Computer Science and Information Systems 17(3):891–906 https://doi.org/10.2298/CSIS190917026K

A Load Balancing Scheme for Gaming Server applying Reinforcement Learning in IoT *

Hye-Young Kim[1] and Jinsul Kim[2]

[1] School of Games, Major of Game Software,
Hongik University, Korea
hykim@hongik.ac.kr

[2] School of Electronics and Computer Engineering,
Chonnam National University, Gwangju, South Kore
jsworld@jnu.ac.kr

Abstract. A lot of data generated on the game server causes overtime in IoT environment. Recently, both researchers and developers have developed great interests in load balancing schemes in gaming servers. The existing literature have proposed algorithms that distribute loads in servers by mostly concentrating on load balancing and cooperative offloading in Internt of Things (IoT) environment. The dynamic load balancing algorithms have applied a technique of calculating the workload of the network and dynamically allocating the workload according to the network situation, taking into account the capacity of the servers. However, the various previous researches proposed are difficult to reflect the real world by imposing a lot of restrictions and assumptions on the IoT environment, and it is not enough to meet the wide range of service requirements for the IoT environment. Therefore, we proposed an agent that applies a deep reinforced learning method to distribute loads for gaming servers. The agent has accomplished this by measuring network loads and analyzing a large amount of user data. We specifically have chosen deep reinforcement learning because no labels would need to be obtained in advance and it enabled our agent to immediately make the right decisions to load balancing in IoT environment. We have showed several siginicicant functions of our proposed scheme and derived through mathematical analysis. Also, we have compared performances of our proposed scheme and a previus research, ProGreGA, widely used scheme through simulation.

Keywords: deep reinforcement learning, load balancing, gaming server, reward, achievable rate, loss rate, policy

1. Introduction

Recently, much study has been done applying load balancing to network [1]. Maintaining balanced workloads benefits the cloud service provider by increasing their resources utilization, eliminating the performance bottlenecks, and improving the quality of services to their customers. Load balancing schemes have been widely adopted by distributed servers and their effectiveness is of importance to the quality of services provided by such servers.

Load balancing has been studied using various approaches [2,3,4]. Centralized solutions are computationally extensive, require much information exchange overhead. To

* Co-corresponding Author: Jinsul Kim

overcome these limitations, decentralized approaches have been proposed in citeRef4, and a distributed algorithms are used to solve it. However, most of these studies impose many restrictions and assumptions the networks that often do not apply in realistic networks [5]. Reinforcement Learning (RL) is a learning algorithm of decision the action to be performed in the learning system for maximizing reward to the action [6,7]. Due to the RL methods' advantages, it has been largely discussed in developing load balancing problems. In [8], RL was implemented for distributed load balancing in IoT network.

Therefore, we proposed a load balancing scheme applying RL method in IoT network. We address the key functions for the proposed scheme and simulate its efficiency using mathematical analysis.

The rest of the paper is organized as follows. Section 2 gives the previous reseaches related to open load balancing and reinforcement learning. In Section 3, we describes the detailed load balancing scheme of ours. In section 4, we describe the experimental results and show that the proposed scheme can effectively improve the performance for gaming server in IoT environment. In the final section, we constitute a summary of our proposal and suggest further study directions in IoT environment.

2. Related Works

The contributions of IoT depend on the increased value of information created by the number of interconnections among things and the subsequent transformation of processed information into knowledge for the benefit of society. Various researches use a clustering algorithm to utilize contextual information. In [9,10], the authors proposed a subvariance method based on neural regulation filtering by applying context information clustering and latent function learning fuzzy theory. After they have investigated similar neighbors of users and similar neighbors of services. When the clustering result is ready to learn the latent function of contextual information, join the potential node to the cluster [11].

Load balancing mechanisms are widely used in a distributed computing environment to balance the workloads on different servers, and the effectiveness of such mechanisms is critical to the overall performance and service quality. Load balancing can distribute workload across multiple entities to achieve optimal utilization, maximize throughput, minimize response time, and avoid overload. A lot of research has been done on how to design an effective load balancing such as in [12,13,14].

Deep learning has been applied to a many fields such as speech recognition, computer vision, natural language processing, social network filtering and bioinformatics. Deep learning is also applied when adopting multiple layers of nonlinear processing units for feature extraction and transformation [15]. The effect on deep learning can be guaranteed to be a universal approximation theorem, since this theory can be represented as a small subset of continuous functions in a feedforward network with a single hidden layer containing a finite number of neurons [16,17,18].

In [19],the authors have proposed a method that modeled a new neighbor feature learning method as a matrix by combining the advantages of a neighbor-based method, a model-based method, and a method based on deep learning. The proposed mothod was able to achieve high accuracy in neighborhood selection even with high data scarcity, and was able to learn deep features. The learning systems they limit use a learning convolutional neural network to learn deep learning from the selected neighbor's cell record, and also learn the relationship with the features of the target user or target service.

The RL is a different from supervised learning in that it doesn't need input/output pairs. This focuses on performance, which involves finding the balance between explorations. This learning system creates $a_1, a_2...a_n$ actions to interact with the environment. These actions affect the environmental condition, and as a result, the RL system receives scalar rewards $r_1, r_2...r_n$. The goal of this learning system is to learn how to act in a way that maximizes future rewards through learning. The RL approaches store the results of interaction with the past environment and find the optimal policy for repetitive learning [20].

RL could be applied as a method for making optimal decisions. The agent for this has taken into account the environment. At every step, the agent has taken action and receives observations and rewards. RL algorithms has tried to take full account of a given, previously unknown environment. RL made choices to maximize rewards in each stage of learning, and learned the policy to find the maximum reward value by repeating the steps. They have been applied to many different fields. The policy optimization method used the policy of each step to map the agent's state to the next action and learns by reflecting the result value in the next step. These methods showed RL as a numerical optimization method. We could optimize the expected rewards for an efficient learning system in relation to the parameters of the policy.

The challenges herein are to consider a priori how many interactions are important to learn a specific task and what exact features should be extracted. Deep neural networks are the quintessential technique for automatic feature extraction in reinforcement learning [21,22]. Also, various previous researches in load balancing had not effectively taken into account the rapidly increased event and uncertainly status for gaming server in IoT network.

Therefore, we proposed a load balancing scheme applying reinforcement learning in order to efficient load balancing in IoT environment.

3. Proposed Scheme

3.1. System Configuration

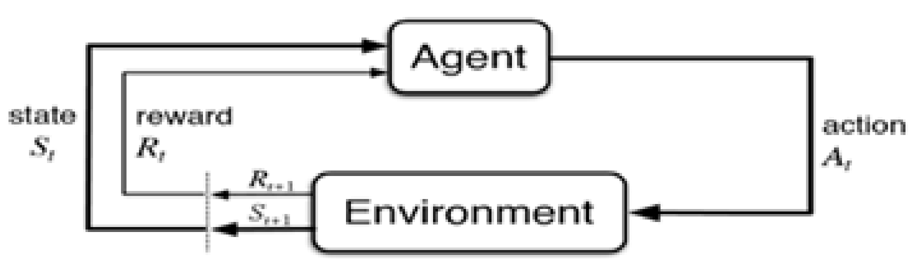


Fig. 1. Conceptual Diagram of Reinforcement Learning[26]

RL is one of a machine learning used to automate goal learning and decision making. Fig. 1 has been showned to the concept of RL method applied to the our scheme in this paper. When the agent in the proposed system received input, in the current state s, the agent performs the corresponding action a. As the result, the reward value r has provided. Based on the reward value of r, the learning system transmitted to the new state $s^{'}$ and the agent processed again as $a^{'}$. Depending on its current rewards and status, the reinforcement system has chosen the next action based on a policy that increases the likelihood of agent positive rewards. The goal of RL agents is to maximize the total rewards received from the proposed system to find the optimized policy.

3.2. Network Load Learning Algorithm

The RL algorithms are generally applied to obtain optimal results by adjusting the motions in observed state of discontinuous and low-dimensional motions [23]. However, along with the development of computing capacity and deep learning, a new algorithm called Deep Reinforcement Learning (DRL) has appeared. In order to model complex nonlinear relationships such as IoT networks, we have applied RL to map the structure of the network load. In IoT networks, load balancing is only achieved in a small local area, so the network area is divided into several smaller areas based on the zone. Our proposed load balancing scheme is by system configuration applying DRL algorithm for distributed static load balancing of gaming server. For the mathematical modeling of our proposed scheme, we have used the following algorithm. The Variables used to model our proposed scheme have shown in Table 1.

As each base station generally have been served a large number of user nodes, the

Table 1. parameters of modeling

Parameters	Descriptions
B	base station
N	nodes of user
P_j	power of base station, j
σ	noise power
t_0	start time
r_j	reward of j
Z	zone area
w	weight of the node or server or cell)
S_j	state of j node in IoT
π	map states
β	control value for learning
λ	learning rate
A	action space

important metric for network performance has the rate of service speed, not Signal to Interference Plus Noise Ratio (SINR) [24,25]. The rate of service speed being experienced by the user node depends on the network load. We have defined the service speed rate as R_{ij} and the achievable rate as c_{jj} for independent of channel qualities. Let B be the set of base stations with more than one node in network and that share resources and N be the node of users. Let P_j be a power of base station j and σ^2 be a noise power level and G_{ij} be a channel gain between i node and j base station. Therefore, $\sum_{k \in B, k=j} P_k G_{ik}$ has shown the interferences in the network. Let t_0 be a start time and t as a present time and $\tau(t_0 \leq \tau \leq t)$ as a variable respresenting time, respectively. Let $F_{ij}(t)$ be a fraction time of resource that is the base station j servers node i and H_{ij} be a long term service rate. Let $x_{ij}\tau$ be a scheduling indicator.

$$c_{ij} = \log_2(1 + SINR_{ij} = \log_2(1 + \frac{P_j G_{ij}}{\sum_{k \in B, k=j} P_k G_{ik} + \sigma^2}) \tag{1}$$

$$H_{ij}(t) = F_{ij}(t) \int_{t_0}^{t} x_{ij}(\tau) C_{ij}(\tau) d(\tau) \tag{2}$$

For efficient load balancing, the larger the value of $\sum_{j \in B} \sum_{i \in N} H_{ij}(t)$ and the smaller the variance of user's service rate $H_{ij}(t)$ should be. We have derived the results by constructing an estimator of our system using importance sampling in a large and continuous state. Our scheme have allowed to make decisions for each process according to a

Algorithm 1 Load Balancing Agent applying Reinforcement Learning

```
for i = 1, 2,... S do do
    for j = 1, 2, ... N do do
        choose the best decision value, argmax_j D_ij(t)
        calculate its current reward, r_j(t) based on 1
        calculate long term average reward of set S_i^j, j is different set of actions
    end for
    set allocation of servers for optimize load balancing
end for
```

learned policy without wasting time for complex calculations. It is also possible to determine the optimal action based on the reward value of each phase without accurate information on the reward value or probability value of all environments. The agent of ours has learned to output the desired result value by input using present input and output data sets. Our network load learning algorithm has calculated and stored the current network load and learned the result value effectively.

