    end if
11: end for
```

Algorithm 5 Sending Process of Buffer to Learner
```
1: Wait for replay memory to be full
2: for $t = 1$ to $T$ do
3:     if Receive $\boldsymbol{p}_{\boldsymbol{id}_{old}}$, $\boldsymbol{id}_{old}$ from learner then
4:         if $\boldsymbol{\tau}_{\boldsymbol{id}_{old}}$ have not been updated then
5:             UpdatePriorities($\boldsymbol{id}_{old}$)
6:         end if
7:         $\boldsymbol{id}, \boldsymbol{\tau} \leftarrow$ Sampling(Batch_Size)
8:         Push($\boldsymbol{id}, \boldsymbol{\tau}$)
9:     end if
10:    if Receive new parameters from learner then
11:        Update $\boldsymbol{\theta}_t$
12:    end if
13: end for
```

IV. PROPOSED DISTRIBUTED DEEP REINFORCEMENT LEARNING ARCHITECTURE

A. OVERVIEW OF PROPOSED APPROACH

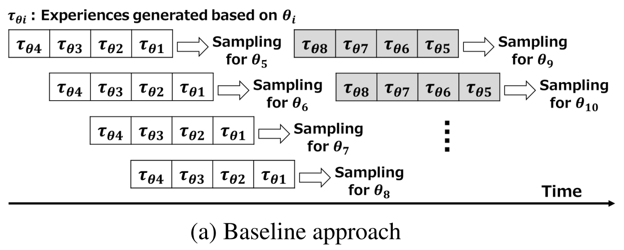
Figure 4 shows an overview of the proposed architecture. As well as the baseline architecture, it consists of actor node, buffer node, and learner node. The behavior of the actors is the same as that of the baseline. The major difference from the baseline is that the buffer node includes the replay memory functionality, and it sends only sampled experiences to the learner. As shown in a slightly narrow arrows from buffer to learner in Figure 4, a batch of sampled experiences is transferred to the learner whenever the learner consumes a batch of the experiences for the model training. This may increase the frequency of data transfers from the buffer to learner nodes, while it can transfer only the sampled experiences, which stochastically have higher priorities, thereby increasing the transfer efficiency compared to the baseline architecture.

B. PROCESSES OF PROPOSED APPROACH

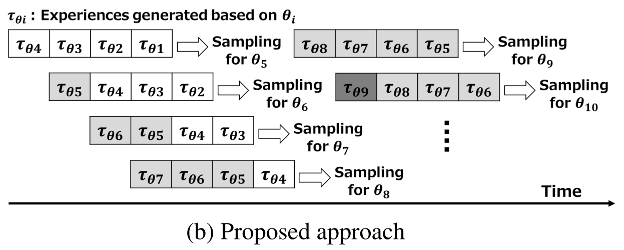
In the proposed approach, modifications are not needed for the actors, while the algorithms of the buffer and learner nodes are changed from the baseline. Especially, a main difference from the baseline is that the buffer node has a replay memory. The buffer node receives the experiences from the actors and adds them to the replay memory. In the proposed approach, contents of replay memory are sequentially updated in a FIFO manner, while an entire replay memory is updated at once in the baseline.

Because the buffer node communicates with both actor and learner nodes asynchronously, Algorithm 4 shows the receiving part of the buffer node from actors, and Algorithm 5 shows the sending part of the buffer node to learner. Variables in these algorithms are same as those in Algorithms 1 and 2. In these new algorithms, $\boldsymbol{\tau}$ is a batch of received experience, $\boldsymbol{p}$ is a batch of their priorities, and $\boldsymbol{id}$ is a batch of IDs of experiences which are extracted by the sampling. A subscript *old* means that these variables are corresponding to a previous training batch.

At the startup time, since no experience is accumulated in the replay memory, the received experiences are only stored in the replay memory (and thus they are not sent to the learner) until the replay memory becomes full (Lines 1-4 in Algorithm 4). After the replay memory becomes full, whenever new experiences are received, these experiences are added to the buffer while the oldest experiences in the buffer are deleted (Lines 6-7 in Algorithm 4). When the buffer node receives a sending request from the learner, $\boldsymbol{p}_{\boldsymbol{id}_{old}}$ and $\boldsymbol{id}_{old}$ of the previous training batch are also received from the learner (Line 3 in Algorithm 5). Then, it can update the priorities related to $\boldsymbol{id}_{old}$ (Line 5 in Algorithm 5). Please note that there is a possibility that the experiences of $\boldsymbol{id}_{old}$ have



(a) Baseline approach



(b) Proposed approach

FIGURE 5. Sampling method of replay memory in baseline approach (a) and that in proposed approach (b).

been already updated due to the receipt of new experiences from actors asynchronously. Therefore, only the priorities of the experiences which have not been updated are updated (Lines 4-6 in Algorithm 5). This update is followed by the sampling that adds new experiences to the buffer (Lines 7-8 in Algorithm 5). The latest model parameters are sent from the learner to actors periodically. When the buffer node receives new parameters, $\boldsymbol{\theta_t}$ are updated (Lines 10-12 in Algorithm 5).

