


Article

Latency-Classification-Based Deadline-Aware Task Offloading Algorithm in Mobile Edge Computing Environments

HeeSeok Choi ¹ , Heonchang Yu ¹ and EunYoung Lee ^{2,*}¹ Department of Computer Science and Engineering, Korea University, Seoul 02841, Korea; hsrangken@korea.ac.kr (H.C.); yuhc@korea.ac.kr (H.Y.)² Department of Computer Science, Dongduk Women's University, Seoul 02748, Korea

* Correspondence: elee@dongduk.ac.kr; Tel.: +82-2-940-4588

Received: 31 August 2019; Accepted: 1 November 2019; Published: 4 November 2019



Abstract: In this study, we consider an edge cloud server in which a lightweight server is placed near a user device for the rapid processing and storage of large amounts of data. For the edge cloud server, we propose a latency classification algorithm based on deadlines and urgency levels (i.e., latency-sensitive and latency-tolerant). Furthermore, we design a task offloading algorithm to reduce the execution time of latency-sensitive tasks without violating deadlines. Unlike prior studies on task offloading or scheduling that have applied no deadlines or task-based deadlines, we focus on a comprehensive deadline-aware task scheduling scheme that performs task offloading by considering the real-time properties of latency-sensitive tasks. Specifically, when a task is offloaded to the edge cloud server due to a lack of resources on the user device, services could be provided without delay by offloading latency-tolerant tasks first, which are presumed to perform relatively important functions. When offloading a task, the type of the task, weight of the task, task size, estimated execution time, and offloading time are considered. By distributing and offloading latency-sensitive tasks as much as possible, the performance degradation of the system can be minimized. Based on experimental performance evaluations, we prove that our latency-based task offloading algorithm achieves a significant execution time reduction compared to previous solutions without incurring deadline violations. Unlike existing research, we applied delays with various network types in the MEC (mobile edge computing) environment for verification, and the experimental result was measured not only by the total response time but also by the cause of the task failure rate.

Keywords: task offloading; mobile edge computing; latency-classification; latency-aware

1. Introduction

Based on the rapid development of the Internet of Things (IoT) technology in various industrial fields, billions of mobile systems for smart cities, autonomous vehicles, artificial intelligence, IoT gateways, and augmented reality now demand computational resources to handle large amounts of data and communication networks to connect large numbers of devices [1]. Consequently, the data generated by various devices connected to the Internet is growing exponentially, and it has become necessary to process and store a large amount of data rapidly. Existing client-server environments and centralized cloud computing technology have limitations in terms of large-scale data processing; however, recently, mobile edge computing (MEC) technology has emerged to handle this issue.

By applying distributed computing technology to wireless base stations, MEC technology can dramatically reduce delay time and overall network traffic: in MEC, various services are provided with caching content at the wireless base station closest to a target user device. A typical MEC framework is

composed of many mobile devices, a smaller number of edge cloud servers, and one central cloud server [2]. For example, when a user executes a task on his/her device, if the processing power of the device is insufficient to perform the task, it is offloaded to an edge server. If the processing power on the edge cloud server is not sufficient enough, the task is offloaded to the central cloud server. This system provides applications with ultra-low latency, high-bandwidth, and real-time access to network information. Currently, standardization of MEC is being performed by the European Telecommunications Standards Institute, while the industrial world is attempting to develop long-term evolution in mobile networks. Additionally, MEC technology is emerging as the core technology for 5G networks that will be introduced within the next decade, with software-defined networks and network function virtualization as auxiliary technologies [3].

Although research on MEC has been very active, there are certain limitations in handling real-time services requiring strict adherence to response time standards [4]. In general, an application service can be classified as a latency-sensitive service or a latency-tolerant service according to the urgency level of task end-time violation [5–8]. A service ought to be distinguished as a latency-sensitive service that must adhere to deadlines or a latency-tolerant service that may be allowed to violate a deadline by an arbitrary amount. In latency-sensitive services, such as autonomous driving services, one second of delay could have serious consequences—the service must be terminated within a deadline. However, in current MEC systems, it is not possible to guarantee the task-end time of services with real-time properties because resources are allocated to all virtual machines without considering the characteristics of these applications. For example, assume that a latency-sensitive service and a latency-tolerant service are running on the same server. If computing resources are allocated to the virtual machine performing the latency-tolerant service first, the latency-sensitive service will not be guaranteed to meet the task deadline. Additionally, if the tasks of latency-sensitive services, whose deadline is close, are offloaded without considering those deadlines, a deadline violation will be incurred, and the service's real-time availability will not be guaranteed.

In this paper, we propose a novel latency-classification-based deadline-aware task offloading algorithm (LCDA) for MEC environments to solve the problems discussed above. The MEC environments of the proposed algorithm and the performance evaluation are as follows. The devices (users) randomly move the path and upload and download data by accessing the AP of one of the edge cloud servers in a WLAN (wireless local area network). These edge cloud servers communicate with each other in a MAN (metropolitan area networks), and an edge cloud cluster and a central cloud server communicate through a WAN (wide area networks) network. The connection with the base station provided by the ISP is not assumed. For the evaluation, we measured the delay by network type (i.e., WLAN, MAN, and WAN) and considered the failure caused by the user's mobility and the excessive CPU usage of the node—these elaborations of the experiments proved the performance improvement of our proposed algorithm more accurately. We will explain the details of the experiments and discuss the result in detail in the later part of the paper.