Let r_j be denote the reward of a phase j, S be indicate the status, and A be denote the behavior. Therefore, the S_j is the phase in j where j is the station. Given each phase, s , it mapes directly to the determined action a. Every $a \in A(s)$ has a probability distribution or could be deterministic $\pi(s)$. The policy for load balancing, that is the action determined by the state s or stochastic $pi(a \mid s)$. In order to achieve efficient load balancing, RL has applied in our system, and a pi policy has developed to select possible behaviors in each phase and map behavior to state that improve pi to be optimal. Load balancing policies could be either stochastic $\pi(a \mid s)$, which given a state s, each action $a \in A(s)$ is a probability distribution, or deterministic $\pi(s)$, that has maped a state, s, to a determined action, a. We have calculated the reward of base station j as r_j.

$$r_j = \frac{1}{\sum_i^{S_j} \frac{1}{s_j} \cdot (R_{ij} - \frac{\sum_{k=1}^{|B|} \sum_{i=1}^{|V|} R_{ik}}{|V|})^2} \tag{3}$$

Let $AL_{ij}(t)$ be the allocation priority of node i at the base station j and C_{ij} is broadcast at achievable rate.

$$AL_{ij}(t) = \frac{C_{ij}(t)}{R_{ij}(t-1)} \tag{4}$$

We have pi to represent the probabilistic policy π: S X $A \longleftarrow$ [0, 1], and the expected discount compensation for η (π) to indicate. We have considered the policy $\pi_\theta(a|s)$ with parameter vector θ, and we have used function of θ rather than $overloaded.\pi$, $L_\theta(\tilde{\theta})$: = $L_\pi(\pi_{\tilde{\theta}})$ and D_{KL} $(\theta \parallel \tilde{\theta})$: = D_{KL} $(\pi_\theta \parallel \pi_{\tilde{\theta}})$. We used θ_{old} as a parameter to improve the previous policy in our scheme. We have sampled $s_0 \sim \rho_0$ and simulated the π_{θ_i} policy. Then, following these trajectories of $s1, s2, ..., s_m$, have selected a subset of N states. The agent $\hat{Q}_{\theta_i}$ $(s_m,\, a_{m,k})$ for each of the action have sampled from each the status s_m and $a_{m,k}$ and processed actions in that phase.

$$L_m\theta = \sum_{k=1}^{k} \pi_\theta(a_k \mid s_m)\hat{Q}(s_m, a_k) \tag{5}$$

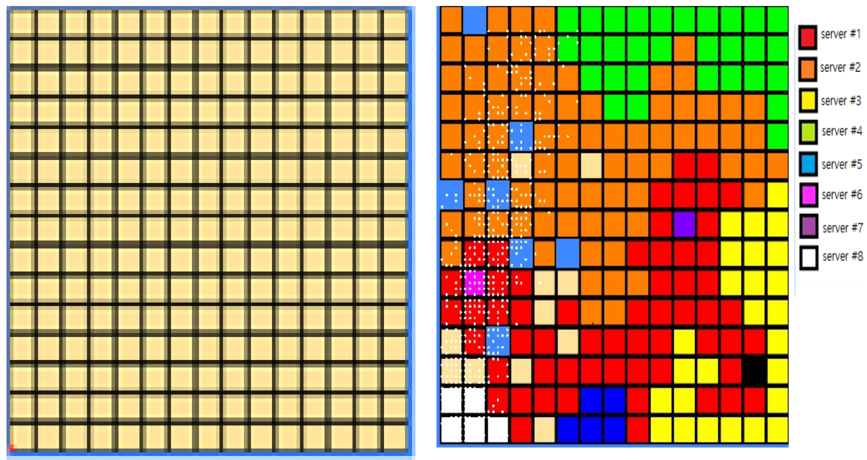
We have generated for every possible action in the state of that phase given in each of the phase. In our scheme, the agent have processed the $a_{n,k}$action as K behavior in each phase state, s_n, as represented $a_{n,1}, a_{n,2}, ..., a_{n,k}$. The results obtained in each pahse have represented as $L_{\theta_{old}}$. We have estimated $L_{\theta_{old}}$ from the expectation and gradient of $s_n \sim \rho(\pi)$ for $L_{\theta_{old}}$.

$$L_m(\theta) = \frac{\sum_{k=1}^{k} \frac{\pi_\theta(a_{m,k}|s_m)}{\pi_{\theta_{old}(a_{m,k}|s_m)}} \hat{Q}(s_m, a_{m,k})}{\sum_{k=1}^{k} \frac{\pi_\theta(a_{m,k}|s_m)}{\pi_{\theta_{old}(a_{m,k}|s_m)}}} \tag{6}$$

4. Performance Analysis

4.1. Simulation

In this study, we have used ML-agents library from Unity3D to simulate the load balancing agent for gaming server applying our proposed scheme at Section 3. The learning is processed by the repeated cycle of sending variables by Tensor-flow which are collected from learning environments created by Unity3D and sending back results which are learned from Proximal Policy Optimization (PPO) algorithms, one of the RL. We have set the hidden layer of neural network to 3 and the node of a hidden layer to 256 as refered in [27,28]. Also, we have set the size of batch to 512 and the β value to control the entropy to $\log_e^{-3}$. In addition, we have set the learning step to 5 million steps for our simulation environment.



(a) Simulation environment (b) Initial State

Fig. 2. Initial Simulation environment

It consists of a two-dimensional map of the game world, each of that contained a Finite State Machine(FSM) in the place of 750 gaming users. The weight of each bot is reset to 1. We have assumed that there are 8 servers and used policy that load balancing agent learned, to disperse the load. Fig.2(a) is shown our environment of simulation.

For analysis of load balancing result, we have differentiated each server by color as shown in Fig. 2(b). Each rectangle represents a cell, and a group of areas is defined as an area, and a group of the areas is defined as a world. Each server is given an area.

The game world is made up of 15*15 grid world and has 225 cells which are allocated with 8 servers. User load is occurred by activating 750 users, all which are processed by FSM. Also, server capacity is defined as i*20000 and i is a value between 1 to 8. Therefore, we have assumed the capacity of a server 1, 2, and 3 as 20000, 40000, and 160000. Fig. 2(b) is shown the initial state.

4.2. Performance Analysis

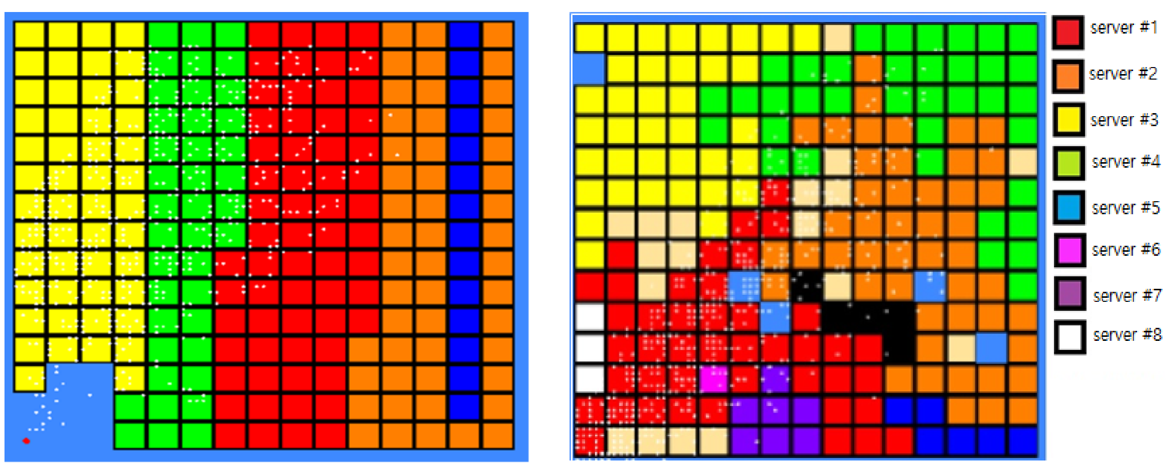
In the experiment, a standard that we have set are as follow: 1) The weight of 750 users has been set as 1
2) After defining the weight, we set a number of users between 500 to 1000
3) After defining the number of users, we multiply the weight by a value between 0.5 to 2

It was showed in [28] that the ProGreGA algorithm has numerous advantages compared to other load-balancing algorithms such as BFBCT, Kernighan-Lin, and Ahmed such as fewer walk migrations, minimized overhead, and the maintenance of the maximum possible number of cells when rebalancing, resulting in the ProGReGA algorithm having the most efficient all the simulated algorithms.

Algorithm 2 ProGReGA Load Balancing Algorithm

initiate $Weight_{Division}, Capacity$
for each zone z in zone list Z do **do**
    $Weight_Division = Weight_{Division} + W_z(Z)$
    $Capacity = Capacity + Y(S_Z)$
end for
sort zone list in decreasing $Y(S_Z)$
for each zone z in zone list Z do **do**
    $Weight = Weight_{Division} \times \frac{Capacity}{Weight_{Division}}$
    while $W_z(Z) \leq Weight$ do **do**
        if any cell from Z
        Z= $Z \cup$ with heihest cell in the zone
        else
        $Z \cup$ the cell
    end while
end for

Therefore we will compare our proposed scheme only with the ProGReGA algorithm. The Alorithm 2 is shown the ProGReGA methods as in [28], and we have experiment based on 2 to simulate the ProGReGA.



(a) Proposed Scheme (b) Previous(ProGReGA) Scheme

Fig. 3. Result of load balancing in simulation environment

In the propose scheme, the average fragmented cells occurred is 2 or 3 times more than that in the ProGReGA. A result of 100 experiments is sorted in ascending order of fragmented cells and divided it in three to indicate which are the worst, average, and best results.

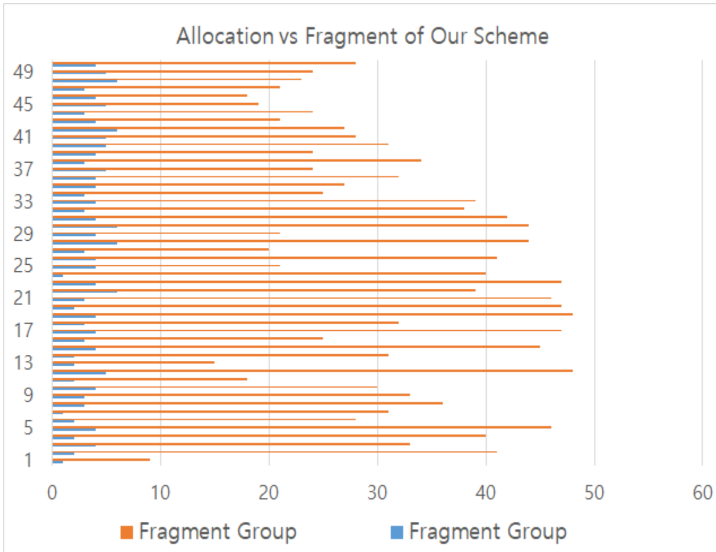
We have put a criterion that user one load interacts with all users, and to confirm the result of balancing, we have dispersed the load in a way that is not affected by the previous load balancing result. To check the process of dispersion, we have set a color for each server and alter the color of the dispersed cell to the corresponding server color. We have selected a scenario that where each user randomly selects bearing and move to that direction.

When an agent chooses a cell that hasn't been selected before, it gets +1 as a reward, otherwise, it gets -0.5 as a reward. In addition, if it selects a cell which is not near the cell it is currently located, it gets -0.1 as a reward. Thus, when one episode ends, we add up the reward it got and additionally add it with the value of the standard deviation of dispersed server usage, multiply with 10. It means that as usage standard deviation decrease, distribution of usage is constant, i.e., usage of all server is distributed equally

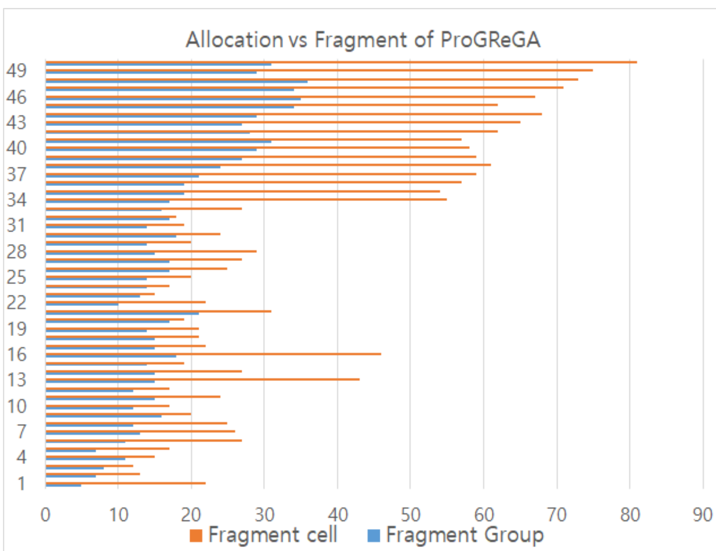
The Fig.3(a) and Fig.3(b) are shown the result of the load balancing of our proposed scheme and previous scheme, respectively.

The experimental result of a model that has been learned through our scheme shows 75% increase in its performance compared with the previous research, ProGreGA. The occurrence aspect of the conditional fragmented cell are shown no much difference from ours. However, it could be realized that the occurrence rate for the fragmented cell is too much and change in the occurrence rate changed drastically.

As shown in Fig. 4, our proposed scheme have showed better performance than previous research, ProGreGA. The conditional fragment cell of model that has been learned through the same condition is shown as a graph in Fig. 5.

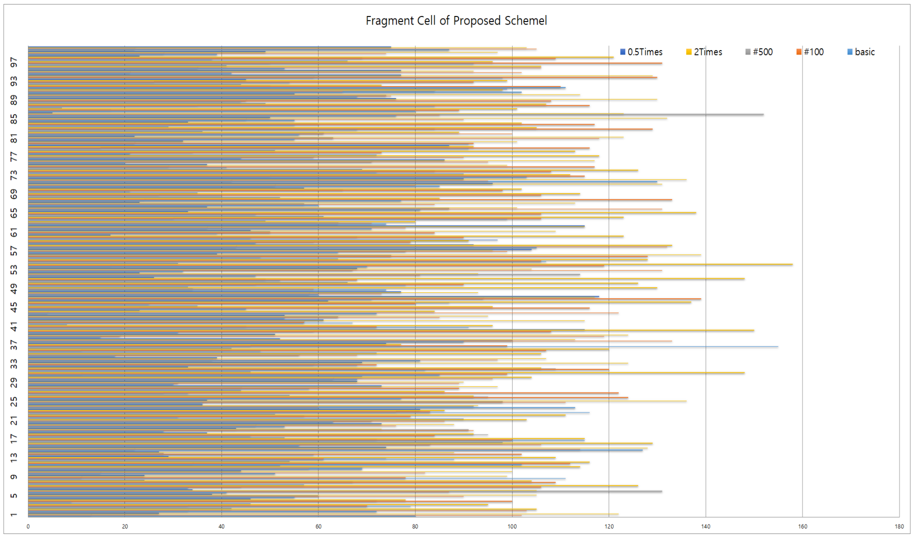


(a) Proposed Scheme

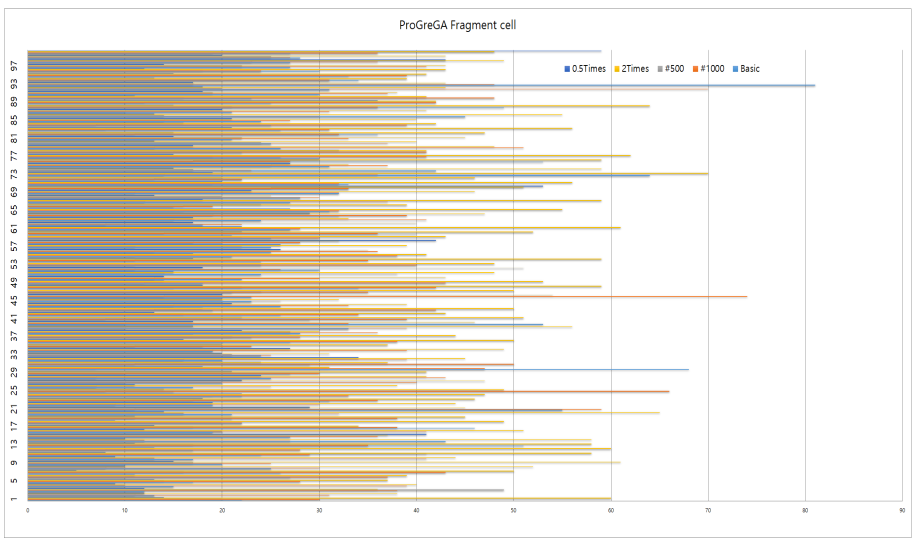


(b) Previous(ProGReGA) Scheme

Fig. 4. Result of allocation and fragement cells of server



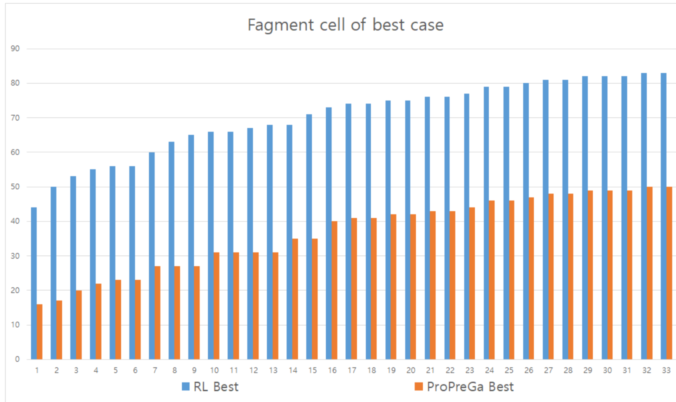
(a) Fragment Cell of Proposed Schemel



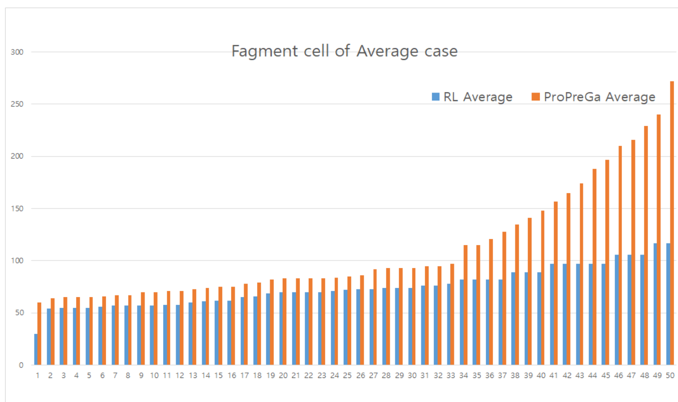
(b) ProGreGA Fragment cell

Fig. 5. Fragement cells after Load balancing

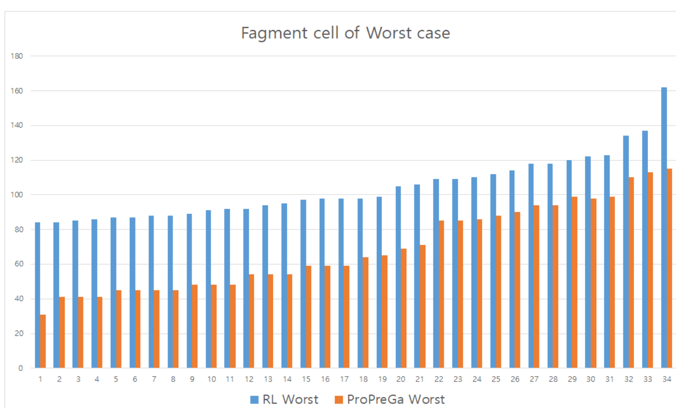
Also, our proposed scheme have shown no significant change in the occurrence rate in case of average and worst at the simulation as shown Fig.6(b) and Fig. 6(c). As shown the Fig.6(a), our proposed scheme have shown no rapid change in the occurrence rate just like in average result.



(a) best fragment cell



(b) average fragment cell



(c) worst fragment cell

Fig. 6. Result of fragment cell after load balancing each of gaming server

We have simulated 100 experiments on load balancing under various conditions, and the results have been shown in Fig.7. The more fragmented cells that are not allocated to servers for load balancing, the lower the performance. The reason why the number of occurrences of the fragmented group is important because servers are assigned based on the group unit.