The learner algorithm is changed as follows. First, the new algorithm eliminates the periodic updates of replay memory. Second, the learner requests the buffer node to fetch a batch of experiences instead of sampling the local replay memory. Third, the learner sends the model parameters $\boldsymbol{\theta_t}$ every N_{update} steps to the buffer node instead of sending them to an actor when the learner received a parameter pull request from the actor.

The proposed approach has two advantages over the baseline. The first advantage is the efficient use of communication bandwidth. In the proposed approach, only the sampled experiences are sent to the learner node. Thus, higher priority experiences are more likely to be sampled, and all the transferred experiences are used for the model training. In the baseline approach, on the other hand, experiences coming from actors are directly transferred to the learner. Since lower priority experiences may not be sampled for training at the learner side, many transferred experiences are not used for the training, which decreases the communication efficiency.

The second advantage is that the proposed approach can prevent overfitting, because in our approach experiences in a replay memory used for the sampling are different for each batch transfer. Figure 5 illustrates experiences stored in the replay memory for sampling in the cases of the proposed and baseline approaches. In Figure 5, white-colored cells represent experiences generated based on $\boldsymbol{\theta_{1-4}}$, while dark-colored cells represent those generated based on newer parameters $\boldsymbol{\theta_{5-8}}$. As shown in Algorithm 2, the model parameters are updated every N_{update} times. In the proposed approach, immediately after the parameters are updated, the experiences obtained by the updated parameters are added to the replay memory by a FIFO manner, so that the replay memory is updated like a sliding window. In other words, experiences based on the parameters closer to the latest ones can be included in the replay memory as shown in Figure 5(b).

Such a FIFO method is one of updating methods of replay memory [31], [32], and this method is suitable for the proposed approach to avoid overfitting and improve the transfer efficiency of the proposed approach. The benefit of this FIFO manner is evaluated and discussed in Section V. On the other hand, in the baseline approach, the entire replay memory is updated at the same time but with a lower frequency, so the same experiences tend to be sampled for the training repeatedly until then (see Figure 5(a)). Thanks to this modification, the proposed approach can prevent overfitting and use the most recent parameters for the sampling, thereby increasing the training efficiency.

V. EVALUATIONS

A. EVALUATION ENVIRONMENT

In this evaluation, a separate machine is used for each of actor node, buffer node, and learner node. These machines are connected by 25GbE. In particular, the buffer and learner nodes are connected via a 25GbE switch (QNAP QSW-M5216-1T) and a 10km optical fiber for 25GBASE-LR, to build a long-haul communication between the buffer and learner nodes assuming practical use cases. The actor and buffer nodes are connected via a direct attach copper cable. Specifications of these machines are the same. Table 1 shows the specification of the three machines.

TABLE 1. Specification of actor, buffer, and learner machines (three machines with the same specification).

OS	Ubuntu 22.04.2 LTS
CPU	Intel Core i7-12700 4.8GHz
Memory	64GB
GPU	NVIDIA GeForce RTX 3090 × 1
25GbE NIC	Intel XXV710-DA2
CUDA	11.6.124
PyTorch	1.13

As a benchmark environment, we use Atari breakout included in OpenAI Gym environment [33]. Dueling Network Architecture [34] is a well-known deep neural network model used in DQNs such as in [7]. It is used in our evaluation in cooperation with double-DQN and n-step bootstrap target ($n = 3$) techniques, with a parameter size of approximately 13MB. Training batch size of experiences at learner is set to 512, and the data size of the batch is approximately 120MB. The size of replay memory and

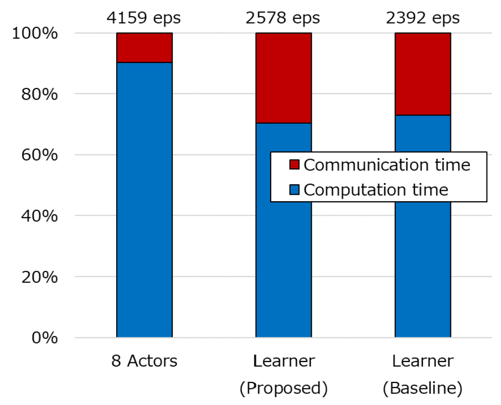


FIGURE 6. **Execution time breakdown and throughput ("eps" means experiences per second).**

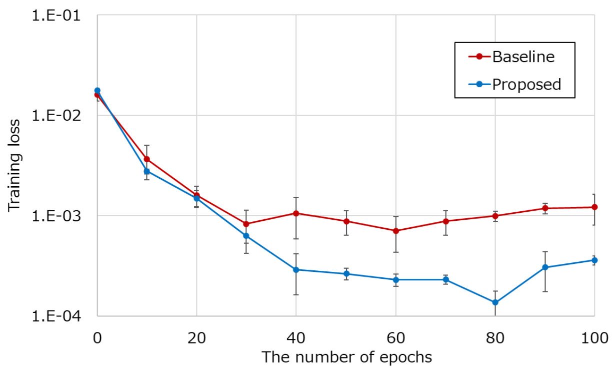


FIGURE 7. **Training loss L_t of proposed and baseline approaches.**

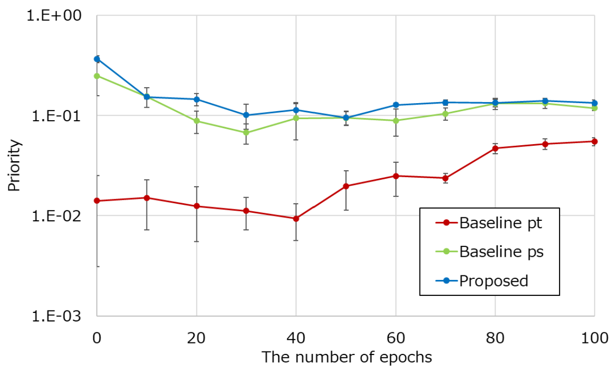


FIGURE 8. **Transferred priority p_t and sampled priority p_s of proposed and baseline approaches.**

