J Inf Process Syst, Vol.17, No.2, pp.297~305, April 2021
https://doi.org/10.3745/JIPS.03.0158

ISSN 1976-913X (Print)
ISSN 2092-805X (Electronic)

JOURNAL OF INFORMATION PROCESSING SYSTEMS JIPS

An Efficient Load Balancing Scheme for Gaming Server Using Proximal Policy Optimization Algorithm

Hye-Young Kim*

Abstract
Large amount of data is being generated in gaming servers due to the increase in the number of users and the variety of game services being provided. In particular, load balancing schemes for gaming servers are crucial consideration. The existing literature proposes algorithms that distribute loads in servers by mostly concentrating on load balancing and cooperative offloading. However, many proposed schemes impose heavy restrictions and assumptions, and such a limited service classification method is not enough to satisfy the wide range of service requirements. We propose a load balancing agent that combines the dynamic allocation programming method, a type of greedy algorithm, and proximal policy optimization, a reinforcement learning. Also, we compare performances of our proposed scheme and those of a scheme from previous literature, ProGreGA, by running a simulation.

Keywords
Dynamic Allocation, Greedy Algorithm, Load Balancing, Proximal Policy Optimization, Reinforcement Learning

1. Introduction

Load balancing has been researched according to various methods of allocating loads such as dynamic allocation, static allocation, user-oriented allocation and applying deep learning method. An overview can be found in [1,2]. The centralized methods not available to be extended as it requires and enormous information exchange overhead and are thus not scalable. To overcome these limitations, decentralized methods have been proposed. Load balancing is modeled as a mathematical optimization problem in [3] and a distributed algorithm is proposed to solve it. It showed that this makes the strong resource a lot of load and the weak resource a lot of idle resources. However, most previous optimization researches such as gradient descent method and Lagrange multiplier method only apply to the traffic flow generated by static allocation. The authors assume the channel quality is stable. Nevertheless, the lack of feedback from environment causes the gradient descent to lose the direction. However, most of these researches impose many limitations and assumptions over the networks, which often do not relate with realistic in networks [4].

Therefore, we propose a effectual scheme combing greedy algorithm of dynamic allocation method with proximal policy optimization (PPO) which is among the reinforcement learning method in order to

Manuscript received March 26, 2020; first revision September 15, 2020; accepted October 4, 2020.
Corresponding Author: Hye-Young Kim (hykim@hongik.ac.kr)
* Dept. of Game Software, School of Games, Hongik University, Seoul, Korea (hykim@hongik.ac.kr)

achieve efficient load balancing in network. We used ml-agents library from Unity3D in order to proceed on reinforced learning. The learning is processed by the repeated cycle of sending variables by TensorFlow which are collected from learning environments created by Unity3D and sending back results which are learned from PPO algorithms, one of the reinforced learning.

The rest of the paper is structured as follows. Section 2 describes the background involving the load balancing schemes and reinforcement learning. In Section 3, we describe our proposed architecture and provide the detailed a load balancing scheme. In the final section, we constitute a summary of our proposal and suggest a further study direction for load balancing in massively multiplayer online gaming servers in networks.

2. Related Works

Load balancing mechanisms are widely used in distributed computing environments to balance the workload of multiple servers, and the effectiveness of these mechanisms is critical to the overall performance and quality of service of the distributed platform. Workloads across multiple entities, maximizing throughput, minimizing response time, achieving optimal performance and avoiding overload. There has been a lot of research on load balancing and how to design an effective load balancing mechanism attracted the attention of many researchers. Recently, load balancing has been applied in various ways to various fields [5]. For example, in [6], multipath is formed between the sink and the source through path dispersion by applying deep learning, and data is spread and distributed along different paths according to various energy-related parameters to reduce the load of different nodes. However, even with the multipath approach in this method, it is still possible that multiple individual routes are merged at a specific point in the network, resulting in insufficient routes and reduced efficiency.

In the individual path merging technique, nodes in the merged path consume more energy than other individual paths. As a result, the network is out of balance and goes down early. To solve this problem, [7] suggested that inactive nodes participate in the routing path with a local greed algorithm. In another study, the authors proposed a multi-path routing scheme that generates multi-paths by applying a deep reinforcement algorithm to distribute the workload of network traffic.