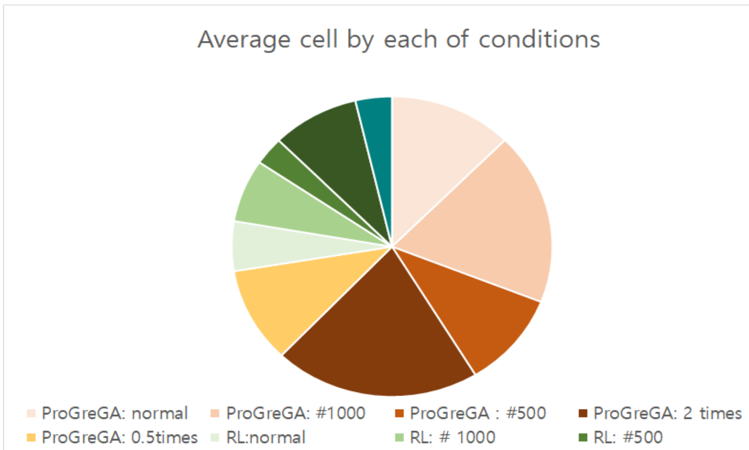


Fig. 7. Result of 100 executin fragment cell by conditions

5. Conclusion and future works

In this study, we have proposed an agent that applies a deep reinforced learning method to distribute static loads for gaming servers. We have addressed several key functions of our proposed scheme and derived the efficiency of ours through mathematical analysis. The agent has been accomplished this by measuring network loads and analyzed the large amount of user data and network loads, all with the aforementioned DRL.

We have used ML-agents library from Unity3D to simulate the load balancing agent for gaming server applying our proposed scheme. The learning was processed by the repeated cycle of sending variables by Tensor-flow collected from learning environments created by Unity3D and sending back results that are learned from Proximal Policy Optimization (PPO) algorithms, one of the reinforced learning. We have simulated 100 experiments on load balancing under various conditions, where, the more fragmented cells not allocated to servers for load balancing, the lower the performance. The number of occurrences of the fragmented group is important because servers are assigned based on the group unit. We compared the performance of the ProGreGA algorithm which was shown to be the most efficient among the previous research as in [28] with our proposed scheme by running mathematical modeling and simulations. The simulation result of a model learned through our scheme has been shown 75% increase in its performance compared with the ProGreGA. The occurrence aspect of the conditional fragmented cell has been shown no much difference from ours. However, the occurrence rate for the fragmented cell was too much and the occurrence rate changed drastically. Our proposed scheme have shown the efficiency of load balancing and it is required further works reflect real world in network.

In the future, we intend to evaluate performances by collecting data from applying the proposed scheme in the real world, such as in game servers and blockchain platforms. In addition, we would analyze the collected data and analyze performance through various deep learning algorithms.

Acknowledgments. This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. 2019R1A2C1008533) and this research was supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Science, ICT and Future Planning(No. 2016RIA2B4012386) and this work was supported by 2020 Hongik University Research Fund.

References

1. Ahmed D., Shirmohammadi S.: A microcell oriented load balancing model for collaborative virtual environments. In: Proceeding of the IEEE International Conference on Virtual Environments, Human Computer Interfaces and Measurement Systems, VECIMS, pp.86-91 (2008)
2. Andrews J., S. Singh, Q., Lin X.,Dhillon H.: An overview of load balancing in HetNets: Old myths and open problems. In: IEEE Wireless Com-mun., 21(2), 18–25, Apr. (2014)
3. O. K. Tonguz, Yanmaz E.: The mathematical theory of dynamic load balancing in cellular networks. In: IEEE Trans. Mobile Comput. 7(12), 1504–1518, Dec (2008)
4. H. Kim, Veciana G., Yang X., Venkatachalam M.: Distributed alpha-optimal user association and cell load balancing in wireless net- works. In: IEEE/ACM Trans. Netw., 20(1), 177–190, Feb. (2012)
5. Zhang Z., Ma L.,, Leung K., Tassiulas L., Tucker J.: Q-placement: Reinforce-ment-Learning-Based Service Placement in Software-Defined Networks. In: Proceeding IEEE International Conference on Distributed Computing Syst. (ICDCS), 1527-1532, July (2018)
6. Sutton R., Barto A.: Reinforcement Learning: An Introduction. In: Cambridge, MA, USA: MIT Press, (1998)
7. Honghao G., Kuang L., Yin Y., Guo B., Dou K,: Mining Consuming Behaviors with Temporal Evolution for Personalized Recommendation in Mobile Marketing Apps. In: ACM/Springer Mobile Networks and Applications (MONET), 2020, DOI: 10.1007/s11036-020-01535-1
8. Parent J., Verbeeck K., Lemeire J.: Adaptive Load Balancing of Parallel Applica-tions with Reinforcement Learning on Heterogeneous Networks. In: Interational Symposeum Distributed Computing and Application for Bussiness Enginering and Science, 16-20 Dec. (2002)
9. Honghao G.,Yueshen X., Yuyu Y., Weipeng Z., Rui L., Xinheng W.: Context-aware QoS Prediction with Neural Collaborative Filtering for Internet-of-Things Ser-vices. In: IEEE Internet of Things Journal, (2019), https://doi.org/10.1109/JIOT.2019.2956827.
10. Xiaoxian Y., Sijing Z., Min C.: An Approach to Alleviate the Sparsity Problem of Hybrid Collaborative Filtering Based Recommendations: The Product-Attribute Perspective from User Reviews. In: MOBILE NETWORKS and APPLICATIONS, 25(2), 376-390 (2020)
11. Honghao G., Wanqiu H., Yucong D.: The Cloud-Edge Based Dynamic Reconfiguration to Service Workflow for Mobile Ecommerce Environments: A QoS Prediction Perspective. In: ACM Transactions on Internet Technology (2020), DOI: 10.1145/3391198