the number of actors are set to 65,536 and 8, respectively, unless otherwise noticed (these values are varied in some evaluations). The number of experiences trained during an epoch is 65,536, which is equal to the replay memory size. N_{update} in Algorithm 2 is set to 16 in all the evaluations. Thus, the model parameters in a local buffer are updated every 16 training batches, and the parameters are pulled by the actors asynchronously. The data size of replay memory is approximately 1.6GB. As a practical implementation, it is assumed that the actors are implemented with edge computers, but in this evaluation, multiple actors are executed on a single machine for simplicity.

B. EXECUTION TIME BREAKDOWN

Figure 6 shows the execution time breakdown and throughput of the actors (the number of actors is 8), the learner of the proposed approach, and the learner of the baseline approach. In this figure, the graphs show the breakdown of each execution time, in which the blue part corresponds to the computation time and the red part corresponds to the communication time. The numbers above the graphs show their throughput. The update frequency of replay memory for the baseline approach is set to once per one epoch to normalize the transfer amount per learning in the proposed and baseline approaches.

The throughput difference between the proposed and baseline approaches is whether the experience sampling is done before the experiences are transferred to the learner or not. The proposed approach that performs the sampling before the communication can shorten the learner's computation time. The proportion of communications is 10% for the actor, 30% for the learner in the proposed approach, and 27% for the learner in the baseline approach. Even when these nodes are interconnected by a high-bandwidth 25GbE network, the communication overhead is significant, which reduces the throughput for the model training. If the transfer amount is further increased, the throughput is also reduced, so it is important to extract and transfer experiences that are expected to be useful for the model training.

C. COMPARISON BETWEEN PROPOSED AND BASELINE APPROACHES

To compare the proposed and baseline approaches, the game score of the benchmark can be considered as one of evaluation metrics, but the evaluation using the game score is too time-consuming and shows a significantly different result for each trial, making it difficult to evaluate our approach from various perspectives due to time constraints. In this evaluation, thus, the following three metrics are used to evaluate the training and communication efficiencies: L_t, p_t, and p_s. L_t is the training loss. p_t is the average priority of experiences sent to the learner node (called "transferred priority"), and p_s is the average priority of the sampled experiences (called "sampled priority").

In the baseline approach, all the experiences stored in a replay memory are transferred from the buffer to the learner; thus, the transfer amount depends on the replay memory size. In the proposed approach, since only sampled experiences are transferred, the transferred and sampled experiences are the same (i.e., $p_t = p_s$). In this section, the proposed and baseline approaches are compared in term of the above-mentioned three evaluation metrics. Then, additional evaluations are conducted with different numbers of actors and different sizes of replay memory to analyze characteristics of the proposed approach.

1) COMPARISON IN STANDARD CASE

Figure 7 shows L_t of the proposed and baseline approaches, while Figure 8 shows p_t and p_s of the proposed and baseline approaches. In the proposed approach, Figure 8 shows only p_s since p_t is equivalent to p_s. In these figures, X-axis shows the number of epochs for the training and Y-axis shows the training loss or the priorities in a logarithmic scale. The training loss and the priorities are the averaged values over 10 trials. Their standard deviations are also shown as the error bars. To keep the number of batch trainings for the proposed and baseline approaches the same, the replay memory is updated every 128 batch trainings in the baseline approach. Thus, it is updated once per epoch because the replay memory size is 128 times larger than the batch size for the training.

In terms of the training loss, the difference between these approaches is small at the beginning of the training, while the loss of the proposed approach is lower than that of the baseline approach at the 40 epochs or more. The training loss of the proposed approach after the training loss is converged (i.e., average training loss during 50-100 epochs in Figure 7) is 26% of the baseline approach, and this shows that the proposed approach has a higher training efficiency than the baseline approach.

In terms of the transferred priority, the priority of the proposed approach is higher than the baseline approach, which means that the proposed approach can transfer higher priority experiences efficiently. However, the difference of the transferred priority between the proposed and baseline approaches is gradually decreasing as the training progresses; for example, the transferred priority of the proposed approach is 26.1 times higher than that of the baseline at the beginning of the training while it is 3.92 times higher after the training loss is converged (i.e., average transferred priority during 50-100 epochs in Figure 8). There are two reasons of the decrease. The first reason is that the wasted experiences (e.g., those when the game is over with 0 points) are decreased as the training progresses. The second reason is related to the difference of the training loss between the proposed and baseline approaches. Because the transferred priority tends to be decreased as the training loss decreases, the priority of the proposed approach becomes low compared to the beginning of the training.

In terms of the sampled priority, the priority of the proposed approach is higher than the baseline at the beginning of training, though contents of the replay memory should be the same in both the approaches at the beginning of training. This is due to the difference of the update method for the replay memory. In the following, we evaluate the training loss and the sampled priority within each epoch to further analyze this difference. Figures 9 and 10 show L_t and p_s of the proposed and baseline approaches within each epoch. X-axis shows the number of trained batches in each epoch, and Y-axis shows training loss or sampled priority. In these figures, the results at each number of trained batches are averaged values of 100 epochs which were presented in

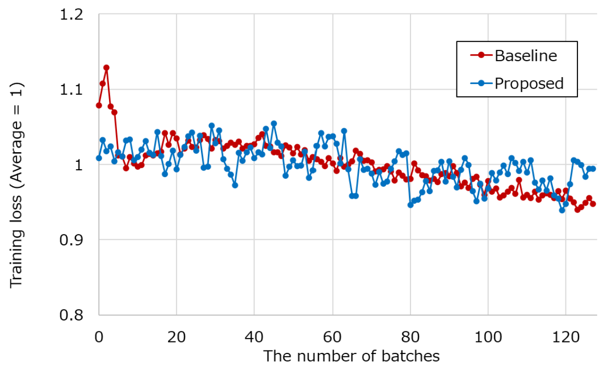


FIGURE 9. Training loss L_t of proposed and baseline approaches within each epoch.

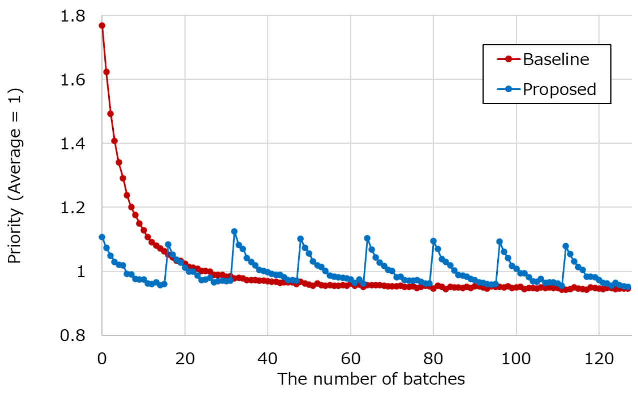


FIGURE 10. Sampled priority p_s of proposed and baseline approaches within each epoch.

Figures 7 and 8. These Y-axis values are normalized so that the average values become 1.

The major difference between the proposed and baseline approaches is the change pattern of the sampled priority. The sampled priority of the baseline approach is very high at the beginning of each epoch and then gradually decreases. In the baseline approach, a replay memory is updated only once at the beginning of each epoch, and then the batch training is repeated using the same replay memory. The sampled priority gradually decreases as the number of trained batches increases, because priorities of sampled experiences are updated with low values after these experiences are used for the training. Thus, experiences with low priorities become more likely to be sampled as the training progresses since priorities of the sampled (thus, high priority) experiences become low. In the proposed approach, on the other hand, the sampled priority decreases and then increases periodically. This cyclic period is same as the update cycle of the model parameters for the actors. The sampled priority becomes high temporarily because new experiences generated by new model parameters are included in the replay memory. In the proposed approach, peaks of the sampled priority are 1.10 times higher than the overall average. These peaks are significantly low compared to the peak of sampled priority

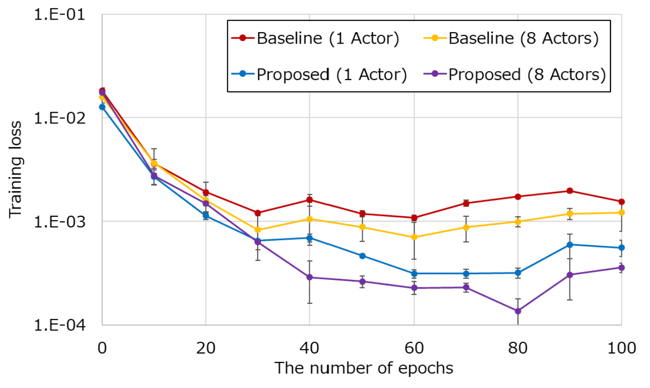


FIGURE 11. Training loss L_t of proposed and baseline approaches when the number of actors is changed.

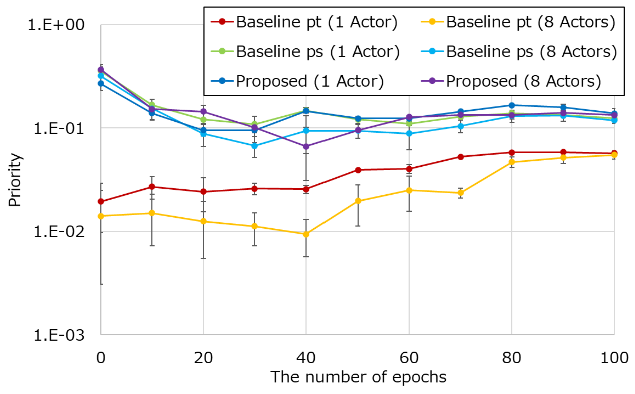


FIGURE 12. Transferred priority p_t and sampled priority p_s of proposed and baseline approaches when the number of actors is changed.

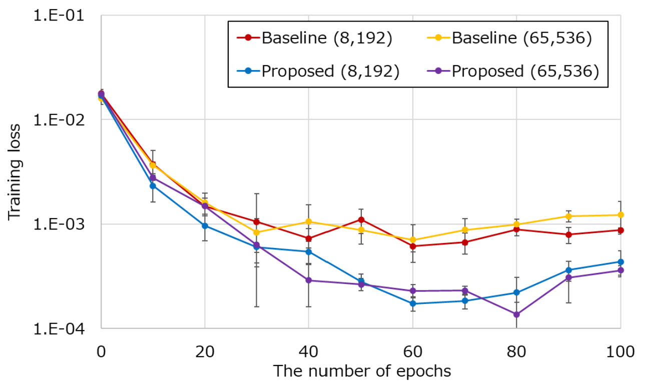


FIGURE 13. Training loss L_t of proposed and baseline approaches when replay memory size is changed.

of the baseline at the beginning of the training, which is 1.77 times higher than the overall average. As mentioned earlier, the average sampled priority of the proposed approach within the first epoch is higher than the baseline approach. This is because, although there is no difference between the proposed and baseline approaches at the first batch, after that the sampled priority of the baseline approach is sharply decreased compared to the proposed approach.[2] This is the reason for the difference of the sampled priority between the proposed and baseline approaches at the beginning of training in Figure 8.

In terms of the training loss, the range of fluctuation is much smaller than that of the sampled priority. However, the training loss of the baseline approach shows a small increase at the beginning of the epoch, suggesting the repeated ineffective trainings on the same replay memory. The proposed approach does not show such a tendency, so it can effectively train the model.

2) DIFFERENT NUMBERS OF ACTORS

Since the number of experiences generated at a time is increased as the number of actors is increased, the proposed and baseline approaches can be evaluated with different speeds of experience generation by changing the number of actors. Figures 11 and 12 show L_t, p_t, and p_s of the proposed and baseline approaches when the number of actors is 1 and 8. X-axis shows the number of epochs, and Y-axis shows the training loss or the priorities as well as those in Figures 7 and 8.

In both the approaches, the training loss becomes low at the 40 epochs or more by changing the number of actors from 1 to 8. This result shows the faster the speed of experience generation, the more efficient the learning process. The training loss of the proposed approach is lower than that of the baseline approach regardless of the number of actors, indicating that the proposed approach is effective even when the speed of experience generation is low.

[2]Note that Y-axis of Figure 10 is normalized so that the averaged value within each approach becomes 1.

In terms of priorities, the transferred priority of the baseline approach decreases when the number of actors is increased. This is because the probability that the low-priority experiences will be generated becomes high when the training loss is low. Considering this effect, the proposed approach can extract high-priority experiences despite the low training loss; thus, the proposed approach is highly efficient in terms of the experience extraction.

3) DIFFERENT SIZES OF REPLAY MEMORY

Figures 13 and 14 show L_t, p_t, and p_s of the proposed and baseline approaches when the replay memory size is 8,192 and 65,536. In this evaluation, the number of batch trainings within each epoch is fixed at 128 for a fair comparison between different replay memory sizes. However, the transfer frequency of replay memory is changed to once per 16 batch trainings when the replay memory size is 8,192, so that the ratio of the number of transferred experiences and the number of experiences used for the model training is fixed. X-axis shows the number of epochs, and Y-axis shows the training loss or the priorities as well as those in Figures 7 and 8.

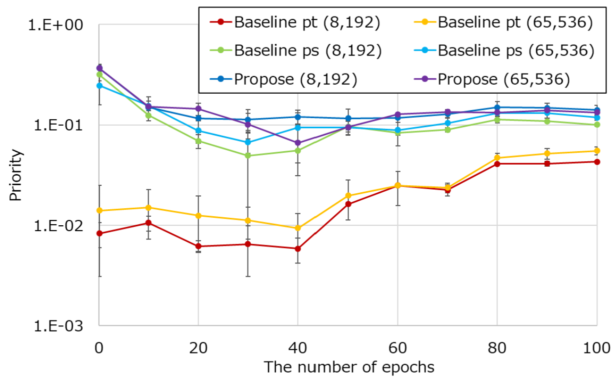


FIGURE 14. Transferred priority p_t and sampled priority p_s of proposed and baseline approaches when replay memory size is changed.