The proposed method is based on task classification, in which we divide cloud tasks into two categories: latency-sensitive and latency-tolerant tasks. A latency-sensitive task, such as augmented reality tasks or autonomous vehicle tasks, must be completed within a given deadline. A latency-tolerant task is a task that can tolerate a certain level of deadline violation. Generally, a task is classified based on its real-time properties, and the proposed approach consists of two main procedures. First, when we need to offload tasks based on an offloading policy, we prefer latency-tolerant tasks over latency-sensitive tasks because the performance degradation incurred by offloading can lead to deadline violation of latency-sensitive tasks. Second, we develop a task offloading scheduling mechanism that provides fine-grained priority control for selecting tasks to be offloaded and selecting destination edge cloud servers for offloading.

Our work has strengths over traditional task scheduling algorithms in that we design and implement a novel offloading mechanism based on a task classification approach. The major contributions of this study can be summarized as follows:

- We designed a latency-aware task classification mechanism based on the urgency level of tasks and the avoidance level of deadline violation.
- We developed task scheduling and target server selection algorithms that solve problems regarding when and how tasks should be offloaded in a manner which minimizes the delay.
- We conducted extensive simulations to evaluate the performance of the proposed LCDA. The results demonstrate that the LCDA not only achieves a significant reduction in the delay of latency-sensitive tasks compared to previous algorithms, but also guarantees resource efficiency, maximum profit, and the preservation of real-time attributes.
- We proved the effectiveness of our technique by separating the cause of the task failure into three factors: CPU capacity, user mobility, and network delays. This classification results in a more accurate evaluation compared to the evaluations of prior researches, which use only performance measure: delay time.

The remainder of this paper is organized as follows. The motivation and the core concepts of this study are discussed in Section 2. The proposed LCDA is detailed in Section 3. Performance evaluation and related work are presented in Sections 4 and 5, respectively. Finally, our conclusions are presented in Section 6.

2. Motivation and Core Concepts

Based on the continued development of IoT technology, a number of applications, such as health monitoring applications, are emerging that require feedback within a short time slot. These applications are difficult for mobile devices to handle; therefore, MEC technology has emerged as a solution.

Figure 1 shows the task failure rate and processing time for three cases: (1) using only mobile devices, (2) using only edge devices, and (3) using a built MEC environment.

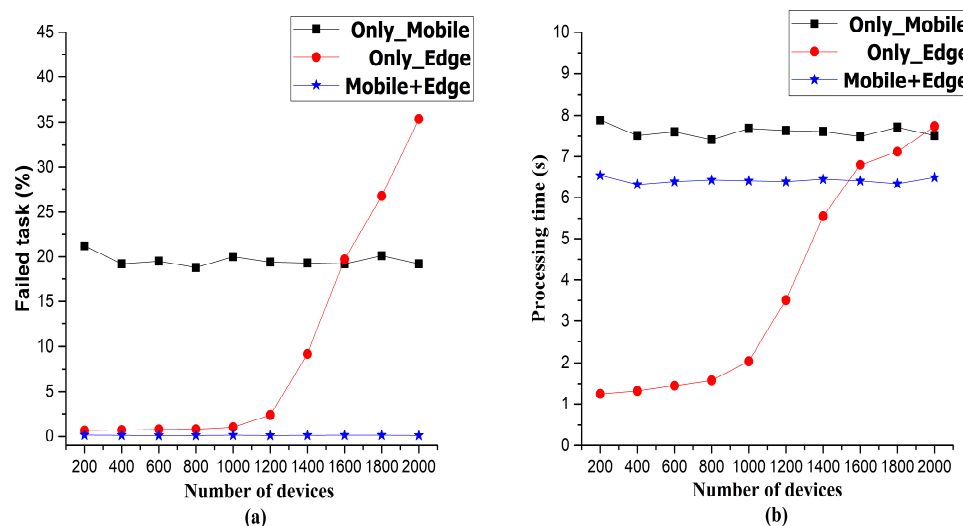


Figure 1. Task failure rate and processing time in the MEC (mobile edge computing) environment. (a) Percentage of failed tasks; (b) Processing time.

The failed tasks represent the percentage of failed tasks out of the total tasks, and the processing time represents the average service time minus the average network delays. We divided failure into two causes. First, the CPU capacity of the user device and the edge cloud server is exceeded. Second, the task cannot be completed due to user mobility or network (WLAN/MAN/WAN) delay. For the sake of simplicity, we use the term “failed task” in the paper. The processing time is derived from the following formula: processing time + network delay = service time. If the operation is performed in Only_Mobile or Only_Edge, since no network delay occurs, the processing time and service time are the same. In the case of Mobile + Edge, the actual service time minus WLAN or MAN delay is

processing time because the task is offloaded to the edge cloud server. If the task is offloaded to the central cloud server, the actual service time minus WAN delay becomes the processing time. Moreover, we operate the latency-tolerant tasks and latency-sensitivity tasks at a ratio of 5:5 in this experiment.

The deployment and use of a MEC environment are advantageous in terms of task failure rate and processing time. From the analysis of the task failure rate, it is observed that task failures occur due to the lack of resources in the case where only mobiles are used. When using only edge devices, the task failures occur due to the frequent movement of mobile devices. In particular, when the number of users (devices) exceeds 1000, the performance of using only edge devices sharply deteriorates. The MEC environment complements these shortcomings, and the task failure rate is relatively low. However, the MEC environment uses offloading techniques to offload tasks, which may slow down the processing time compared to using mobile devices alone. When offloading occurs, the latency caused by sending a task to an edge cloud server and returning a response from the server can have strongly negative effects. In addition, if a task of such performance degradation is latency-sensitive, such as autonomous driving services, the damage suffered by the user would be profound. