Adaptive Scheduling Based on Intelligent Agents in Edge-Cloud Computing Environments

*JongBeom Lim**

Division of ICT Convergence, Pyeongtaek University, Korea
jblim@ptu.ac.kr

Abstract

Scheduling in cloud computing environments has been extended to support the Internet of Things (IoT) applications, which require additional quality of services such as energy consumption and real-time properties. To this end, edge-cloud computing environments are prevalently deployed by encompassing the fog management layer. However, traditional scheduling techniques for cloud tasks have limited capabilities to support real-time properties required for IoT applications. In this paper, we propose a deep learning-based dynamic cloud scheduling technique using intelligent agents, which intelligently adapt to users' requirements and selective quality of services based on distributed learning in edge-cloud computing environments. The proposed cloud task scheduling method is composed of two logical components: distributed learning management (learning distribution and aggregation) and intelligence management of multi-agents, which are independent of each other. The performance results show that the self-employed agents intelligently adapt to their environments and perform hyperparameter learning for efficient and effective task scheduling in edge-cloud computing environments.

Keywords: Edge computing, Cloud computing, Task scheduling, Distributed learning, Multi-agents

1 Introduction

According to a recent report, the size of the cloud computing market is expected to surpass $2,321 billion by 2023, with 16% of CAGR from 2023 to 2032 [1]. Thanks to the enhanced capabilities, cloud computing embraces emerging technologies, such as the Internet of Things (IoT) and artificial intelligence (machine learning and deep learning) [2]. One of the examples is edge-cloud computing [3-5]. Instead of contacting the central cloud computing server, edge-cloud computing deploys edge servers to several places near cloud users [6-8]. Thus, edge-cloud computing supports low latency and real-time properties for IoT applications [9-10].

At the same time, many cloud task scheduling techniques have been proposed using nature-inspired algorithms [11-13], linear programming [14-15], and multi-objective optimization [16-18]. However, these cloud task scheduling techniques have one critical limitation: The complexity of the scheduling methods is unavoidably high. Thus, the scheduling time is affected by the number of nodes, the number of cloud tasks, or the number of cloud users [19-20]. Due to the long scheduling time, deployment of the scheduling policies is not recommended.

Recently, artificial intelligence-based cloud task scheduling techniques also have been proposed to overcome the limitation [21-22]. These techniques are based on deep neural networks [23-24], deep Q-learning (DQL) [25-26], deep reinforcement learning (DRL) [27-28], etc. The advantage of using artificial intelligence-based cloud task scheduling techniques is a relatively lower complexity in scheduling algorithms compared to traditional scheduling methods [29-30]. In addition, the parameters of cloud task scheduling policies can be adjusted and optimized dynamically. To further optimize the scheduling time and learning, distributed learning techniques have been used for deep neural network-based scheduling algorithms [31-32]. However, local agents for distributed learning are designed to be compatible only with the parameter server. Hence, local decision or intelligent cloud task scheduling of multi-agents is limited [33].

In this paper, we propose a deep learning-based dynamic cloud scheduling technique using intelligent agents, which intelligently adapts to users' requirements and selective quality of services [34] based on distributed learning in edge–cloud computing environments. The proposed cloud task scheduling method is composed of two logical components: distributed learning management (learning distribution and aggregation) and intelligent management of multi-agents, independent from each other. The first component (distributed learning management) allows multi-agents to communicate with the parameter server and process local learning. The second component (intelligent management) lets each local agent perform intelligent orchestration of learning. In other words, with intelligent multi-agents, our cloud task scheduling technique adapts to the system environment based on requirements, quality of services, and service-level objectives.

The main contributions of the paper can be summarized as follows:

1. We formulate the cloud task scheduling problem and edge-cloud computing environments with the fog management layer.

*Corresponding Author: JongBeom Lim; E-mail: jblim@ptu.ac.kr
DOI: https://doi.org/10.70003/160792642024072504011

2. We reveal the limitations of existing cloud task scheduling techniques with mini benchmark results.
3. We design and implement a deep learning-based dynamic cloud scheduling technique using intelligent agents, which intelligently adapt to users' requirements and selective quality of services based on distributed learning.
4. We compare the performance results with state-of-the-art studies based on deep Q-learning and deep reinforcement learning techniques.

2 System Model and Problem Definition

2.1 System Model

The basic architecture of edge-cloud computing environments is depicted in Figure 1. Since we consider IoT applications, there is an IoT layer in the architecture. In the IoT layer, there are numerous devices that can be deployed, such as laptops, smartphones, and small computing devices (Arduino, Raspberry Pi, etc.). Due to the diversity of IoT devices, orchestrating them requires a pre-defined lifecycle. That is, the following lifecycle stages can be used on a cyclic basis: (1) deploy, (2) monitor, (3) service, (4) manage, (5) update, and (6) decommission.

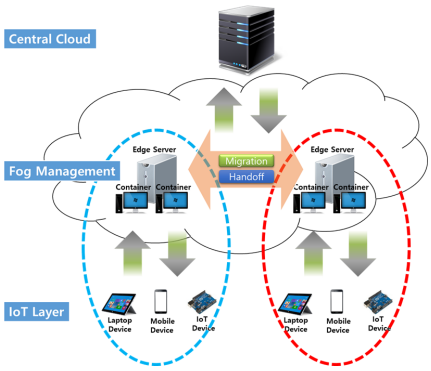

Figure 1. Edge-cloud computing architecture

In order to maintain the lifecycle of IoT devices, another level of management of the edge-cloud architecture is required: fog management. Instead of contacting the central cloud server, IoT devices communicate with nearby edge servers. Thus, the edge servers are deployed between IoT devices and the central cloud server in edge-cloud computing environments. The fundamental roles of the edge servers are the core aspect of cloud task scheduling for the basic edge-cloud computing architecture. In other words, the edge servers directly communicate with both IoT devices and the central cloud server. In addition, an edge server collaborates with other edge servers in the edge-cloud computing environments. Specifically, an edge server is capable of handing off a user's task and migrating computing nodes (virtual machines or containers) from one to another. In this manner, cloud tasks can be scheduled efficiently. Considering operational costs and migration overheads, we use containers for our edge-cloud computing architecture.