Therefore, these studies have made it possible to select remote nodes even if a more energy-efficient route to the source exists. However, the network was initially down because the node consumed more energy along a longer path. In order to solve this, the node was selected by applying reinforcement learning instead of selecting the node based on the distance from the sender in [8].

In server-client architecture, server nodes can easily be overloaded by user's high demand for status updates. Many studies suggest algorithms to distribute the load across server nodes, but this load is usually defined by the number of users on each server and is not an ideal measure. Also, the possible heterogeneity of the system is often overlooked.

As for the load shedding method, the overloaded game server tries to cut off the load and it tries to redistribute the load [9]. After finding a low-load game server, it sends some boundary microcells to that low-load game server. For the next timestamp, the next event is mapped to another grid cell to do load balancing. The concept of node contribution potential is used to avoid situations where events are mapped to locations where there are not enough resources to service the network on the surrounding nodes [10-12]. Deep learning is applied in a variety of fields, including computer vision, speech recognition, audio recognition, social network filtering, machine translation, and bioinformatics, and can be compared to humans, and in some cases better than humans producing results. For the detection approach, the authors were compared to each other according to the importance factor of [13]. Their pros and cons were discussed in terms of data mining models, evaluation methods and proficiency. In addition, for privacy protection, the authors [14] combine logistic regression with local differential privacy protection in combination with machine learning to perform classification using noise addition and feature selection.

Deep learning uses multi-layered nonlinear processing units to cascade for feature extraction and transformation [15]. The effectiveness of deep learning can be ensured by a universal approximation theorem, which is a small sub-layer of Rn under light assumptions about the activation function of the feed-forward network with a single hidden layer containing a finite number of neurons. It is a continuous state in the set [16]. Meanwhile, the reinforcement learning algorithm stores the sequence of past interactions with the past environment in memory and extracts related functions based on this memory. Also, making a series of decisions is an area of machine learning. Agents consider agents located in the environment. At every step, the agent acts, receives observations and rewards. The reinforcement learning algorithm tries to maximize the agent's total reward in a previously unknown environment through a trial and error learning process. Regarding the reward maximizing agent, the reinforcement learning problem described above is very common and the algorithm has been applied to various other fields. The policy optimization method revolves around policies, the ability to map the agent's state to the next task.

These methods consider reinforcement learning as a numerical optimization problem that optimizes the expected rewards in relation to the parameters of the policy. AlphaGo, who defeated the human world champion in Go [17], was similar to IBM's Watson DeepQA system, winning the best Weeford! Player [18]. Unlike the homemade rules that dominated the chess game system, AlphaGo consisted of trained neural networks and traditional heuristic search algorithms using coaching and reinforcement learning.

In previous studies [19,20] considered the workload of interactions to learn specific tasks and the exact functions to extract, and deep neural networks or multi-layer perception applied typical techniques for automatic feature extraction in reinforcement learning.

An important innovation in automatic feature extraction using deep learning is presented in [21], and to successfully play the Atari game, the authors apply a convolutional neural network to automatically extract relevant features based on visual input data. Unfortunately, the most widely used optimization algorithms such as gradient descent and Lagrange multiplier methods only apply to scenarios where traffic flow generated by mobile users is almost static. They assume that the channel quality is stable in theirs research.

However, in a real-world environment, traffic changes are not stable across the network. In addition, if the network scenario changes, the existing connection algorithm must be process at high cost across the entire network. Also, in general, most previous studies on load balancing and distributed decision-making did not effectively consider the events and uncertainties of fast-growing game servers.

3. Our Proposed and Performance Analysis

The agent controlled by the reinforcement learning algorithm observes the state, s, from the learning system at time step *t*. The agent interacts with the learning system by processing actions in the s_t. When the agent processes an action, the agent transition to a new state $s_{t=1}$ depending on the current state and the selected action. In this paper, we denoted actions and are a_t and x_t, respectively.