Basically, a larger replay memory size contributes to higher priorities since the larger replay memory has more experiences to be extracted. In the baseline approach, the transferred and sampled priorities tend to be low when the replay memory size is 8,192, while this tendency is not observed in the proposed approach. In the proposed approach, new experiences, which have stochastically higher priorities, are always included in the replay memory, and this can compensate for the decrease of the priorities due to the small replay memory.

In the baseline approach, the training loss does not increase when the replay memory size is decreased from 65,536 to 8,192 though the priorities become low. In this evaluation, the update frequency of the replay memory is high when the replay memory size is 8,192; it is expected that such high frequency updates of replay memory can mitigate a negative impact of the overfitting induced by the repeated training for the same replay memory.

VI. DISCUSSIONS

A. CONDITION FOR EFFECTIVENESS OF PROPOSED APPROACH

The evaluation results of the baseline approach show that the repeated training against the same replay memory causes overfitting, while the proposed approach is effective to prevent this kind of overfitting. Other than our proposed approach, another approach that transfers experiences and updates replay memory more frequently may be able to prevent the overfitting. In this alternative approach, only a part of the replay memory is transferred at a time and the replay memory is updated in a FIFO manner. Such an update method of replay memory is similar to the proposed approach as shown in Figure 5. However, unlike the proposed approach, this alternative approach requires all the experiences in the replay memory, which means that experiences with lower priorities are also transferred. Thus, this alternative will suffer from negative impacts of the lower transferred priority as well as the baseline approach.

The benefit of the extracting higher-priority experiences by the proposed approach depends on the balance between the generation rate of experiences and the consumption rate of experiences for the training. Here, this ratio is denoted as α, and it can be calculated by the number of experiences generated in a given time period divided by the number of experiences used for the training in the same period. If α is less than 1, the extraction of experiences by the proposed approach does not improve the transfer efficiency because all the experiences are transferred to learner node.[3] On the other hand, when α is greater than 1, the training efficiency can be improved by extracting only higher-priority experiences.

When the number of actors is 8 in our evaluation, α is 1.52, which means that the proposed approach improves the transfer efficiency in our environment. In this evaluation environment, more than 8 actors were not implemented in the actor node due to a performance limitation of our single actor machine, but further efficiency improvement would be expected with a larger number of actors. For example, 360 actors run on 360 CPUs in [7]. The proposed approach is considered to be more effective in such a large environment. Please note that there is a possibility that the proposed approach may still transfer the same experiences multiple times when sampling targets are overlapped; thus, further efficiency improvement can be expected by newly introducing a cache on the learner side to reuse experiences selected multiple times. Exploring such optimization techniques is our future work.

B. EFFECTIVENESS AND SCALABILITY OF PROPOSED APPROACH FOR REAL-WORLD APPLICATIONS

To evaluate the proposed distributed reinforcement learning approach in practical environments, in this paper, we built a test environment where the buffer and learner nodes are connected via a 25GbE switch and 10km optical fiber. However, we should consider the following two additional cases in our test environment for practical applications including those introduced in Section II-D:

- A situation where the communication distance and bandwidth between the buffer and learner nodes become longer and narrower, and
- A situation where multiple sets of actor and buffer nodes exist.

In the first situation, the communication overhead increases and the transfer efficiency of experiences to the learner node decreases. In this case, since the consumption rate of experiences by the learner node decreases, α would be increased from 1.52 which is corresponding to our test environment. As mentioned in Section VI-A, the training efficiency can be improved when α is greater than 1; thus, it is expected that the proposed approach is effective in the first situation where α is greater than 1.52.

[3]Strictly speaking, the baseline approach requires sampling of experiences at a learner node, which incurs an additional computation time at the learner, so the training throughput will be lower than that of the proposed approach. However, for the sake of simplicity, the training throughput is assumed to be constant in this discussion.

The second situation may occur in applications where experiences are collected from larger systems (e.g., multiple LANs) as in [26], [27], [28], [29], and [30]. The proposed approach can address such a situation by adding more buffer nodes each of which has its own experience replay memory. In this case, since the data processing between the actor and buffer nodes in an LAN is independent from that of the other LANs, Algorithm 4 will not be changed. On the other hand, Algorithm 5 should be slightly modified so that the experience sampling can be done from multiple replay memories. More specifically, the experience sampling should be modified so that $\frac{N}{R}$ experiences are sampled from each replay memory, where N is the batch size and R is the number of replay memories. Thus, the proposed approach can be used for the practical applications that felt into the second situation. To address the heterogeneity of various computing platforms, it is expected that multiple buffer nodes are implemented on multiple edge servers; in this case, the number of experiences sampled from each buffer node should be tuned depending on the experience generation rate of each buffer node (e.g., more experiences should be sampled from a buffer node that produces more experiences). As the number of the buffer and actor nodes increases, the learner node would be a performance bottleneck; such a scalability issue can be addressed by introducing a hierarchical structure as in [24], and this direction is our future work.

VII. CONCLUSION

To improve the performance of the deep reinforcement learning such as DQN, the distributed deep reinforcement learning using a cluster of computers is a promising approach. In a practical use case of distributed deep reinforcement learning, since it is assumed that learner node is located in cloud side while actor nodes are located in edge side, the communication overheads negatively impact the training efficiency. We implemented the realistic environment with three types of nodes (i.e., actor, buffer, and learner nodes), in which the buffer and learner nodes are connected via a 25GbE switch and a 10km optical fiber. In this environment, the communication overhead accounts for 27% of the execution time of the learner in the baseline approach. This suggests a high importance to extract and transfer only experiences which are useful for the model training.

In this paper, we proposed an architecture that places the replay memory on the buffer node, whereas existing methods place it on the learner side. We evaluated the proposed and existing approaches in terms of the training loss and priorities of the transferred experiences. The evaluation results using the Atari breakout game show that the training loss of the proposed approach is 26% of the existing architecture. The transferred priority of experiences is 3.92 times higher than the existing approach just after the training loss is converged. These results demonstrate that the proposed approach outperforms the existing approach in terms of both the training efficiency and the transfer efficiency of experiences in long-haul cloud-edge communication environments.

REFERENCES

[1] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, Jan. 2016.

[2] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, and S. Levine, "QT-opt: Scalable deep reinforcement learning for vision-based robotic manipulation," 2018, *arXiv:1806.10293*.

[3] M. M. Afsar, T. Crump, and B. Far, "Reinforcement learning based recommender systems: A survey," *ACM Comput. Surv.*, vol. 55, no. 7, pp. 1–38, Dec. 2022.

[4] B. R. Kiran, I. Sobh, V. Talpaert, P. Mannion, A. A. A. Sallab, S. Yogamani, and P. Pérez, "Deep reinforcement learning for autonomous driving: A survey," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 6, pp. 4909–4926, Jun. 2022.

[5] N. Le, V. S. Rathour, K. Yamazaki, K. Luu, and M. Savvides, "Deep reinforcement learning in computer vision: A comprehensive survey," *Artif. Intell. Rev.*, vol. 55, no. 4, pp. 2733–2819, Apr. 2022.

[6] Y. Xiao, J. Liu, J. Wu, and N. Ansari, "Leveraging deep reinforcement learning for traffic engineering: A survey," *IEEE Commun. Surveys Tuts.*, vol. 23, no. 4, pp. 2064–2097, 4th Quart., 2021.

[7] D. Horgan, J. Quan, D. Budden, G. Barth-Maron, M. Hessel, H. van Hasselt, and D. Silver, "Distributed prioritized experience replay," in *Proc. Int. Conf. Learn. Represent.*, May 2018, pp. 1–19.

[8] A. Nair, P. Srinivasan, S. Blackwell, C. Alcicek, R. Fearon, A. De Maria, V. Panneershelvam, M. Suleyman, C. Beattie, S. Petersen, S. Legg, V. Mnih, K. Kavukcuoglu, and D. Silver, "Massively parallel methods for deep reinforcement learning," 2015, *arXiv:1507.04296*.

[9] S. Kapturowski, G. Ostrovski, W. Dabney, J. Quan, and R. Munos, "Recurrent experience replay in distributed reinforcement learning," in *Proc. Int. Conf. Learn. Represent.*, May 2019, pp. 1–19.

[10] A. P. Badia, P. Sprechmann, A. Vitvitskyi, D. Guo, B. Piot, S. Kapturowski, O. Tieleman, M. Arjovsky, A. Pritzel, A. Bolt, and C. Blundell, "Never give up: Learning directed exploration strategies," in *Proc. Int. Conf. Learn. Represent.*, Apr. 2020, pp. 1–28.

[11] A. P. Badia, B. Piot, S. Kapturowski, P. Sprechmann, A. Vitvitskyi, Z. D. Guo, and C. Blundell, "Agent57: Outperforming the Atari human benchmark," in *Proc. Int. Conf. Mach. Learn.*, Jul. 2020, pp. 507–517.

[12] M. Furukawa and H. Matsutani, "Accelerating distributed deep reinforcement learning by in-network experience sampling," in *Proc. 30th Euromicro Int. Conf. Parallel, Distrib. Netw.-Based Process. (PDP)*, Mar. 2022, pp. 75–82.

[13] J. Li, S. Koyamada, Q. Ye, G. Liu, C. Wang, R. Yang, L. Zhao, T. Qin, T.-Y. Liu, and H.-W. Hon, "Suphx: Mastering mahjong with deep reinforcement learning," 2020, *arXiv:2003.13590*.

[14] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Debiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, and R. Jozefowicz, "Dota 2 with large scale deep reinforcement learning," 2019, *arXiv:1912.06680*.

[15] Q. Yin, T. Yu, S. Shen, J. Yang, M. Zhao, K. Huang, B. Liang, and L. Wang, "Distributed deep reinforcement learning: A survey and a multi-player multi-agent learning toolbox," 2022, *arXiv:2212.00253*.