2.2 Problem Definition

We consider cloud task scheduling in edge-cloud computing environments based on deep learning [35-36]. Specifically, we use the feed-forward neural network architecture for scheduling decisions (task allocation, task migration, and task management for users' requirements, quality of services, and service-level objectives). To optimize the learning process of cloud task scheduling, we use a distributed learning approach. Instead of making the central cloud server perform the learning process solely, we let edge servers cooperatively perform the hyper-parameter learning.

The distributed learning architecture is shown in Figure 2. There is a parameter server and multiple agents in the architecture. The parameter server is in charge of the distribution of data partitions to the multiple agents in edge-cloud computing environments. Then, it aggregates the results from the multiple agents. After these processes are finished, cloud task scheduling can proceed based on the scheduling decision. The agents are in charge of performing the hyperparameter learning process based on the data partition. Then, each agent forwards the result to the parameter server.

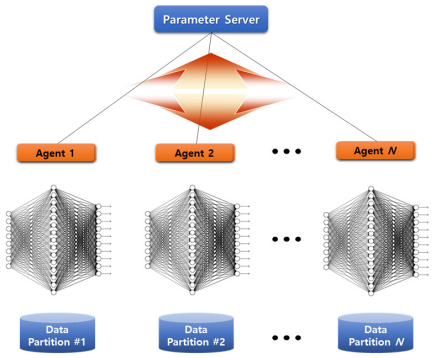

Figure 2. Basic distributed learning architecture

Although the basic distributed learning process and architecture are functional, we enhance and extend the architecture by resolving two limitations: (1) The basic distributed learning architecture cannot perform the adaptive hyper-parameter learning since all the agents have the same system parameter; and (2) it does not consider users' requirements, quality of services, or service level objectives; these properties cannot be applied online in a dynamic fashion.

The problem we are trying to solve in this paper is based on the following research questions: (1) how to make multi-agents intelligently learn their environment based on users' requirements, quality of services, or service level objectives; and (2) how to apply multi-agent intelligence to task scheduling in edge-cloud computing environments.

3 Proposed Solution

In this section, we detail the two core algorithms for a deep learning–based dynamic cloud scheduling technique. The first algorithm is for the adaptive learning process for

an agent, and the second algorithm is for the task scheduling algorithm based on the first algorithm. Note that the first algorithm is executed in each agent independently, and the second algorithm can be executed in the parameter server or the central cloud server after the first algorithm is complete.

Algorithm 1. Adaptive learning process for an agent

```
1:  Input: Environmental parameters of the system
2:  Output: Meta data for task scheduling
3:  Initialization: Server ← get(ParameterServer);
4:     Coverage ← get(Location, Edge_i);
5:     Set_user ← get(User, Edge_i, Coverage);
6:     Set_task ← get(Task, Edge_i, Coverage);
7:     Sum_affinity ← null;
8:     Meta ← null;
9:  for each User_i ∈ Set_user do
10:    Hist_i ← get(History, User_i);
11:    Curr_i ← get(Current, User_i);
12:    Vio_i ← get(Hist_i, SLO);
13:    Pref_i ← get(Curr_i, Priority);
14:    Meta ← Meta ∪ Vio_i ∪ Pref_i;
15: end for
16: for each Task_i ∈ Set_task do
17:    Mon_i ← monitor(Task_i);
18:    Aff_i ← evaluate_affiinity(Mon_i, Edge_i);
19:    Sum_affinity ← Sum_affinity + Affi_i;
20: end for
21: Meta ← Meta ∪ Sum_affinity;
22: return Meta;
```

Algorithm 1 shows the adaptive learning process for an agent. The input of the algorithm is the environmental parameters of the edge-cloud system, and the output is metadata for cloud task scheduling. Therefore, the cloud task algorithm could proceed after Algorithm 1 is complete at each agent. Based on the cloud task scheduling policies, we allow the cloud task scheduling algorithm to be executed before Algorithm 1 is complete. In this scenario, however, the effectiveness of the adaptive learning process can be insignificant.

There are a few initialization steps for Algorithm 1. In other words, it retrieves some environmental parameters by performing the get() function. In line 3, the algorithm gets the parameter server, and it retrieves the coverage based on its location and specifications of the edge server the agent resides on in line 4. Further, it gets the set of users and tasks, and sets two parameters (*Sum_affinity* and *Meta*) to null (lines 5–8).

For the adaptive learning process, each agent performs two repetitive statements. The first repetitive statement is for *Set_user*. For each *User_i*, it retrieves both historical information and current data for the user (lines 9–11). Based on the history information, it calculates the cumulative violations (*Vio_i*) of service level objectives (line 12). Note that the *Vio_i* variable is used in the loss function of deep neural networks as Equation 1.

$$J(U^1, ..., U^L) = 1/|M| \cdot \Sigma |\mathbf{y} - \mathbf{o}|^2 + \lambda Vio\_i \cdot E(SLO). \quad \textbf{(1)}$$

In line 13, it retrieves the user's preference (priority and quality of services such as energy consumption and low latency), which can be used in Algorithm 2 when performing the cloud task scheduling. Then, the user data (*Vio_i* and *Pref_i*) are merged into the *Meta* variable.

