applied sciences

MDPI

Article

Scalable Fog Computing Orchestration for Reliable Cloud Task Scheduling

Jongbeom Lim

Smart Contents Major, Division of ICT Convergence, Pyeongtaek University, 3825, Seodong-daero, Pyeongtaek-si 17869, Korea; jblim@ptu.ac.kr

Abstract: As Internet of Things (IoT) and Industrial Internet of Things (IIoT) devices are becoming increasingly popular in the era of the Fourth Industrial Revolution, the orchestration and management of numerous fog devices encounter a scalability problem. In fog computing environments, to embrace various types of computation, cloud virtualization technology is widely used. With virtualization technology, IoT and IIoT tasks can be run on virtual machines or containers, which are able to migrate from one machine to another. However, efficient and scalable orchestration of migrations for mobile users and devices in fog computing environments is not an easy task. Naïve or unmanaged migrations may impinge on the reliability of cloud tasks. In this paper, we propose a scalable fog computing orchestration mechanism for reliable cloud task scheduling. The proposed scalable orchestration mechanism considers live migrations of virtual machines and containers for the edge servers to reduce both cloud task failures and suspended time when a device is disconnected due to mobility. The performance evaluation shows that our proposed fog computing orchestration is scalable while preserving the reliability of cloud tasks.

Keywords: fog computing; IoT; IIoT; cloud computing; resource management; scheduling

Citation: Lim, J. Scalable Fog Computing Orchestration for Reliable Cloud Task Scheduling. *Appl. Sci.* **2021**, *11*, 10996. https://doi.org/10.3390/app112210996

Academic Editor: Jinho Kim

Received: 29 September 2021
Accepted: 17 November 2021
Published: 19 November 2021

1. Introduction

Fog computing is a computing architecture that extends cloud computing by enabling data processing at the network edge instead of the central cloud computing server for small and tiny devices, also known as Internet of Things (IoT) devices [1,2]. IoT and Industrial Internet of things (IIoT) devices are increasing by 35% annually, and IoT and IIoT businesses will be worth around USD 7.1 trillion to the United States and EUR 1.2 trillion to Europe by 2030 [3].

To support the ever-increasing number of devices and amount of data generated in IoT and IIoT computing environments, cloud computing provides service offload and virtual machine migration capabilities at the network edge level (edge server) [4,5]. However, the inherited characteristics of IoT and IIoT devices hinder reliable cloud task computation [5–7]. In other words, devices in fog computing environments can move around a field, and when a device moves from one spot to another, the device can be disconnected from the edge server [8,9].

In this case, the device that disconnected from the edge server should try to connect to another edge server nearby. If the device cannot access another edge server, the cloud tasks of the device cannot be completed [10–12]. Even when the device is able to access another edge server nearby, the cloud tasks of the device can be suspended until data and virtual machine migrations are complete. This will worsen the reliability of cloud tasks when multiple mobile users perform them simultaneously in IoT and IIoT computing environments [13,14].

Figure 1 shows an IoT and IIoT computing architecture with an edge cloud. There are three layers in the architecture: the central cloud center, edge servers and containers, and IoT applications (tuple). As far as migrations of services and applications are concerned,

edge servers should cooperate with the central cloud center and other nearby edge servers. When unconsidered scheduling and migration algorithms trigger migrations, tuples or related data can be lost.

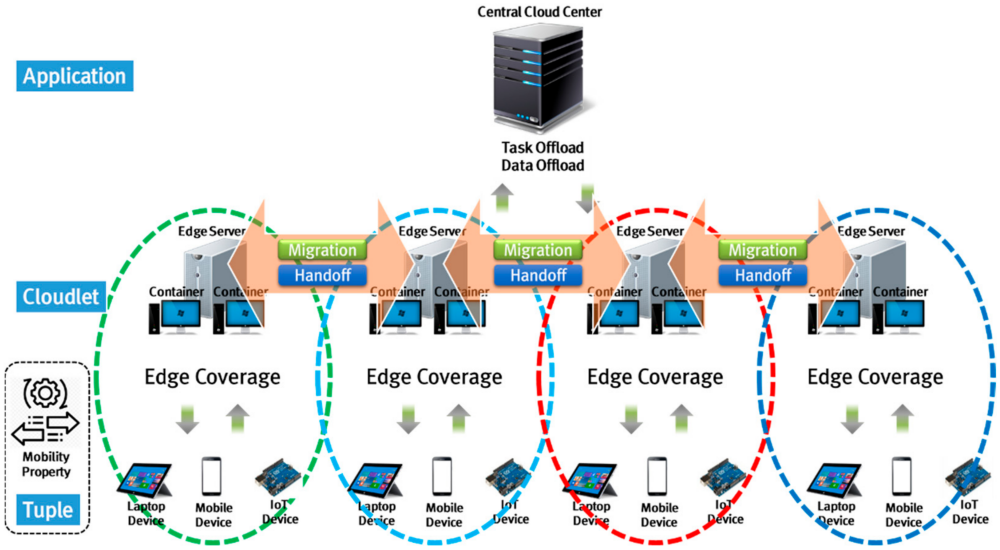


Figure 1. IoT and IIoT computing architecture with an edge cloud.

To mitigate the reliability issues of cloud tasks in IoT and IIoT computing environments, in this paper, we propose a scalable fog computing orchestration mechanism for reliable cloud task scheduling. The proposed scalable orchestration mechanism considers live migrations of virtual machines and containers for the edge servers to reduce both cloud task failures and suspended time when a device is disconnected due to mobility. Furthermore, we use information about the distance between a user and nearby edge servers in order to optimize task scheduling and migration decisions.

Unlike the previous work related to IoT and IIoT cloud task scheduling [15,16], we take users' mobility speed into account so that our scalable fog computing orchestration can select an optimal destination edge server for the migration. A performance evaluation based on realistic mobility data (simulation of urban mobility—SUMO [17,18]) shows that our proposed fog computing orchestration mechanism for a reliable cloud task scheduling algorithm outperforms previous work in terms of the reliability of cloud tasks while introducing affordable data traffic on the network. It also shows that our algorithm scales well in terms of the number of IoT and IIoT users.

The contributions of our study can be summarized as follows.

- We discuss the IoT and IIoT system architectures to show their characteristics and their limitations in supporting cloud-based mobile applications, which will help both architects of IoT and IIoT systems and cloud data center administrators.
- We implement two scalable fog computing orchestration algorithms: one is for scheduling and assigning cloud tasks to appropriate edge servers and containers with considerations of signal strength, and the other one is for cloud task migrations that improve the overall reliability of mobile applications by checking and calculating migratability.
- We validate our proposed algorithms by comparing various performance metrics and incorporating mobility pattern data obtained from SUMO.

The rest of the paper can be summarized as follows. Section 2 discusses our research background and related work. Section 3 proposes our scalable fog computing orchestration algorithm in IoT and IIoT computing environments. Section 4 provides comparative performance evaluation results to show the efficiency and scalability of the proposed mechanism. Finally, Section 5 concludes the paper.

2. Related Work

In this section, we provide a literature review to support the background and motivation of this research. Our scalable fog computing orchestration mechanism is aimed at optimizing the overall IoT and IIoT task performance by improving the reliability of cloud scheduling. The proposed method adopts a live migration scheme [19,20] for mobile users, which generates more network traffic than cold migration [21,22].

For this reason, a few studies have used the live migration scheme in fog computing environments. Despite the network traffic, our fog computing orchestration mechanism is able to improve the reliability of cloud tasks by considering users' mobility and signal strength.

Martins et al. [23] explored virtual network function [24,25] overheads in fog computing environments. Specifically, to predict the container migration cost, they used a linear regression model that is widely used in machine learning and artificial intelligence.

In a container virtualization platform using LXD [26,27], they identified several variables that affected the performance of container migrations. Then, they provided a mathematical analysis in terms of the processing time and network traffic associated with virtual network function migrations.

Since that study was focused on virtual network functions from the perspective of predicting overheads and validating the predictor model, it lacked an overall performance evaluation in fog computing environments, such as realistic scenarios with multiple mobile users.

To support a stateful container migration in a geometrically distributed fog computing environment, another work [28] used an open-source file system (OverlayFS [27]) and took snapshots of containers to reduce network transfer time when migrating.

It achieved a reduced downtime during the container migration when compared to the baseline (no volume checkpoint). Although it proposed enhanced checkpoint and migration techniques, no high-level orchestrations, such as task allocation or scheduling for fog devices, were provided.

Rosário et al. [29] proposed multi-tier (central cloud node, regional fog node, neighborhood fog node, and local fog node) computing node environments in fog computing. Specifically, they supported the quality of experience for multimedia-based mobile applications.

To enhance the quality of experience, the authors designed a service migration from the central cloud server to multi-tier fog computing nodes based on software-defined networking for video distribution.

The potential downside of the method was weak generality. In other words, it was specialized for mobile applications that used multimedia, while our proposed method can be used in general-purpose mobile applications, including multimedia-related functionalities.

Based on the OpenStack platform [30,31], Puliafito et al. [32] proposed a fog computing service. It uses a container-based implementation for service migrations to support the mobility of mobile devices. By implementing a monitoring server for the decision-making mechanism, it calculates the score of the distance and decides on a migration scheduling policy.

Although this was closely related to our study, it differed in that our research uses signal strength instead of distances. In addition, they did not consider multi-user environments for scalability, unlike the proposed evaluation setup.

To employ a load-balancing feature in fog computing environments, Singh et al. [33] analyzed a load-balancing algorithm by considering energy consumption. For the energy-efficient load-balancing mechanism, they considered two main components in fog computing environments: resource (virtual machines) allocation and scheduling.

While the authors analyzed the state-of-the-art load-balancing techniques in fog computing environments, they did not incorporate a container-based resource allocation or scheduling, which is a common acceptance in resource virtualization environments.

Gonçalves et al. [34] proposed a network slicing technique to satisfy the quality of service requirements and improve network utilization in fog computing environments. Two aspects of the proposed techniques were container migration and network slicing.

Among the two, network slicing was the major contribution of the study. In other words, the physical infrastructure of the fog computing network was logically divided into several slices based on the number of users. With the divided slices, it achieved the requirements of a specific group or application. However, the number of mobile users was stationary, and no scalability settings were provided, unlike in the proposed research method.

3. The Proposed Fog Computing Orchestration Mechanism

In this section, we provide our scalable fog computing orchestration algorithms that enhance the reliability of cloud tasks. The proposed algorithms are composed of two parts: a task scheduling algorithm that schedules a new task for IoT and IIoT devices and a task migration algorithm that transfers existing tasks to another edge server and container.

Figure 2 shows the process and flow of the proposed algorithms. There are eight stages in the performance of our proposed orchestration schemes. First, task scheduling is initiated when a new task for IoT and IIoT devices has arrived. Second, when a task is assigned an edge server and its container, the user's mobile application is associated with the edge server. Third, the central cloud center periodically monitors the edge servers and containers in the cloud computing environment.

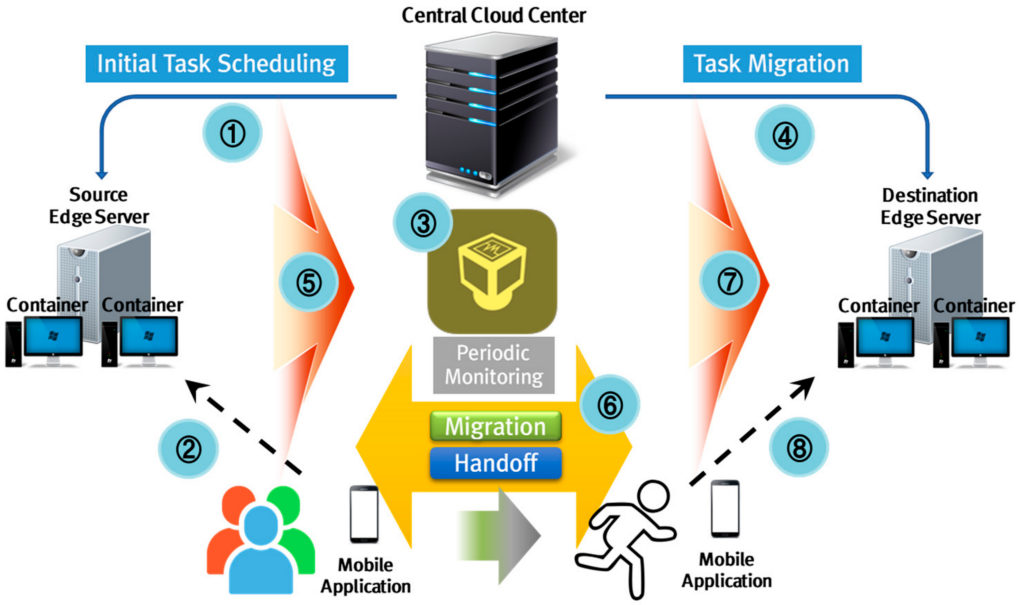


Figure 2. The process and flow of the proposed algorithms (c.f., Algorithms 1 and 2).

Fourth, based on monitoring information, the system may trigger task migration in a reliable fashion. Fifth, the central cloud center collects data and has the target edge server prepare for migrations. Sixth, the source edge server and destination edge server are synchronized for migrations. Seventh, the migration data are transferred from the source edge server to the destination edge server. Finally, the eighth stage is the re-registration of the user's mobile application to the destination edge server, which is transparent to the user.

Algorithm 1 shows the task scheduling algorithm. This algorithm is triggered when a new task for IoT and IIoT devices has arrived. The input is *Tuple_i*, and the output is mapping information for the tuple (*Map_i*), that is, *Tuple_i*, *Edge_j*, and *Container_k*. The algorithm uses location and requirement information based on *Tuple_i*. To compose the

assign_tuple function, several variables are used: *signal_strength*, *target_edge*, *target_container*, and *found*.

It first finds connectable edge servers based on the location. Then, it retrieves the signal strength (*current_signal_strength*) of each edge server by iterating the loop. When *current_signal_strength* is greater than *signal_strength*, the *signal_strength* variable is updated with *current_signal_strength*, and that edge server is assigned to *target_edge*.

After selecting a target edge server, it traverses each container of the target edge server to check whether *Tuple_i* can be assigned to an existing container. When one of the existing containers meets the requirement, it assigns *Tuple_i* to the container. If it cannot find a container that meets the requirement, it triggers a new container provision for the *target_edge*.

Algorithm 2 shows the task migration algorithm. The input of the algorithm is *Map_i*, and the output is *New_Map_i*. Both *Map_i* and *New_Map_i* contain tuple, edge server, and container information. For the task migration, several variables are initialized (*location*, *signal_strength*, *mobility_speed*, *mobility_property*, *target_edge*, and *bool_migration*).

Before it schedules the task migration, the *check_condition* function is called. Note that the *check_condition* function is called at a regular interval. The *check_condition* function traverses neighbor edge servers of *Edge_i*. After retrieving *signal_target* for *Edge_i*, it calculates the threshold value for the condition. The threshold value is a result of the threshold function with three parameters (*signal_strength*, *location*, and *mobility_speed*). If *signal_target* is greater than the threshold value (*val_threshold*), *Edge_i* is assigned to *target_edge*.

Algorithm 1 Task Scheduling Algorithm.

Input: *Tuple_i*
Output: *Map_i* = (*Tuple_i, Edge_j, Container_k*)
Initialization: *location* ← get_location (*Tuple_i*);
requirement ← get_requirement (*Tuple_i*);
call assign_tuple (*Tuple_i*);
function assign_tuple (*Tuple_i*)
    signal_strength ← 0;
    target_edge ← *null*;
    target_container ← *null*;
    found ← *false*;
    for all *Edge_i* ∈ *Set_Edge* (*location*) **do**
        current_signal_strength ← get_edge_info (*Edge_i, signal*);
        if (*current_signal_strength* > *signal_strength*) **then**
            signal_strength ← *current_signal_strength*;
            target_edge ← *Edge_i*;
        end if
    end for
    for all *Container_i* ∈ *target_edge* **do**
        if (*Container_i* meets *requirement*) **then**
            target_container ← *Container_i*;
            assign_info ← assign_tuple (*Tuple_i*, *target_edge, target_container*);
            found ← *true*;
        end if
    end for
    if (*found* == *false*) **then**
        target_container ← provision_container (*target_edge*);
        assign_info ← assign_tuple (*Tuple_i*, *target_edge, target_container*);
    end if
    return *assign_info*;
end function

Algorithm 2 Task Migration Algorithm.

Input: *Map_i* = (*Tuple_i*, *Edge_j*, *Container_k*)
Output: *New_Map_i* = (*Tuple_i*, *Edge_j*, *Container_k*)
Initialization: *location* ← get_location (*Tuple_i*);
signal_strength ← get_signal_info (*Map_i*);
mobility_speed ← get_mobility_speed (*Map_i*);
mobility_property ← get_mobility_property (*Map_i*);
target_edge ← *null*;
bool_migration ← *false*;

```
call check_condition (Map_i);
if (bool_migration) then
    call perform_migration (Map_i, target_edge);
end if
function check_condition (Map_i)
    signal_target ← 0;
    for all Edge_i ∈ Near_Edge (Map_i) do
        signal_target ← get_edge_info (Edge_i, signal);
        val_threshold ← threshold(signal_strength, location, mobility_speed);
        if (signal_target > val_threshold) then
            target_edge ← Edge_i;
        end if
    end for
    if (migratability (target_edge, mobility_property)) then
        bool_migration ← true;
    end if
end function
function perform_migration (Map_i, target_edge)
    handoff (Map_i, target_edge);
    for all Container_i ∈ target_edge do
        if (Container_i meets requirement) then
            target_container ← Container_i;
            assign_info ← assign_tuple (Tuple_i, target_container);
            found ← true;
        end if
    end for
    if (found == false) then
        target_container ← provision_container (Edge_i);
        assign_info ← assign_tuple (Tuple_i, target_container);
    end if
    migrate_tuple (Map_i, target_container);
    migrate_data (Map_i, target_container);
    sync (Map_i, target_container);
    return New_Map_i = (Tuple_i, target_edge, target_container)
end function
```

With *target_edge* and *mobility_property*, *bool_migration* can be true if the *migratability* function returns true. When *bool_migration* is true after calling the *check_condition* function, the *perform_migration* function is called. The *perform_migration* function does handoff, container selection, and task migration (tuples and data). Note that the container selection phase is similar to that of the task scheduling algorithm. The task migration phase calls three additional functions: *migrate_tuple*, *migrate_data*, and *sync*. Then, it returns *New_Map_i*.

4. Evaluation

In this section, we provide the performance results of our proposed scalable orchestration mechanism for improving migration downtime and task reliability. To evaluate the proposed mechanism in a simulated environment and to measure the network traffic, aver-

age downtime, total migration time, and tuples lost, we use SUMO mobility data, which can be used to simulate mobility patterns and network traffics based on realistic scenarios.

4.1. Performance Results

Figure 3 shows the numbers of migrations of the previous and proposed methods. When the number of users is small (one or two users), the difference between the previous and proposed methods is relatively small. However, as the number of users increases, the difference between the previous and proposed methods becomes greater.

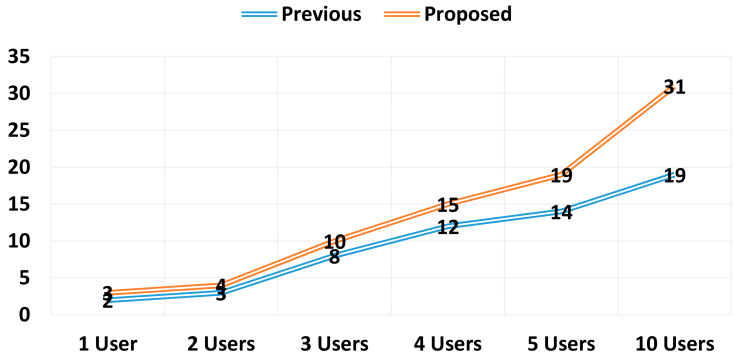


Figure 3. The number of migrations with the number of users.

Specifically, when the number of users is 5, the numbers of migrations of the previous and proposed methods are 14 and 19, respectively, with a difference of 5. If the number of users is 10, the numbers of migrations of the previous and proposed methods are 19 and 31, respectively, with a difference of 12. This signifies that the proposed method optimizes the performance of tasks by migrating them appropriately.

Figure 4 shows the network usage of the previous and proposed methods. Note that the total network usage is the sum of that of the devices and migration, and the baseline data in the figure indicate when the number of users is 1. The network usage of migrations accounts for most of the total network usage, which is affordable in recent high-bandwidth network environments.

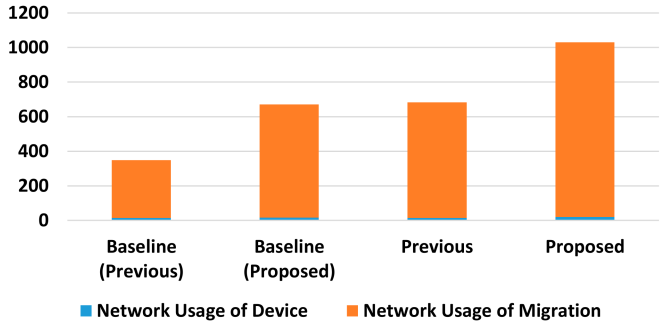
Since the proposed method introduces more live migrations than the previous method, the network usage of the proposed method generates more network traffic than the previous method.

In a multi-user environment (when the number of users is 5), the difference in network usage between the previous and the proposed methods is about 1 GB. Although the proposed method incurs more network traffic, it improves the overall performance in terms of downtime and the reliability of tasks.

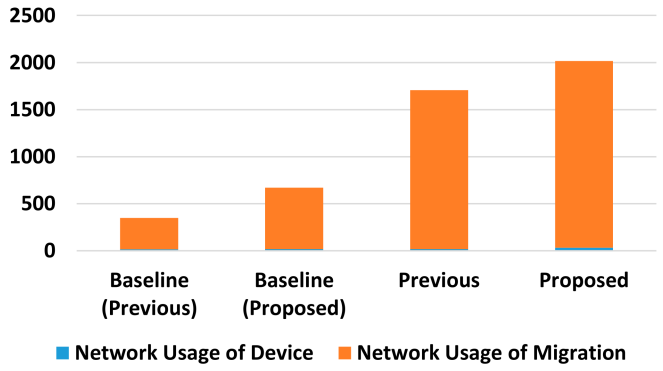
To figure out the performance of time-related metrics, we measure the average downtime and the total time of migration (c.f., Figure 5). As in Figure 4, we refer to the baseline performance in Figure 5 for comparison purposes. In terms of the average downtime, the proposed method outperforms the previous method. Specifically, the proposed method has only about 19% of the downtime seen in the previous method.

This means that our method is more than five times better than the previous method in terms of downtime. When we compare the total time of migration, our method takes longer than the previous method, since our method incurs more live migrations than the previous one. Note that the proposed method does not generate meaningful overhead as the number of users increases.

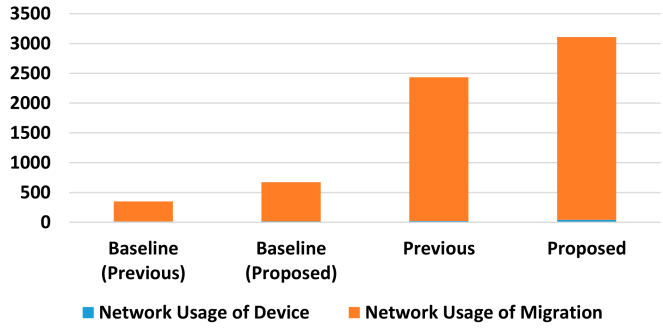
As for the task reliability, we measure the percentages and the numbers of Tuples lost for the previous and proposed methods (c.f., Figure 6). Figure 6a shows the percentages of tuples lost. For the previous method, the percentages of tuples lost are 3.580%, 3.769%, 7.399%, 8.002%, 7.919%, and 6.194% when the numbers of users are 1, 2, 3, 4, 5, and 10, respectively. For the proposed method, the percentages of tuples lost are 0.027%, 0.033%, 0.038%, 0.041%, 0.041%, and 0.041% when the numbers of users are 1, 2, 3, 4, 5, and 10, respectively.



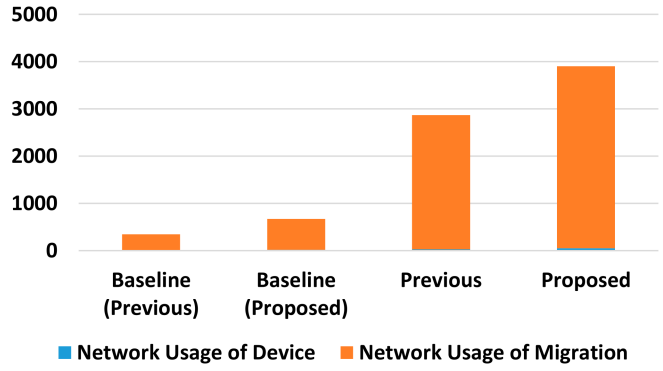
(**a**)



(**b**)



(**c**)



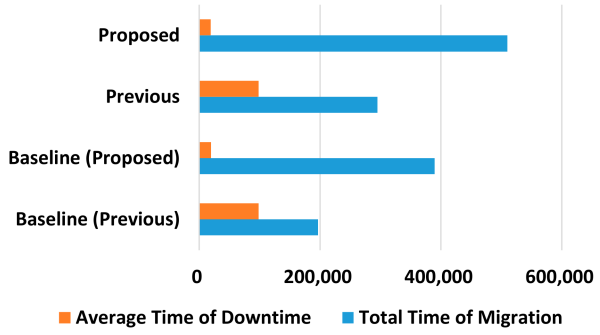
(**d**)

Figure 4. Network traffic usage in MB. (**a**) The number of users: 2. (**b**) The number of users: 3. (**c**) The number of users: 4. (**d**) The number of users: 5.

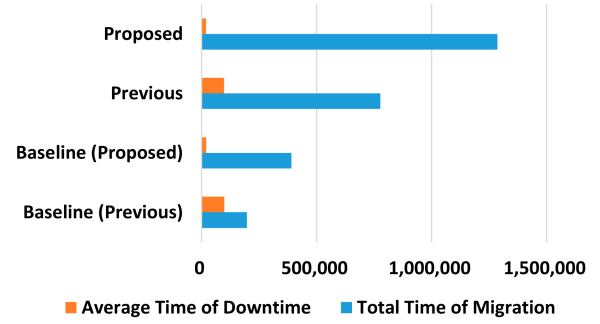
Figure 6b shows the numbers of tuples lost. For the previous method, the numbers of tuples lost are 87,078, 136,802, 376,588, 547,976, 666,795, and 979,050 when the numbers of users are 1, 2, 3, 4, 5, and 10, respectively. For the proposed method, the numbers of tuples lost are 762, 1416, 2637, 4029, 4875, and 13,661 when the numbers of users are 1, 2, 3, 4, 5, and 10, respectively.

Compared to the previous and proposed methods, the proposed method generates 0.875%, 1.035%, 0.700%, 0.735%, 0.731%, and 1.395% of the tuples lost compared with the previous method when the numbers of users are 1, 2, 3, 4, 5, and 10.

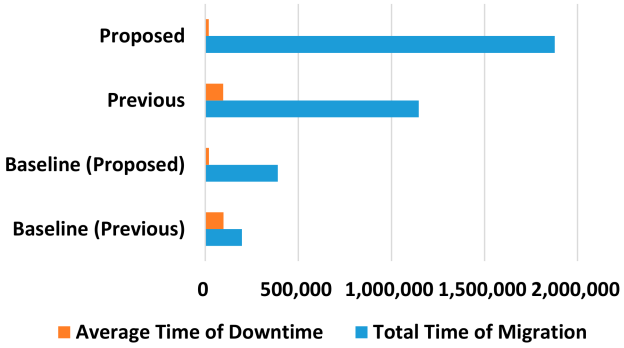
On the whole, the proposed method improves the overall performance for cloud tasks of IoT and IIoT devices in fog computing environments by optimizing the average downtime and the reliability of cloud tasks while generating affordable network traffic for live migrations.



(a)

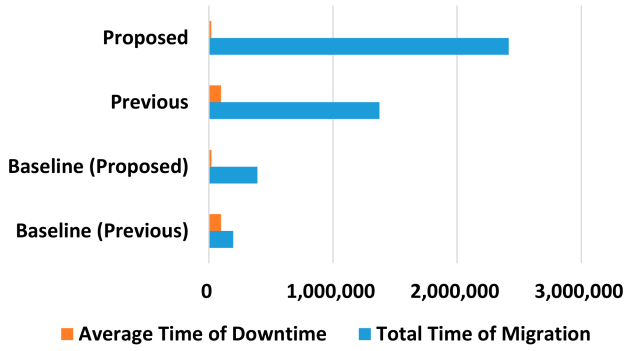


(b)



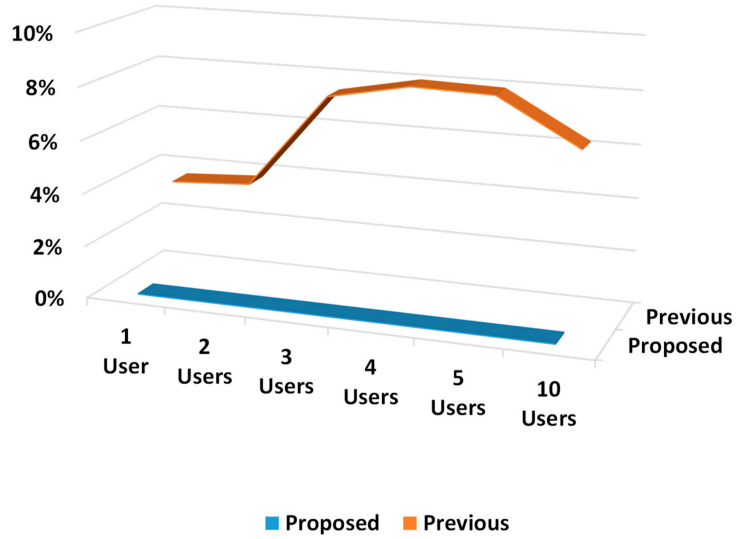
(c)

Figure 5. *Cont.*

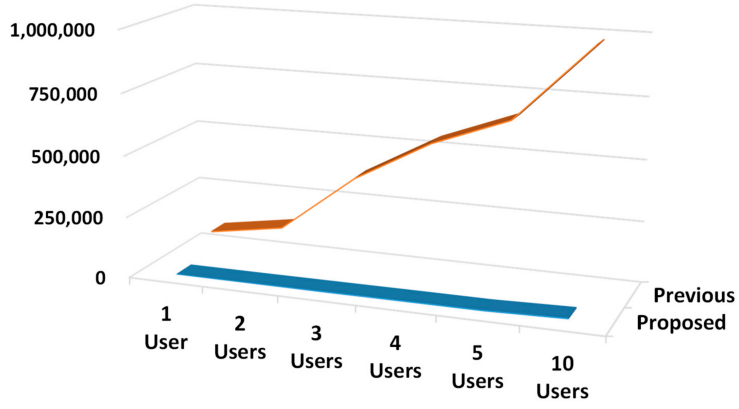


(**d**)

Figure 5. The average downtime and the total time of migration in milliseconds. (**a**) The number of users: 2. (**b**) The number of users: 3. (**c**) The number of users: 4. (**d**) The number of users: 5.



(**a**)



(**b**)

Figure 6. Percentages and numbers of tuples lost. (**a**) Percentages of tuples lost. (**b**) The numbers of tuples lost.

4.2. Discussion

The proposed task scheduling and task migration algorithms are designed to carefully assign and migrate users' tuples in terms of reliability. To this end, we considered various criteria (location, signal strength, mobility speed, and mobility properties) for tuple migrations. Table 1 shows the comparison of cloud task management schemes. Recent studies have shown that the live migration feature in IoT and IIoT computing environments is encouraging, since it helps increase the task throughput and reduces the overall downtime.

Table 1. Comparison of cloud task management schemes. (○: considered, △: partially considered, and ×: not considered).

Study	Scalability	Mobility Support	Live Migration	Migration Criteria	Optimized for Task Reliability
[23]	×	×	○	Migration time Data transferred	×
[28]	○	○	△	Geo-distribution Migration downtime Disk usage	×
[29]	×	○	Service migration	Quality of experience Video distribution	×
[32]	△	○	○	Round-trip latency Migration downtime	×
[34]	△	○	○	User speed Location	△
Proposed	○	○	○	Signal strength Mobility speed and properties	○

However, a few studies thoroughly analyzed and evaluated IoT and IIoT resource management schemes, including live migrations, in terms of scalability and task reliability. For this matter, we developed task scheduling and migration algorithms to improve the scalability and task reliability. Even when the number of users increases, the tuples lost are below 1% on average compared to the previous method.

The downside of the proposed orchestration technique is that it generates more network traffic for live migration data. The network traffic and other performance metrics (downtime and tuples lost) are in a trade-off relationship. To improve the task reliability, we went for the overall performance metrics rather than the network traffic. The performance results show that our scheme introduces more network traffic than the previous method.

To mitigate network overheads, one can use state-of-the-art network chipsets (e.g., WiFi 6 [802.11 ax]). Since WiFi 6 is capable of a maximum throughput speed of 9.6 Gbps, the network transmission time for live migration data can be reduced. We conjecture that as network technology evolves, the network overheads can be drastically reduced.

5. Conclusions

When IoT and IIoT devices' tasks are provisioned to a cloud computing environment, fog computing orchestration may encounter bottleneck problems and cause the unreliability of cloud tasks. In this paper, we proposed a scalable fog computing orchestration mechanism for reliable cloud task scheduling. The proposed scalable orchestration is designed to reduce both cloud task failures and suspended time when a device is disconnected due to mobility. Even when the number of users increases, the tuples lost are below 1% on average compared to the previous method. Specifically, we take various migration criteria (location, signal strength, mobility speed, and mobility properties) into account in our algorithms for task reliability. Although our method introduces more network traffic than the traditional orchestration mechanism, the overall performance, including the downtime and tuples lost, is satisfactory. Since current network environments support high bandwidths (e.g., the fifth-generation mobile network and WiFi 6 [802.11 ax]), the overhead of additional network traffic is negligible. Future work will include the adoption

of artificial intelligence techniques to manage and schedule virtualized resources in IoT and IIoT computing environments.

Funding: This research was supported by a grant from the National Research Foundation of Korea funded by the Korean Government (NRF-2021R1F1A1063307).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Acknowledgments: We thank the anonymous reviewers for their careful reading and insightful suggestions to help improve the manuscript.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Gadasin, D.V.; Shvedov, A.V.; Ermolovich, A.V. The concept "fog computing"—The evolutionary stage of development of infocommunication technologies. In Proceedings of the 2018 Systems of Signals Generating and Processing in the Field of on Board Communications, Moscow, Russia, 14–15 March 2018; pp. 1–3.
2. Karagiannis, V.; Schulte, S. Comparison of Alternative Architectures in Fog Computing. In Proceedings of the 2020 IEEE 4th International Conference on Fog and Edge Computing (ICFEC), Melbourne, Australia, 11–14 May 2020; pp. 19–28.
3. Tange, K.; De Donno, M.; Fafoutis, X.; Dragoni, N. A Systematic Survey of Industrial Internet of Things Security: Requirements and Fog Computing Opportunities. *IEEE Commun. Surv. Tutorials* **2020**, *22*, 2489–2520. [CrossRef]
4. Mostafa, G.-A.; Alireza, S.; Rahmanian, A.A. Resource Management Approaches in Fog Computing: A Comprehensive Review. *J. Grid Comput.* **2020**, *18*, 1–42. [CrossRef]
5. Joshi, V.; Patil, K. *A Survey on Energy-Efficient Task Offloading and Virtual Machine Migration for Mobile Edge Computation BT—Data Management, Analytics and Innovation*; Sharma, N., Chakrabarti, A., Balas, V.E., Bruckstein, A.M., Eds.; Springer: Singapore, 2022; pp. 333–347.
6. Paniagua, C.; Delsing, J. Industrial Frameworks for Internet of Things: A Survey. *IEEE Syst. J.* **2021**, *15*, 1149–1159. [CrossRef]
7. Liu, P.; Liu, K.; Fu, T.; Zhang, Y.; Hu, J. A privacy-preserving resource trading scheme for Cloud Manufacturing with edge-PLCs in IIoT. *J. Syst. Archit.* **2021**, *117*, 102104. [CrossRef]
8. Kabbaj, S.; Rahman, A.U.; Malik, A.W.; Baba, A.I.; Ravana, S.D. Time-bound single-path opportunistic forwarding in disconnected industrial environments. *Veh. Commun.* **2021**, *27*, 100302. [CrossRef]
9. Chen, P.-Y.; Bhatia, L.; Kolcun, R.; Boyle, D.; McCann, J.A. Contact-Aware Opportunistic Data Forwarding in Disconnected LoRaWAN Mobile Networks. In Proceedings of the 2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS), Singapore, 29 November–1 December 2020; pp. 574–583.
10. Ma, S.; Song, S.; Yang, L.; Zhao, J.; Yang, F.; Zhai, L. Dependent tasks offloading based on particle swarm optimization algorithm in multi-access edge computing. *Appl. Soft Comput.* **2021**, *112*, 107790. [CrossRef]
11. Wang, X.; Ning, Z.; Guo, S. Multi-Agent Imitation Learning for Pervasive Edge Computing: A Decentralized Computation Offloading Algorithm. *IEEE Trans. Parallel Distrib. Syst.* **2021**, *32*, 411–425. [CrossRef]
12. Chen, S.; Li, Q.; Zhou, M.; Abusorrah, A. Recent Advances in Collaborative Scheduling of Computing Tasks in an Edge Computing Paradigm. *Sensors* **2021**, *21*, 779. [CrossRef]
13. Rejiba, Z.; Masip-Bruin, X.; Marín-Tordera, E. A survey on mobility-induced service migration in the fog, edge, and related computing paradigms. *ACM Comput. Surv.* **2019**, *52*, 1–33. [CrossRef]
14. Zhang, G.; Ni, S.; Zhao, P. Learning-based Joint Optimization of Energy-Delay and Privacy in Multiple-User Edge-Cloud Collaboration MEC Systems. *IEEE Internet Things J.* **2021**, *1*, 8607. [CrossRef]
15. Shakarami, A.; Ghobaei-Arani, M.; Masdari, M.; Hosseinzadeh, M. A Survey on the Computation Offloading Approaches in Mobile Edge/Cloud Computing Environment: A Stochastic-based Perspective. *J. Grid Comput.* **2020**, *18*, 639–671. [CrossRef]
16. Chen, S.; Zheng, Y.; Wang, K.; Lu, W. Delay Guaranteed Energy-Efficient Computation Offloading for Industrial IoT in Fog Computing. In Proceedings of the ICC 2019—2019 IEEE International Conference on Communications (ICC), Shanghai, China, 20–24 May 2019; pp. 1–6.
17. Krajzewicz, D. *Traffic Simulation with SUMO—Simulation of Urban Mobility BT—Fundamentals of Traffic Simulation*; Barceló, J., Ed.; Springer: New York, NY, USA, 2010; pp. 269–293. ISBN 978-1-4419-6142-6.
18. Acosta, A.F.; Espinosa, J.E.; Espinosa, J. *Application of the SCRUM Software Methodology for Extending Simulation of Urban Mobility (SUMO) Tools BT—Simulating Urban Traffic Scenarios*; Behrisch, M., Weber, M., Eds.; Springer International Publishing: Cham, Switzerland, 2019; pp. 3–15.
19. Roig, P.J.; Alcaraz, S.; Gilly, K.; Juiz, C. Modelling VM Migration in a Fog Computing Environment. *Elektron. Elektrotechnika* **2019**, *25*, 75–81. [CrossRef]
20. Osanaiye, O.; Chen, S.; Yan, Z.; Lu, R.; Choo, K.-K.R.; Dlodlo, M. From Cloud to Fog Computing: A Review and a Conceptual Live VM Migration Framework. *IEEE Access* **2017**, *5*, 8284–8300. [CrossRef]