[16] X. Wang, S. Wang, X. Liang, D. Zhao, J. Huang, X. Xu, B. Dai, and Q. Miao, "Deep reinforcement learning: A survey," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 35, no. 4, pp. 5064–5078, Apr. 2024.

[17] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 1928–1937.

[18] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning, S. Legg, and K. Kavukcuoglu, "IMPALA: Scalable distributed deep-RL with importance weighted actor-learner architectures," 2018, *arXiv:1802.01561*.

[19] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," in *Proc. Int. Conf. Learn. Represent.*, May 2016, pp. 1–21.

[20] S. Zhang and R. S. Sutton, "A deeper look at experience replay," 2017, *arXiv:1712.01275*.

[21] P. Sun, W. Zhou, and H. Li, "Attentive experience replay," in *Proc. AAAI Conf. Artif. Intell.*, vol. 34, Apr. 2020, pp. 5900–5907.

[22] T. Lahire, M. Geist, and E. Rachelson, "Large batch experience replay," 2021, *arXiv:2110.01528*.

[23] Y. Li, I.-J. Liu, Y. Yuan, D. Chen, A. Schwing, and J. Huang, "Accelerating distributed reinforcement learning with in-switch computing," in *Proc. ACM/IEEE 46th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2019, pp. 279–291.

[24] S. Liu, P. Qiao, Y. Dou, and R. Li, "Ddper: Decentralized distributed prioritized experience replay," in *Proc. IEEE Int. Conf. Multimedia Expo (ICME)*, Jul. 2021, pp. 1–6.

[25] T. Chen, K. Zhang, G. B. Giannakis, and T. Basar, "Communication-efficient policy gradient methods for distributed reinforcement learning," *IEEE Trans. Control Netw. Syst.*, vol. 9, no. 2, pp. 917–929, Jun. 2022.

[26] A. K. Chabi Sika Boni, Y. Hablatou, H. Hassan, and K. Drira, "Distributed deep reinforcement learning architecture for task offloading in autonomous IoT systems," in *Proc. 12th Int. Conf. Internet Things*, Nov. 2022, pp. 112–118.

[27] C. H. Liu, Z. Dai, Y. Zhao, J. Crowcroft, D. Wu, and K. K. Leung, "Distributed and energy-efficient mobile crowdsensing with charging stations by deep reinforcement learning," *IEEE Trans. Mobile Comput.*, vol. 20, no. 1, pp. 130–146, Jan. 2021.

[28] W. Li, H. Zhang, S. Gao, C. Xue, X. Wang, and S. Lu, "SmartCC: A reinforcement learning approach for multipath TCP congestion control in heterogeneous networks," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 11, pp. 2621–2633, Nov. 2019.

[29] L. Geng, H. Zhao, J. Wang, A. Kaushik, S. Yuan, and W. Feng, "Deep reinforcement learning based distributed computation offloading in vehicular edge computing networks," *IEEE Internet Things J.*, vol. 10, no. 14, pp. 12416–12433, Jul. 2023.

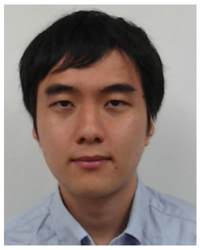
[30] X. Qiu, L. Liu, W. Chen, Z. Hong, and Z. Zheng, "Online deep reinforcement learning for computation offloading in blockchain-empowered mobile edge computing," *IEEE Trans. Veh. Technol.*, vol. 68, no. 8, pp. 8050–8062, Aug. 2019.

[31] T. de Bruin, J. Kober, K. Tuyls, and R. Babuska, "Improved deep reinforcement learning for robotics through distribution-based experience retention," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Oct. 2016, pp. 3947–3952.

[32] L. Zhang, Z. Zhang, Z. Pan, Y. Chen, J. Zhu, Z. Wang, M. Wang, and C. Fan, "A framework of dual replay buffer: Balancing forgetting and generalization in reinforcement learning," in *Proc. Workshop Scaling Up Reinforcement Learn.*, Aug. 2019, pp. 1–7.

[33] *OpenAI Gym*. Accessed: Dec. 26, 2023. [Online]. Available: https://gym.openai.com

[34] Z. Wang, N. D. Freitas, and M. Lanctot, "Dueling network architectures for deep reinforcement learning," in *Proc. Int. Conf. Int. Conf. Mach. Learn.*, vol. 48, Jun. 2016, pp. 1995–2003.



SHIN MORISHIMA received the B.E., M.E., and Ph.D. degrees from Keio University, in 2014, 2016, and 2018, respectively. He is currently an Assistant Professor with Toyama Prefectural University.

HIROKI MATSUTANI (Member, IEEE) received the B.A., M.E., and Ph.D. degrees from Keio University, in 2004, 2006, and 2008, respectively. He is currently a Professor with the Department of Information and Computer Science, Keio University. His research interests include the areas of computer architecture and interconnection networks.