The second repetitive statement is for *Set_task* (lines 16–20). Note that the *Set_task* variable contains the currently running cloud tasks in the edge-cloud system. With the *Set_task* variable, it retrieves the monitoring information of each *Task_i* (line 17). Then, it calculates the affinity between *Task_i* and *Edge_i* (line 18). The evaluate_affinity() function returns [0, 1] and a higher numeric value if *Task_i* is suitable for *Edge_i*. Then, the *Affi_i* variable is added to the *Sum_affinity* variable (line 19). After merging the *Sum_affinity* variable to the *Meta* variable, it returns *Meta* (line 22).

Algorithm 2 shows the task scheduling algorithm for intelligent multi-agents. The input of the algorithm is the *Meta* variable from Algorithm 1, and the output is the initial cloud task allocation and migration schedule based on affinity metadata. The initialization steps retrieve the two threshold values (*Thd_sum* and *Thd_affi*), which are used in the algorithm when allocation and migration are scheduled, respectively.

Algorithm 2. Task scheduling algorithm for intelligent multi-agents

```
1:  Input: Meta data for task scheduling
2:  Output: Task allocation and migration schedule
3:  Initialization: Meta_global ← null;
4:     Thd_sum ← get(Sum_Affinity, Threshold);
5:     Thd_affi ← get(Affinity, Threshold);
6:  // Code for Passive Thread
7:  for each Agent_i ∈ Set_agent do
8:     Meta_i ← wait(Agent_i);
9:     Meta_global ← Meta_global ∪ Meta_i;
10: end for
11: // Code for Active Thread
12: for each Task_i ∈ Set_new_task do
13:    Edge_j ← find_edge(Pref_k of the owner);
14:    allocate(Task_i, Edge_j);
15: end for
16: for min(Sum_affinity_i) ∈ Meta_global do
17:    if (Sum_affinity_i > Thd_sum) then
18:       break;
19:    end if
20:    for each Task_i ∈ Edge_i do
21:       Edge_j ← find_max_affinity(Task_i);
22:       if calc_affi(Task_i, Edge_j) > Thd_affi then
23:          schedule_migration(Task_i, Edge_j);
24:       end if
25:    end for
26: end for
```

There are two threads for the cloud task scheduling algorithm: passive and active threads. The passive thread is for collecting and aggregating the *Meta* variable from multi-agents in the edge-cloud system. The active thread is for the cloud task scheduling for the initial allocation of cloud tasks and migration scheduling. In lines 7–9, the passive thread waits for each agent to make the global variable (*Meta_global*).

In the active thread, there are two repetitive statements. The first repetitive statement (lines 12–15) is for the initial

allocation of cloud tasks. When a new task (*Task_i*) arrives, it performs the find_edge() function with the *Pref_k* parameter. Note that the *Pref_k* parameter contains the preference information (priority, quality of services such as energy consumption and low latency) of the owner of the new task (*Task_i*). After getting the edge information (*Edge_j*), it allocates *Task_i* to *Edge_j* (line 14).

It is important to note that the cloud task allocation of Algorithm 2 and built-in cloud task allocation based on deep neural networks conflict. However, we resolve the conflict between the two allocations by an additional procedure. Specifically, in deep neural networks, it allocates a task to an edge server based on a randomized fashion, and this randomized fashion does not guarantee the optimal solution. Hence, we use the cloud task allocation of Algorithm 2 for the initial allocation. Then, we let the allocation from deep neural networks orchestrate them afterward.

The second repetitive statement (lines 16–26) shows the migration schedule based on the *Meta* variable from Algorithm 1. The migration policy is based on the *Sum_affinity* variable. The lower *Sum_affinity is*, the poorer the association between the tasks and the edge server is. Thus, it first finds the edge server that has the lowest *Sum_affinity_i* value (line 16). If the retrieved *Sum_affinity_i* is greater than *Thd_sum*, then the iteration is terminated (lines 17–19). Otherwise, it performs the remaining procedures (lines 20–25).

For each *Task_i* running on *Edge_i*, it finds the edge server suitable for *Task_i* based on the affinity relationship (line 21). When the affinity score between *Task_i* and *Edge_j* is greater than *Thd_affi*, it sets *Task_i* for the migration target to *Edge_j*. Like in Algorithm 1, the migration scheduling policy can conflict with the migration decisions of deep neural networks. In this case, we use the proposed migration scheduling policy in favor of Algorithm 2. When the number of tasks for the migration schedule is manageable and the total utilization of edge servers is low, the two migration policies (Algorithm 2 and deep neural networks) can be used simultaneously.

Regarding the two threshold values (*Thd_sum* and *Thd_affi*), the specific values can be determined based on the monitoring information in the edge–cloud system. If the total utilization of edge servers is low and the number of tasks for the migration schedule is low, the two threshold values can be increased online. Conversely, if the total utilization of edge servers is high and there are too many tasks for the migration schedule, the two threshold values are decreased on the fly.

4 Performance Evaluation

In this section, we provide our experimental results for the proposed dynamic cloud scheduling technique using intelligent agents with distributed learning. As we mentioned before, our cloud scheduling technique adapts to users' requirements and selective quality of services in edge–cloud computing environments. To show the effectiveness of the results, we show the two extreme scenarios for energy consumption and low latency for the purpose of comparisons with 100 scheduling intervals (one interval is 300 seconds).

There are 30 hosts in the edge–cloud system, and a host can serve as an edge server. Figure 3 to Figure 5 show the preliminary experimental results for scheduling time, the number of cloud task migrations, and total migration time. Note that Random uses a randomized fashion for cloud task scheduling, DRL [27-28] is an acronym for deep reinforcement learning, DQL [25-26] is an acronym for deep Q-learning, OursS is our approach that uses deep learning-based scheduling only with default parameters disregarding users' requirements or quality of services, OursB is our approach with a balanced policy for energy consumption and low latency, OursE uses our approach with a biased policy for energy consumption, and OursL uses our approach with a biased policy for low latency.