12. Hye-Young K. : A Load Balancing Scheme with LoadBot in IoT Networks. In: Journal of Supercomputing, 74(1), 1215-1226, July (2017), DOI 10.1007/s11227-017-2087-6.
13. Deng S., Xiang Z., Zhao P., Taheri J., Hinghao G., Yin J., Zomaya Y,: Dynamical resource allocation in edge for trustable iot systems: a reinforcement learning method. In: IEEE Transactions on Industrial Informatics, (2020), DOI: 10.1109/TII.2020.2974875
14. Honghao G., Liu C., Youhuizi L., Xiaoxian Y.: V2VR: Reliable Hybrid-Network-Oriented V2V Data Transmission and Routing Considering RSUs and Connectivity Probability. In: IEEE Transactions on Intelligent Transportation Systems, (2020), DOI: 10.1109/TITS.2020.2983835
15. Jinglin L., Guiyang L., Nan C., Quan Y., Zhiheng W., Shang G.,Zhihan L.: An End-to-End load Balancer Based on Deep Learning for Vehicular Network Traffic Control. In: IEEE Internet of Things Journal, 6(1), pp.953-966, February (2019)
16. Cybenko G.: Approximation by superposition of a sigmoidal function. In: Math. Control Signals Syst., 2(4), 303–314 (1989)
17. Jun Y., Chaoqun H., Yong R., Dacheng T.: Multitask Autoencoder Model for Recovering Human Poses. In: IEEE Transactions on Industrial Electronics, 65(6), 5060-5068 (2018)
18. Jun Y., Zhenzhong K., Baopeng Z., Wei Z., Dan L., Jianping F.: Lever-aging Content Sensitiveness and User Trustworthiness to Recommend Fine-Grained Privacy Settings for Social Image Sharing. In: IEEE Transactions on Information Forensics and Security, 13(5), 1317 – 1332 (2018)
19. Yuyu Y., Lu C., Yueshen X., Jian W., He Z., Zhida M.: QoS Prediction for Service Recommendation with Deep Feature Learning in Edge Computing Environ-ment. In: Mobile Networks and Applications, (2019), https://doi.org/10.1007/s11036-019-01241-7.
20. Revar A., Andhariya M., Sutariya D.: Load Balancing in Grid Environment using Machine Learning-Innovative Approach. In: International Journal of Computer Applica-tions, 8(10), 31-34, October (2010)
21. Sutton R., Barto A.G.: Reinforcement Learning: An Introduction. In: Cambridge, MA: MIT Press (1998)
22. Bertsekas D.P.,Tsitsiklis J.N.: Neuro-Dynamic Programming. In: Nashua, NH, Athena Scientific (1996)
23. Goodfellow I., Bengio Y., Courville A.: Deep Learning [Online]. In: MIT Press (2016), http://www.deeplearningbook.org
24. Cano G.L., Ferreira M., Simoes A.S., Luna C.: Intelligent Control of a Quadrotor with Proximal policy Optimization Reinforcement Learning. In: Lat-in America Robotic Symposium, pp.503-508 (2018)
25. Ye Q., Rong B., Chen Y., Al-Shalash M., Caramanis C., Andrews J.: User as-sociation for load balancing in heterogeneous cellular networks. In: IEEE Transactions Wireless Communications, 12(6), 2706–2716, Jun (2013)
26. Richard S.: Reinforcement Learning:An Introduction. In: Cambridge, P.48 (2018)
27. Andrews J., Singh s., Ye Q., Lin A., Dhillon H.: An overview of load balancing in HetNets: Old myths and open problems. In: IEEE Wireless Communications, 21(2), 18–25, Apr. (2014)
28. Carlos E., Cláudio G.: A Load Balancing Scheme for massively multiplayer online games. In: Multimedia Tools and Applications, 45(1), 263-289, October (2009)

Hye-Young Kim received the Ph.D. degree in Computer Science and Engineering from the Korea University of South Korea in February 2005. During her Ph.D. study, she had focusing on location management scheme and traffic modeling, such as mobile IPv6, cellular network and network mobility. Currently, she works at Hongik University of South

Korea as a Full Professor since March 2007. She had developed a network protocol for 9 years while she was working at Hyundai Electronics as a senior researcher. Her research interests include traffic modeling and load balancing scheme applying deep learning in IoT and Block-Chain.

Jin-Sul Kim received the B.S. degree in computer science from The University of Utah, Salt Lake City, UT, USA, in 2001, and the M.S. and Ph.D. degrees in digital media engineering from the Department of Information and Communications, Korea Advanced Institute of Science and Technology (KAIST), Daejeon, South Korea, in 2005 and 2008, respectively. He worked as a Researcher with the IPTV Infrastructure Technology Research Laboratory, Broadcasting/ Telecommunications Convergence Research Division, Electronics and Telecommunications Research Institute (ETRI), Daejeon, from 2005 to 2008. He worked as a Professor with the Korea Nazarene University, Chonan, South Korea, from 2009 to 2011. He is currently a Professor with Chonnam National University, Gwangju, South Korea. His research interests include cloud computing, MEC, smart factory/city based Intelligent application development, and intelligent networking solutions.

Received: September 19, 2019; Accepted: May 8, 2020.