Whenever the learning system transitions from t to a new state $t+1$, the scalar reward r_{t+1} is given as feedback to the agent. In this case, the best phase of action is determined by the rewards provided by the learning system. The agent's goal is to learn policy π, which maximizes the expected return. Given a status, the policy returns what to do. Optimal policy is any policy that maximizes the expected return in the learning system. The policy retrieval method does not need to maintain a value function model, but directly retrieves the optimal policy π^*. In general, the expected yield $E[R|\theta]$, the policy π_θ with parameters updated is selected [22]. Gradient free optimization can effectively display a low-dimensional parametric space and has succeeded in applying it to large networks, but gradient based learning is one of the selection method of reinforcement learning algorithms, and this policy requires many parameters to find the best policy. Gradients can provide powerful learning parameters for how to improve our policy.

Let AD_t is an assessor of the advantage function at timestep t. The expectation $E_{exp}[...]$ denote the empirical average over a finite of samples. Implementations that use automatic differentiation an objective function whose gradient is the policy gradient estimator, the estimator G is obtained by differentiating the Eq. (1). For computing variance reduced advantage function make use a learned state value function $V(s)$. We use a loss function that combines the term error of policy and value function. This purpose can be augmented by adding an entropy bonus to ensure sufficient exploration. c_1, c_2 are coefficients, and S denotes an entropy bonus. It is show to perform multiple steps of optimization on this loss L_{policy} using the same way.

$$\mathrm{G} = \mathrm{E}_{exp}[\nabla_\emptyset \log \pi_\emptyset(a_t|\, s_t)\, AD_t \tag{1}$$

$$L_{policy}(\emptyset) = E_{exp}[\log \pi_\emptyset\,(a_t\,|s_t) AD_t \tag{2}$$

We use an actor-critic style which is one of the PPO algorithms for policy learning as follows:

```
Algorithm 1:
For I = 1, 2, 3,… M;
      For a = 1, 2, 3,…. N;
            $\pi\ \theta$ for T
            average $A_1$, $A_2$, … $A_T$
      EndFor
EndFor
```

Let $r_t(\emptyset)$ denote the probability ratio $r_t(\emptyset) = \frac{\pi_\theta(a_t|s_t)}{\pi\theta_{old}(a_t|s_t)}|$, where t specifies the time index in [0, T], within a given length T trajectory segment. We can use a truncated version of generalized advantage estimation, λ = 1.
Also, we can obtain the Eqs. (3) and (4) from above.

$$AD_t = -v(s_t) + r_t + \gamma r_{t+1} + \gamma^{T-t+1} r_{t-1} + s\gamma^{T-t} V(s_T) \tag{3}$$

$$AD_t = \delta_t + (\lambda r)\delta_t \quad + \gamma V(s_{t+1)} - V(s_t) \tag{4}$$

In this paper, we used ML-agents library from Unity3D to simulate the load balancing agent for massively multiplayer online game (MMOG). It consists of a game world which consists of two-dimensional map and each game world contains finite-state machine (FSM) bot in the place of 750 game users. The weight of each bot is reset to 1. We assume that there are 8 servers and used policy which load balancing agent learned, to disperse the load. We show our simulation environment in Fig. 1 [23].

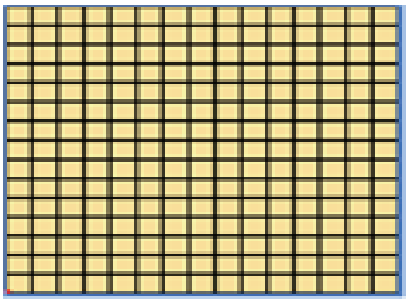


Fig. 1. Simulation environment. Adapted from Park et al. [23].

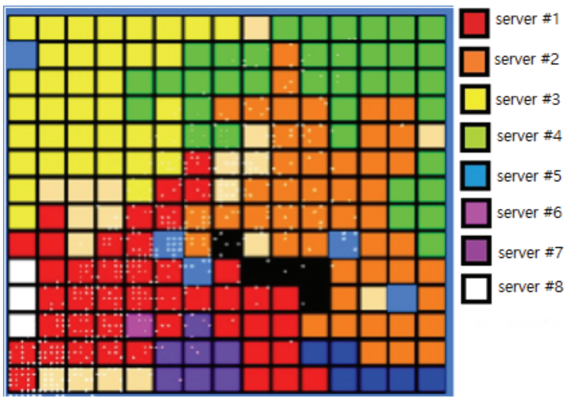


Fig. 2. Initial load balancing. Adapted from Park et al. [23].

In Fig. 2, we differentiate each server by color to analysis of load balancing result.

Each rectangle represents a cell, and a group of cells is defined as an area, and a group of the area is defined as a world. Each server is given an area. The game world is made up of 15 × 15 grid and has 225 cells which are allocated with 8 servers. User load is occurred by activating 750 user AI which is processed by FSM. Also, server capacity is defined as $i \times 2000$ and i is a value between 1 to 8. Thus, we assume the capacity of a sever 1, 2, and 3 as 20000, 40000, and 160000, respectively.

These indicators occur in the process of dispersion. The less the indicator shows, the resources have been dispersed without any waste. Also, less change of indicator implies performance is constant. In the experiment, the standard that we set are as follows:

(i) The weight of 750 users has been set as 1;

(ii) After defining the weight, we set a number of users between 500 to 1000;

(iii) After defining the number of users, we multiply the weight by a value between 0.5 to 2.

Fig. 3 shows the flowchart of our simulation used in this paper.

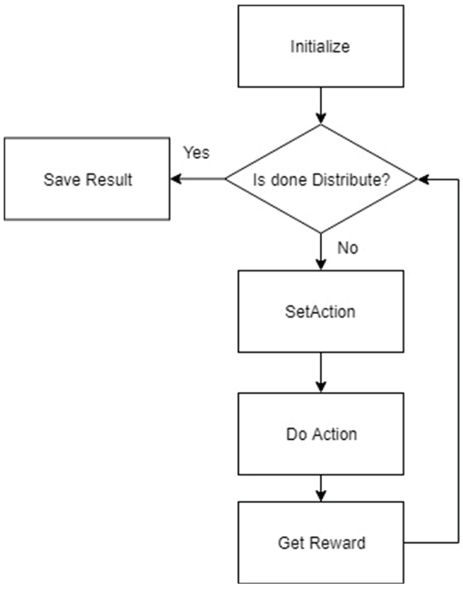
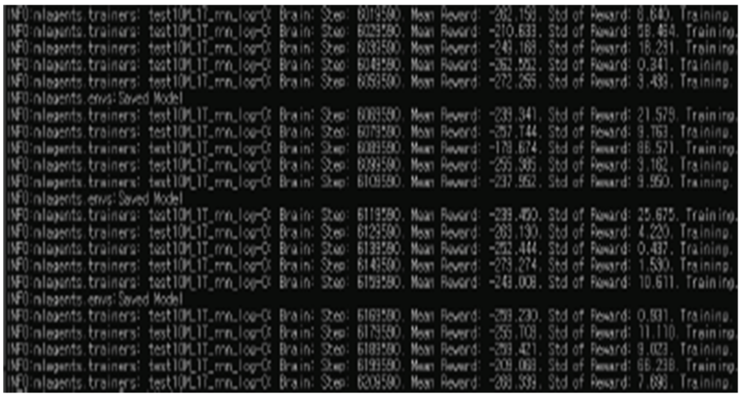


Fig. 3. Flowchart of our scheme. Adapted from Park et al. [23].