Figure 3 shows the preliminary results for scheduling time for Random, DRL, DQL, OursS, OursB, OursE, and OursL. Since Random uses the rand() function for cloud task scheduling, the scheduling complexity is low, and its scheduling time is negligible [37]. However, Random shows a high number of cloud task migrations, as shown in Figure 4. Its concomitant total migration time is shown in Figure 5. Random exhibits over 1,300 cloud task migrations and 3,000 seconds of migration time. Although Random's scheduling time is negligible, Random cannot be used in practice due to high task migration time. Hence, we do not include Random in the subsequent experiments.

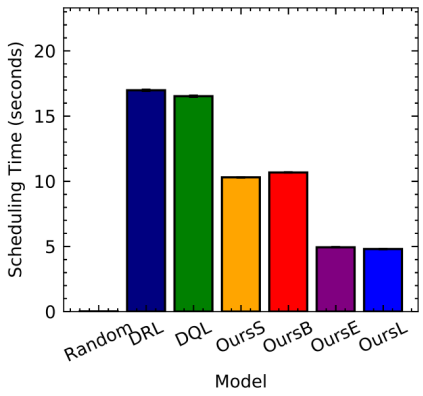

Figure 3. Preliminary results for scheduling time

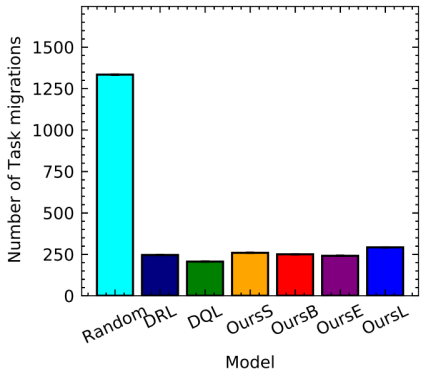

Figure 4. Preliminary results for the number of task migrations

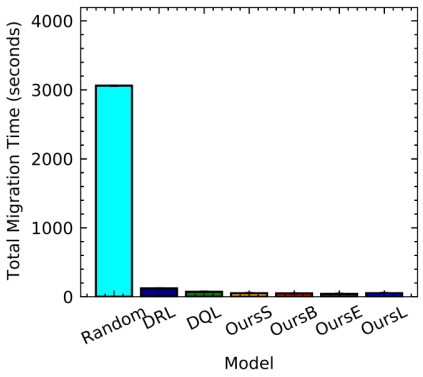

Figure 5. Preliminary results for total migration time

To show the training results with epochs, we measure the average training loss at each epoch for OursB, OursE, and OursL, as shown in Figure 6. The appropriate setting for epochs affects the total training performance in deep neural networks. For the balanced setting for our dynamic cloud task scheduling, we set the epoch parameter as 100. As shown in Figure 6, the average training loss sharply drops in the first 60 epochs, and it stabilizes after 100 epochs.

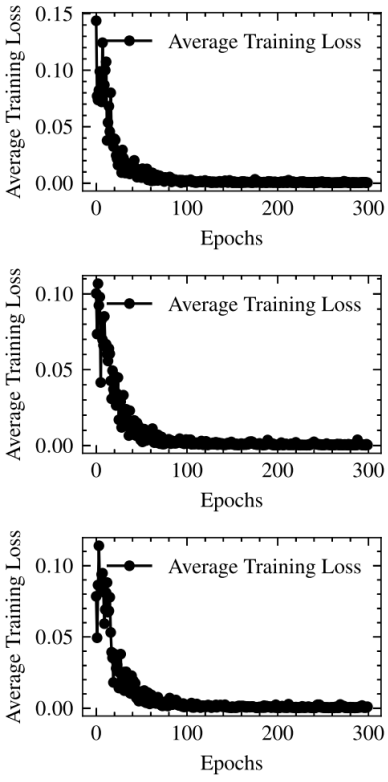

Figure 6. Average training loss for OursE, OursE, and OursL

To show the performance characteristics, we show scheduling time, energy consumption, and response time for DRL, DQL, OursS, OursB, OursE, and OursL. Figure 7 shows the scheduling time characteristics for DRL, DQL, OursS, OursB, OursE, and OursL. The two approaches (OursE and OursL) show the best performance for scheduling time. OursS and OursB show similar performance results, and DRL and DQL show the worst performance for scheduling time. Of the six approaches, DRL exhibits high deviations.

Figure 8 shows the average energy consumption for DRL, DQL, OursS, OursB, OursE, and OursL. DRL and DQL consume lower energy than the other four approaches (OursS, OursB, OursE, and OursL) since DRL and DQL do not perform the distributed learning process. For the average response time, OursB shows the worst performance, and OursL shows the best performance, as shown in Figure 9. The result for OursL is obvious since the users' requirements and quality of services are set for low latency. However, the result for OursB conflicts with basic assumptions. We conjecture that this result stems from the fact that OursB does not put all eggs in one basket. Thus, our implication concludes that it is better to focus on one of the requirements or quality of services.

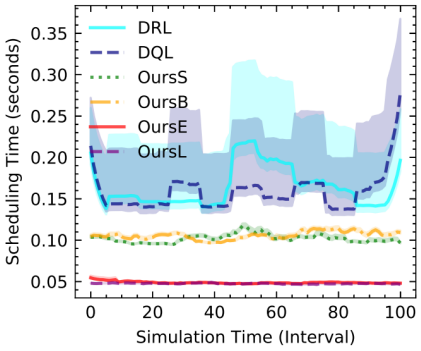

Figure 7. Scheduling time at each interval

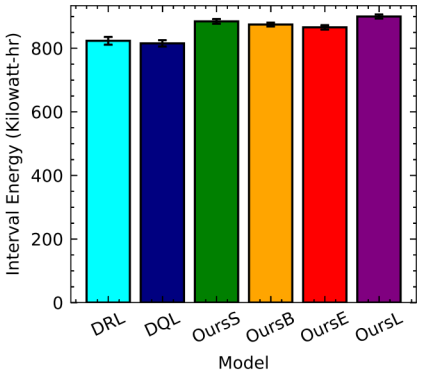

Figure 8. Energy consumption at each interval

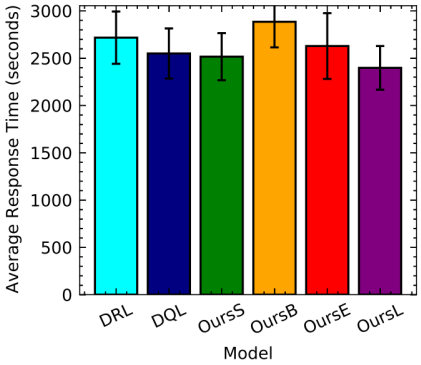

Figure 9. Average response time

To show the main performance results, we measure average CPU utilization, the number of completed tasks per interval, the number of cloud task migrations, and average migration time. Figure 10 shows the average CPU utilization for DRL, DQL, OursS, OursB, OursE, and OursL. The interesting results can be seen from DQL. In early intervals, DQL's CPU utilization is very low and sharply increases from intervals 17 to 24. In other words, it needs some adaptation time in edge-cloud computing environments. On the other hand, OursS, OursE, and OursL show quick adaptation for CPU utilization; they find appropriate positions before 10 intervals.

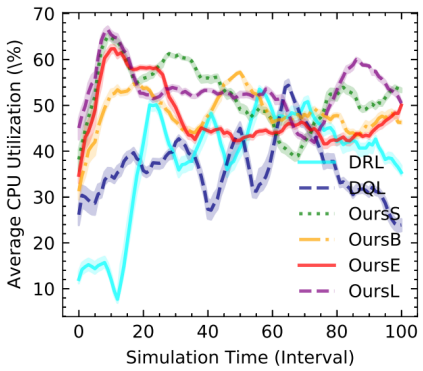

Figure 10. Average CPU utilization

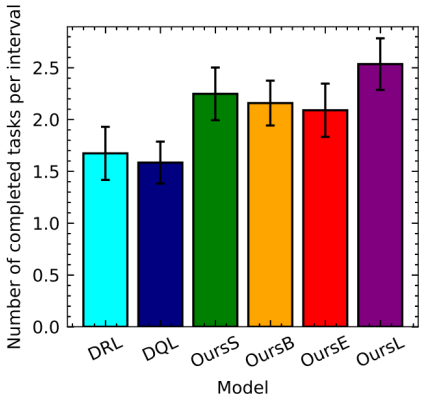

Figure 11. The number of completed tasks

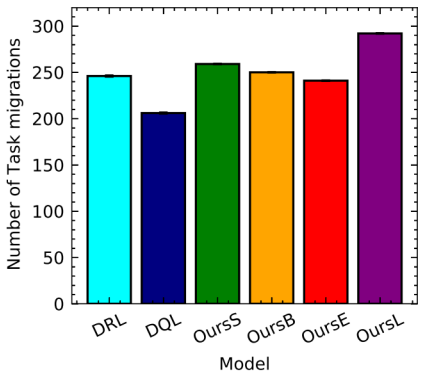

Figure 12. The number of task migrations

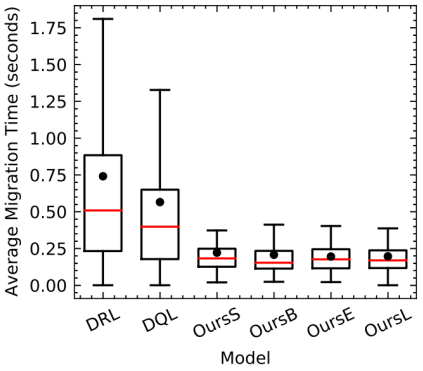

Figure 13. Average migration time

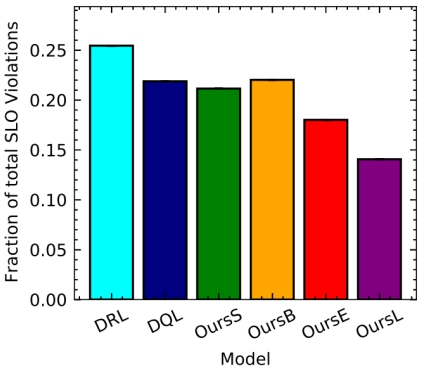

Figure 14. Fraction of total SLO violations

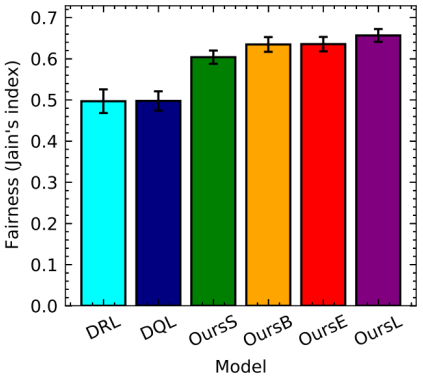

Figure 15. Fairness based on Jain's index

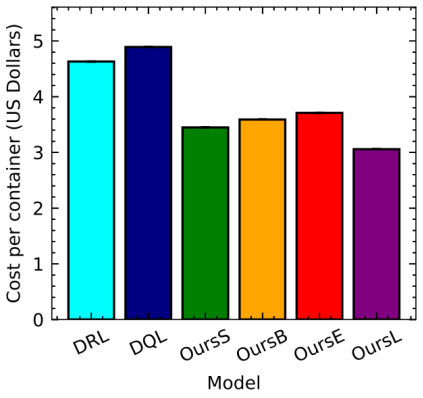

Figure 16. Cost per container

The average CPU utilization significantly affects the number of completed cloud tasks. Figure 11 shows the number of completed cloud tasks for DRL, DQL, OursS, OursB, OursE, and OursL. As we expected, OursS, OursB, OursE, and OursL outperform DRL and DQL. Among our approaches, OursL shows the best performance, and OursE shows the worst performance for the number of completed cloud tasks. Although OursE exhibits relatively low performance, it still outperforms DRL and DQL and consumes low energy among the four approaches (c.f., Figure 8). Figure 12 and Figure 13 shows the number of task migrations and average migration time, respectively. OursL shows the highest number of cloud task migrations; however, the migrations have a low impact on the overall performance since the average migration time is relatively low. The higher migration time means the cloud task scheduler selects the suboptimal migration source and target in terms of coverage, user location, bandwidth, and network speed.

Additionally, we measure service level objective violations, fairness, and cost per container based on Microsoft Azure's cost table to show the usefulness of our approaches. Figure 14 shows the fractions of total service level objective violations for DRL, DQL, OursS, OursB, OursE, and OursL. Note that the baseline of the service level objective violations is the Random approach. DRL shows the worst performance, and OursL shows the best performance for the service level objective violations. The reason for this result is that the service level objective violations are affected by latency and the number of completed tasks. As OursL shows the optimal performance in terms of latency and the number of completed tasks, it shows the lowest service level objective violations.

To show the load-balancing of hosts, we measure Jain's index [38] as defined in Equation 2.

$$J(x_1, x_2, ..., x_n)=(\Sigma x_i)^2/n\cdot x_i^2=1/1+c_v^2, \quad \textbf{(2)}$$

where there are n hosts, x_i is the throughput for the ith interval, and c_v is the coefficient of variations.

Note that the higher the value of Jain's index, the better fairness (load balancing) we have. The worst performance can be seen from DRL and DQL's results, and their values are about 0.5 as shown in Figure 15. On the other hand, our four approaches outperform DRL and DQL, and the best performance can be seen from OursL's result. Lastly, we measure the cost per container of edge servers in the system as shown in Figure 16. The cost is calculated from Microsoft Azure's price table in the South UK region. Our four approaches show a lower cost compared to DRL and DQL. An interesting result can be seen from OursE's result. OursE exhibits lower energy consumption; however, it takes longer to complete the submitted cloud tasks. Therefore, it costs more than other approaches.

5 Conclusion

Making an adaptive cloud task scheduler with effectiveness is not a trivial task and requires careful and comprehensive manual considerations in edge-cloud computing environments. In this paper, we propose a deep learning–based dynamic cloud scheduling technique using intelligent agents that adapt to users' requirements and selective quality of services based on distributed learning in edge-cloud computing environments. The proposed techniques and algorithms answer two research questions: (1) How do we make multi-agents intelligently learn their environment based on users' requirements, quality of services, or service level objectives? (2) How do we apply multi-agents intelligence to task scheduling in edge-cloud computing environments? The performance results show that our dynamic cloud task scheduling algorithms are effective in meeting users' requirements, quality of services, or service level objectives. Future work includes enhancing fault tolerance and localizing straggler nodes in edge-cloud computing environments.

References

[1] K. Thakur, A.-S. K. Pathan, S. Ismat, Distributed Cloud Computing, in: *Emerging ICT Technologies and Cybersecurity: From AI and ML to Other Futuristic Technologies*, Cham: Springer Nature Switzerland, 2023, pp. 185-197.

[2] O. E. L. Castro, X. Deng, J. H. Park, Comprehensive Survey on AI-Based Technologies for Enhancing IoT Privacy and Security: Trends, Challenges, and Solutions, *Human-centric Computing and Information Sciences*, Vol. 13, Article No. 39, August, 2023. https://doi.org/10.22967/HCIS.2023.13.039

[3] M. Alazab, G. Manogaran, C. E. Montenegro-Marin, Trust management for internet of things using cloud computing and security in smart cities, *Cluster Computing*, Vol. 25, No. 3, pp. 1765-1777, June, 2022. https://doi.org/10.1007/s10586-021-03427-9

[4] C. Pandey, Y. K. Sahu, N. Kannan, M. R. Mahmood, P. K. Sethy, S. K. Behera, Futuristic AI Convergence of Megatrends, in: M. R. Mahmood, R. Raja, H. Kaur, S. Kumar, K. K. Nagwanshi (Eds.), *Ambient Intelligence and Internet of Things*, Scrivener Publishing LLC, 2022, pp. 125-188. https://doi.org/10.1002/9781119821847.ch5

[5] M. Gupta, M. Thirumalaisamy, S. Shamsher, A. Pandey, D. Muthiah, N. Suvarna, Patient Health Monitoring using Feed Forward Neural Network with Cloud Based Internet of Things, *2022 2nd International Conference on Advance Computing and Innovative Technologies in Engineering (ICACITE)*, Greater Noida, India, 2022, pp. 924-931. https://doi.org/10.1109/ICACITE53722.2022.9823502

[6] S. Ghafouri, A. Karami, D. B. Bakhtiarvan, A. S. Bigdeli, S. S. Gill, J. Doyle, Mobile-Kube: Mobility-aware and Energy-efficient Service Orchestration on Kubernetes Edge Servers, *2022 IEEE/ACM 15th International Conference on Utility and Cloud Computing (UCC)*, Vancouver, WA, USA, 2022, pp. 82-91. https://doi.org/10.1109/UCC56403.2022.00019

[7] Y. Cui, K. Cao, J. Zhou, T. Wei, Optimizing Training Efficiency and Cost of Hierarchical Federated Learning in Heterogeneous Mobile-Edge Cloud Computing,

IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 42, No. 5, pp. 1518-1531, May, 2023. https://doi.org/10.1109/TCAD.2022.3205551

[8] S. M. Alamouti, F. Arjomandi, M. Burger, Hybrid Edge Cloud: A Pragmatic Approach for Decentralized Cloud Computing, *IEEE Communications Magazine*, Vol. 60, No. 9, pp. 16-29, September, 2022. https://doi.org/10.1109/MCOM.001.2200251

[9] A. Hazra, P. Rana, M. Adhikari, T. Amgoth, Fog computing for next-generation Internet of Things: Fundamental, state-of-the-art and research challenges, *Computer Science Review*, Vol. 48, Article No. 100549, May, 2023. https://doi.org/10.1016/j.cosrev.2023.100549

[10] C. S. T. Shanthakumar, N. Harish, Eshanya, A. Giridharan, Internet of Things and Edge Computing for Real Time Applications, *2023 International Conference on Intelligent and Innovative Technologies in Computing, Electrical and Electronics (IITCEE)*, Bengaluru, India, 2023, pp. 1137-1141. https://doi.org/10.1109/IITCEE57236.2023.10091014

[11] S. Kaul, Y. Kumar, U. Ghosh, W. Alnumay, Nature-inspired optimization algorithms for different computing systems: novel perspective and systematic review, *Multimedia Tools and Applications*, Vol. 81, No. 19, pp. 26779-26801, August, 2022. https://doi.org/10.1007/s11042-021-11011-x

[12] M. Adhikari, S. N. Srirama, T. Amgoth, A comprehensive survey on nature-inspired algorithms and their applications in edge computing: Challenges and future directions, *Software: Practice and Experience*, Vol. 52, No. 4, pp. 1004-1034, April, 2022. https://doi.org/10.1002/spe.3025

[13] G. Natesan, J. Ali, P. Krishnadoss, R. Chidambaram, M. Nanjappan, Optimization techniques for task scheduling criteria in IaaS cloud computing atmosphere using nature inspired hybrid spotted hyena optimization algorithm, *Concurrency and Computation: Practice and Experience*, Vol. 34, No. 24, Article No. e7228, November, 2022. https://doi.org/10.1002/cpe.7228

[14] A. Hazra, P. K. Donta, T. Amgoth, S. Dustdar, Cooperative Transmission Scheduling and Computation Offloading With Collaboration of Fog and Cloud for Industrial IoT Applications, *IEEE Internet of Things Journal*, Vol. 10, No. 5, pp. 3944-3953, March, 2023. https://doi.org/10.1109/JIOT.2022.3150070

[15] A. Mahjoubi, K. J. Grinnemo, J. Taheri, EHGA: A Genetic Algorithm Based Approach for Scheduling Tasks on Distributed Edge-Cloud Infrastructures, *2022 13th International Conference on Network of the Future (NoF)*, Ghent, Belgium, 2022, pp. 1-5. https://doi.org/10.1109/NoF55974.2022.9942552

[16] A. Lakhan, M. A. Mohammed, M. Elhoseny, M. D. Alshehri, K. H. Abdulkareem, Blockchain multi-objective optimization approach-enabled secure and cost-efficient scheduling for the Internet of Medical Things (IoMT) in fog-cloud system, *Soft Computing*, Vol. 26, No. 13, pp. 6429-6442, July, 2022. https://doi.org/10.1007/s00500-022-07167-9

[17] T. Xie, C. Li, N. Hao, Y. Luo, Multi-objective optimization of data deployment and scheduling based on the minimum cost in geo-distributed cloud, *Computer Communications*, Vol. 185, pp. 142-158, March, 2022. https://doi.org/10.1016/j.comcom.2021.12.022

[18] M. Ouyang, J. Xi, W. Bai, K. Li, Band-Area Application Container and Artificial Fish Swarm Algorithm for Multi-Objective Optimization in Internet-of-Things Cloud, *IEEE Access*, Vol. 10, pp. 16408-16423, February, 2022. https://doi.org/10.1109/ACCESS.2022.3150326

[19] P. Jawade, G. M. Borkar, S. Ramachandram, Confinement forest-based enhanced min-min and max-min technique for secure multicloud task scheduling, *Transactions on Emerging Telecommunications Technologies*, Vol. 33, No. 9, Article No. e4515, September, 2022. https://doi.org/10.1002/ett.4515

[20] N. Khaledian, K. Khamforoosh, S. Azizi, V. Maihami, IKH-EFT: An improved method of workflow scheduling using the krill herd algorithm in the fog-cloud environment, *Sustainable Computing: Informatics and Systems*, Vol. 37, Article No. 100834, January, 2023. https://doi.org/10.1016/j.suscom.2022.100834

[21] D. Jorge-Martinez, S. A. Butt, E. M. Onyema, C. Chakraborty, Q. Shaheen, E. De-La-Hoz-Franco, P. Ariza-Colpas, Artificial intelligence-based Kubernetes container for scheduling nodes of energy composition, *International Journal of System Assurance Engineering and Management*, July, 2021. https://doi.org/10.1007/s13198-021-01195-8

[22] C. V. Marian, Artificial Intelligence-Based Algorithm for Resources Allocation, *2022 14th International Conference on Electronics, Computers and Artificial Intelligence (ECAI)*, Ploiesti, Romania, 2022, pp. 1-4. https://doi.org/10.1109/ECAI54874.2022.9847517

[23] S. Tuli, S. R. Poojara, S. N. Srirama, G. Casale, N. R. Jennings, COSCO: Container Orchestration Using Co-Simulation and Gradient Based Optimization for Fog Computing Environments, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 33, No. 1, pp. 101-116, January, 2022. https://doi.org/10.1109/TPDS.2021.3087349

[24] S. Tuli, G. Casale, N. R. Jennings, SplitPlace: AI Augmented Splitting and Placement of Large-Scale Neural Networks in Mobile Edge Environments, *IEEE Transactions on Mobile Computing*, Vol. 22, No. 9, pp. 5539-5554, September, 2023. https://doi.org/10.1109/TMC.2022.3177569

[25] Z. Tong, H. Chen, X. Deng, K. Li, K. Li, A scheduling scheme in the cloud computing environment using deep Q-learning, *Information Sciences*, Vol. 512, pp. 1170-1191, February, 2020. https://doi.org/10.1016/j.ins.2019.10.035

[26] D. Ding, X. Fan, Y. Zhao, K. Kang, Q. Yin, J. Zeng, Q-learning based dynamic task scheduling for energy-efficient cloud computing, *Future Generation Computer Systems*, Vol. 108, pp. 361-371, July, 2020. https://doi.org/10.1016/j.future.2020.02.018

[27] Y. Liu, Y. Ping, L. Zhang, L. Wang, X. Xu, Scheduling of decentralized robot services in cloud manufacturing with deep reinforcement learning, *Robotics and Computer-Integrated Manufacturing*, Vol. 80, Article No. 102454, April, 2023. https://doi.org/10.1016/j.rcim.2022.102454

[28] Y. Ping, Y. Liu, L. Zhang, L. Wang, X. Xu, Sequence generation for multi-task scheduling in cloud manufacturing with deep reinforcement learning, *Journal of Manufacturing Systems*, Vol. 67, pp. 315-337, April, 2023. https://doi.org/10.1016/j.jmsy.2023.02.009

[29] E. H. Houssein, A. G. Gad, Y. M. Wazery, P. N. Suganthan, Task Scheduling in Cloud Computing based on Meta-heuristics: Review, Taxonomy, Open Challenges, and Future Trends, *Swarm and Evolutionary Computation*, Vol. 62, Article No. 100841, April, 2021. https://doi.org/10.1016/j.swevo.2021.100841

[30] X. Wang, X. Li, V. C. M. Leung, Artificial Intelligence-Based Techniques for Emerging Heterogeneous Network: State of the Arts, Opportunities, and Challenges, *IEEE Access*, vol. 3, pp. 1379-1391, August, 2015. https://doi.org/10.1109/ACCESS.2015.2467174

[31] J. Pang, Z. Han, R. Zhou, H. Tan, Y. Cao, Online scheduling algorithms for unbiased distributed learning over wireless edge networks, *Journal of Systems Architecture*, Vol. 131, Article No. 102673, October, 2022. https://doi.org/10.1016/j.sysarc.2022.102673

[32] Z. Han, R. Zhou, J. Pang, Y. Cao, H. Tan, Online Scheduling Unbiased Distributed Learning over Wireless Edge Networks, *2021 IEEE 27th International Conference on Parallel and Distributed Systems (ICPADS)*, Beijing, China, 2021, pp. 599-606. https://doi.org/10.1109/ICPADS53394.2021.00080

[33] M. Chen, D. Gündüz, K. Huang, W. Saad, M. Bennis, A. V. Feljan, H. V. Poor, Distributed Learning in Wireless Networks: Recent Progress and Future Challenges, *IEEE Journal on Selected Areas in Communications*, Vol. 39, No. 12, pp. 3579-3605, December, 2021. https://doi.org/10.1109/JSAC.2021.3118346

[34] K. Peng, B. Zhao, M. Bilal, X. Xu, A. Nayyar, QoS-Aware Cloud-Edge Collaborative Micro-Service Scheduling in the IIoT, *Human-centric Computing and Information Sciences*, Vol. 13, Article No. 28, June, 2023. https://doi.org/10.22967/HCIS.2023.13.028

[35] A. A. Malibari, M. K. Nour, F. N. Al-Wesabi, R. Alabdan, A. Mohamed, M. Al Duhayyim, A. Alkhayyat, D. Gupta, Metaheuristics with Deep Learning Enabled Epileptic Seizure Classification for Smart Healthcare on Cyborg Robots, *Human-centric Computing and Information Sciences*, Vol. 13, Article No. 39, August, 2023. https://doi.org/10.22967/HCIS.2023.13.039

[36] J. I. Lee, A Study on Peak Load Prediction Using TCN Deep Learning Model, *KIPS Transactions on Software and Data Engineering*, Vol. 12, No. 6, pp. 251-258, June, 2023. https://doi.org/10.3745/KTSDE.2023.12.6.251

[37] Z. Peng, D. Cui, J. Zuo, Q. Li, B. Xu, W. Lin, Random task scheduling scheme based on reinforcement learning in cloud computing, *Cluster Computing*, Vol. 18, No. 4, pp. 1595-1607, December, 2015. https://doi.org/10.1007/s10586-015-0484-2

[38] A. B. Sediq, R. H. Gohary, R. Schoenen, H. Yanikomeroglu, Optimal Tradeoff Between Sum-Rate Efficiency and Jain's Fairness Index in Resource Allocation, *IEEE Transactions on Wireless Communications*, Vol. 12, No. 7, pp. 3496-3509, July, 2013. https://doi.org/10.1109/TWC.2013.061413.121703

Biography

JongBeom Lim received M.S. and Ph.D. degrees in computer science and education from Korea University, Korea, in 2011 and 2014, respectively. From 2015 to 2017, he was a visiting professor with the IT Convergence Education Center, Dongguk University, Korea. From 2017 to 2021, he was an assistant professor with the department of game and multimedia engineering, Tech University of Korea, Korea. Since 2021, he is with the division of ICT convergence, Pyeongtaek University, Korea. His research interests fall within the general fields of computer science and its applications including distributed computing and algorithms; cloud computing and virtualization; artificial intelligence and big data analytics; mobile and sensor networks; and fault tolerant and resilient techniques.