21. Puliafito, C.; Virdis, A.; Mingozzi, E. The Impact of Container Migration on Fog Services as Perceived by Mobile Things. In Proceedings of the 2020 IEEE International Conference on Smart Computing (SMARTCOMP), Bologna, Italy, 14–17 September 2020; pp. 9–16.
22. Puliafito, C.; Vallati, C.; Mingozzi, E.; Merlino, G.; Longo, F.; Puliafito, A. Container Migration in the Fog: A Performance Evaluation. *Sensors* **2019**, *19*, 1488. [CrossRef]
23. de Martins, R.J.; Both, C.B.; Wickboldt, J.A.; Granville, L.Z. Virtual Network Functions Migration Cost: From Identification to Prediction. *Comput. Netw.* **2020**, *181*, 107429. [CrossRef]
24. Ponmagal, R.S.; Karthick, S.; Dhiyanesh, B.; Balakrishnan, S.; Venkatachalam, K. Optimized virtual network function provisioning technique for mobile edge cloud computing. *J. Ambient Intell. Humaniz. Comput.* **2021**, *12*, 5807–5815. [CrossRef]
25. Yu, H.; Yang, J.; Fung, C. Fine-Grained Cloud Resource Provisioning for Virtual Network Function. *IEEE Trans. Netw. Serv. Manag.* **2020**, *17*, 1363–1376. [CrossRef]
26. Ma, J.; Kim, H.; Kim, Y. The Virtualization and Performance Comparison with LXC-LXD in ARM64bit Server. In Proceedings of the 2016 6th International Conference on IT Convergence and Security (ICITCS), Prague, Czech Republic, 26–29 September 2016; pp. 1–4.
27. da Cunha, H.G.V.O.; Moreira, R.; de Oliveira Silva, F. *A Comparative Study Between Containerization and Full-Virtualization of Virtualized Everything Functions in Edge Computing BT—Advanced Information Networking and Applications*; Barolli, L., Woungang, I., Enokido, T., Eds.; Springer International Publishing: Cham, Switzerland, 2021; pp. 771–782.
28. Junior, P.S.; Miorandi, D.; Pierre, G. Stateful Container Migration in Geo-Distributed Environments. In Proceedings of the 2020 IEEE International Conference on Cloud Computing Technology and Science (CloudCom), Bangkok, Thailand, 14–17 December 2020; pp. 49–56.
29. Rosário, D.; Schimuneck, M.; Camargo, J.; Nobre, J.; Both, C.; Rochol, J.; Gerla, M. Service Migration from Cloud to Multi-tier Fog Nodes for Multimedia Dissemination with QoE Support. *Sensors* **2018**, *18*, 329. [CrossRef]
30. Kristiani, E.; Yang, C.-T.; Huang, C.-Y.; Wang, Y.-T.; Ko, P.-C. The Implementation of a Cloud-Edge Computing Architecture Using OpenStack and Kubernetes for Air Quality Monitoring Application. *Mob. Netw. Appl.* **2021**, *26*, 1070–1092. [CrossRef]
31. Benomar, Z.; Longo, F.; Merlino, G.; Puliafito, A. Cloud-Based Network Virtualization in IoT with OpenStack. *ACM Trans. Internet Technol.* **2021**, *22*, 1–26. [CrossRef]
32. Puliafito, C.; Vallati, C.; Mingozzi, E.; Merlino, G.; Longo, F. Design and evaluation of a fog platform supporting device mobility through container migration. *Pervasive Mob. Comput.* **2021**, *74*, 101415. [CrossRef]
33. Singh, S.P.; Kumar, R.; Sharma, A.; Nayyar, A. Leveraging energy-efficient load balancing algorithms in fog computing. *Concurr. Comput. Pract. Exp.* **2020**, e5913. [CrossRef]
34. Gonçalves, D.; Puliafito, C.; Mingozzi, E.; Rana, O.; Bittencourt, L.; Madeira, E. Dynamic Network Slicing in Fog Computing for Mobile Users in MobFogSim. In Proceedings of the 2020 IEEE/ACM 13th International Conference on Utility and Cloud Computing (UCC), Leicester, UK, 7–10 December 2020; pp. 237–246.