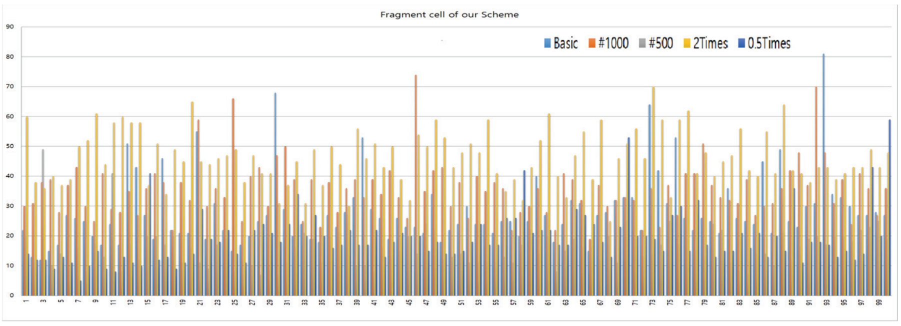
In this paper, we applied the cell selection policy as in Algorithm 2. After checking the cells in all directions based on the cell where the agent is located, it is selected by detecting whether the neighboring cell exists or not. If all neighbor cells are detected and cannot go all the way, random cells are reselected.

```
Algorithm 2:
Get current Cell
If Current Cell != NULL
      Get Current Cell`s Edge
      For all Edge
           If Edge.target != NULL && Edge.target.visited == fall
         Neighbor[i] = true
    EndFor
ENDIF
```

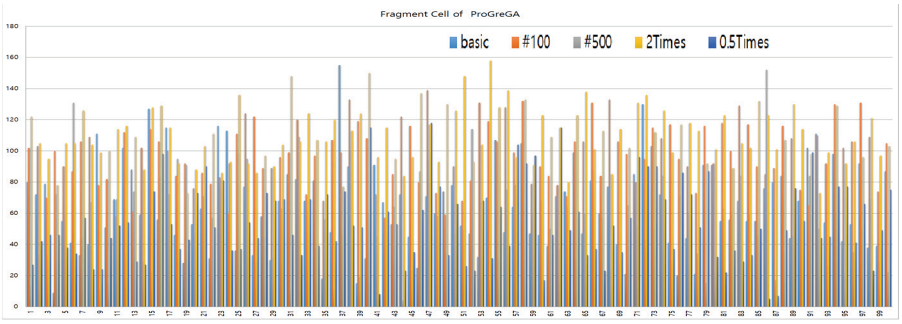


```
INFO:mlagents.trainers: test10M_1T_rnn_log-0: Brain: Step: 6019590. Mean Reward: -282.159. Std of Reward: 8.640. Training.
INFO:mlagents.trainers: test10M_1T_rnn_log-0: Brain: Step: 6029590. Mean Reward: -210.633. Std of Reward: 58.464. Training.
INFO:mlagents.trainers: test10M_1T_rnn_log-0: Brain: Step: 6039590. Mean Reward: -249.168. Std of Reward: 18.231. Training.
INFO:mlagents.trainers: test10M_1T_rnn_log-0: Brain: Step: 6049590. Mean Reward: -262.552. Std of Reward: 0.341. Training.
INFO:mlagents.trainers: test10M_1T_rnn_log-0: Brain: Step: 6059590. Mean Reward: -272.255. Std of Reward: 3.439. Training.
INFO:mlagents.envs:Saved Model
INFO:mlagents.trainers: test10M_1T_rnn_log-0: Brain: Step: 6069590. Mean Reward: -239.341. Std of Reward: 21.579. Training.
INFO:mlagents.trainers: test10M_1T_rnn_log-0: Brain: Step: 6079590. Mean Reward: -257.744. Std of Reward: 9.763. Training.
INFO:mlagents.trainers: test10M_1T_rnn_log-0: Brain: Step: 6089590. Mean Reward: -178.674. Std of Reward: 86.571. Training.
INFO:mlagents.trainers: test10M_1T_rnn_log-0: Brain: Step: 6099590. Mean Reward: -255.985. Std of Reward: 3.162. Training.
INFO:mlagents.trainers: test10M_1T_rnn_log-0: Brain: Step: 6109590. Mean Reward: -237.952. Std of Reward: 9.950. Training.
INFO:mlagents.envs:Saved Model
INFO:mlagents.trainers: test10M_1T_rnn_log-0: Brain: Step: 6119590. Mean Reward: -239.450. Std of Reward: 25.675. Training.
INFO:mlagents.trainers: test10M_1T_rnn_log-0: Brain: Step: 6129590. Mean Reward: -263.130. Std of Reward: 4.220. Training.
INFO:mlagents.trainers: test10M_1T_rnn_log-0: Brain: Step: 6139590. Mean Reward: -252.444. Std of Reward: 0.437. Training.
INFO:mlagents.trainers: test10M_1T_rnn_log-0: Brain: Step: 6149590. Mean Reward: -273.274. Std of Reward: 1.530. Training.
INFO:mlagents.trainers: test10M_1T_rnn_log-0: Brain: Step: 6159590. Mean Reward: -243.008. Std of Reward: 10.611. Training.
INFO:mlagents.envs:Saved Model
INFO:mlagents.trainers: test10M_1T_rnn_log-0: Brain: Step: 6169590. Mean Reward: -259.230. Std of Reward: 0.931. Training.
INFO:mlagents.trainers: test10M_1T_rnn_log-0: Brain: Step: 6179590. Mean Reward: -255.708. Std of Reward: 11.110. Training.
INFO:mlagents.trainers: test10M_1T_rnn_log-0: Brain: Step: 6189590. Mean Reward: -259.421. Std of Reward: 9.023. Training.
INFO:mlagents.trainers: test10M_1T_rnn_log-0: Brain: Step: 6199590. Mean Reward: -209.068. Std of Reward: 66.238. Training.
INFO:mlagents.trainers: test10M_1T_rnn_log-0: Brain: Step: 6209590. Mean Reward: -268.339. Std of Reward: 7.698. Training.
```

Fig. 4. Process of our simulation.



(a)



(b)

Fig. 5. Fragment cells group of our scheme (a) and ProGreGA.

In the propose scheme, the average fragmented cells occurred is two or three times more than that occurred in the original algorithm. Fig. 4 show the process of our simulation.

The 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. The conditional fragmented cell of a model which has been learned through the same condition is shown as a graph in Fig. 5.

The experimental result of a model which has been learned through proposed technic shows 75% increase in its performance compared with the original performance of the algorithm. The occurrence aspect of the conditional fragmented cell shows no big difference from the original algorithm.

4. Conclusion

As game users have increased and various game services have provided, a large amount of event data will be generated over time at gaming servers. Therefore, a load balancing scheme for gaming servers is one of the critical considerations. In this paper, we propose a load balancing agent that combines greedy algorithm of dynamic allocation programming method and PPO which is one of reinforcement learning.

We have used ML-agents library from Unity3D to simulate the load balancing agent for MMOG. It consists of a game world which consists of two-dimensional map and each game world contains FSM bot in the place of 750 game users. Also, we have compared performances of our proposed scheme and a previous research, ProGreGA by running a simulation to verify of our scheme. Our proposed scheme has been shown the show the effectiveness of suggested technic through comparing ProGReGA with simulated results.

Acknowledgement

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. 2019R1A2C1008533) and this work was supported by 2021 Hongik University Research Fund.

References

[1] J. Andrews, S. Singh, Q. Ye, X. Lin, and H. Dhillon, "An overview of load balancing in HetNets: old myths and open problems," *IEEE Wireless Communications*, vol. 21, no. 2, pp. 18-25, 2014.

[2] O. K. Tonguz and E. Yanmaz, "The mathematical theory of dynamic load balancing in cellular networks," *IEEE Transactions on Mobile Computing*, vol. 7, no. 12, pp. 1504-1518, 2008.

[3] H. Kim, G. de Veciana, X. Yang, and M. Venkatachalam, "Distributed alpha-optimal user association and cell load balancing in wireless networks," *IEEE/ACM Transactions on Networking*, vol. 20, no. 1, pp. 177-190, 2012

[4] Z. Zhang, L. Ma, K. K. Leung, L. Tassiulas, and J. Tucker, "Q-placement: reinforcement-Learning-based service placement in software defined networks," in *Proceedings of 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, Vienna, Austria, 2018, pp. 1527-1532.

[5] T. He, J. A. Stankovic, C. Lu and T. Abdelzaher, SPEED: a stateless protocol for real-time communication in sensor networks," in *Proceedings of IEEE 23rd International conference on Distributed Computing Systems*, Providence, RI, 2013, pp. 46-55.

[6] S. T. Cheng and T. Y. Chang, "An adaptive learning scheme for load balancing with zone partition in multi-sink wireless sensor network," *Expert Systems with Applications*, vol. 39, no. 10, pp. 9427-9434, 2012.

[7] R. Kacimi, R. Dhaou and A. L. Beylot, "Load balancing techniques for lifetime maximizing in wireless sensor networks," *Ad Hoc Networks*, vol. 11, pp. 2172-2186, 2013.

[8] W. H. Liao, K. P. Shih, and W. C. Wu, "A grid-based dynamic load balancing approach for data-centric storage in wireless sensor networks," *Computers & Electrical Engineering*, vol. 36, no. 1, pp. 19-30, 2010.

[9] H. Y. Kim, H. J. Park, and S. Lee, "A hybrid load balancing scheme for games in wireless networks," *International Journal of Distributed Sensor Networks*, vol. 10, no. 5, article no. 380318, 2014. https://doi.org/10.1155%2F2014%2F380318

[10] V. Nae, A. Iosup, and R. Prodan, "Dynamic resource provisioning in massively multiplayer online games," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 3, pp. 380-395, 2011.

[11] R. E. De Grande and A. Boukerche, "Dynamic partitioning of distributed virtual simulations for reducing communication load," in *Proceedings of 2009 IEEE International Workshop on Haptic Audio visual Environments and Games*, Lecco, Italy, 2009, pp. 176-181.

[12] P. Quax, J. Cleuren, W. Vanmontfort, and W. Lamotte, "Empirical evaluation of the efficiency of spatial subdivision schemes and load balancing strategies for networked games," in *Proceedings of 2011 Proceedings of 20th International Conference on Computer Communications and Networks (ICCCN)*, Lahaina, HI, 2011, pp. 1-6.

[13] X. Li, Y. J. Kim, R. Govindan, and W. Hong, "Multi-dimensional range queries in sensor networks," in *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems*, Los Angeles, CA, 2003, pp. 63-75.

[14] R. Tlili and Y. Slimani, "A hierarchical dynamic load balancing strategy for distributed data mining," *International Journal of Advanced Science and Technology*, vol. 39, pp. 21-48, 2012.

[15] A. Souri and R. Hosseini, “A state-of-the-art survey of malware detection approaches using data mining techniques,” *Human-centric Computing and Information Sciences*, vol. 8, article no. 3, 2018. https://doi.org/10.1186/s13673-018-0125-x
[16] J. Wang, X. Gu, W. Liu, A. K. Sangaiah, and H. J. Kim, “An empower Hamilton loop based data collection algorithm with mobile agent for WSNs,” *Human-centric Computing and Information Sciences*, vol. 9, article no. 18, 2019. https://doi.org/10.1186/s13673-019-0179-4
[17] C. Yin, B. Zhou, Z. Yin, and J. Wang, “Local privacy protection classification based on human-centric computing,” *Human-centric Computing and Information Sciences*, vol. 9, article no. 33, 2019. https://doi.org/10.1186/s13673-019-0195-4
[18] J. Li, G. Luo, N. Cheng, Q. Yuan, Z. Wu, S. Gao, and Z. Liu, “An end-to-end load balancer based on deep learning for vehicular network traffic control,” *IEEE Internet of Things Journal*, vol. 6, no. 1, pp. 953-966, 2019.
[19] J. N. Foerster, Y. M. Assael, N. De Freitas, and S. Whiteson, “Learning to communicate with deep multi-agent reinforcement learning,” *Advances in Neural Information Processing Systems*, vol. 29, pp. 2137-2145, 2016.
[20] M. C. Fu, “Gradient estimation,” *Handbooks in Operations Research and Management Science*, vol. 13, pp. 575-616, 2006.
[21] L. Chen, J. Lingys, K. Chen, and F. Liu, “Auto: scaling deep reinforcement learning for datacenter-scale automatic traffic optimization,” in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, Budapest, Hungary, 2018, pp. 191-205.
[22] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA: MIT Press, 2016.
[23] J. M. Park, H. Y. Kim, and S. H. Cho, “A study on load distribution of gaming server using proximal policy optimization,” *Journal of Korea Game Society*, vol. 19, no. 3, pp. 5-14, 2019.

Hye-Young Kim **https://orcid.org/0000-0001-7013-7510**

She 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.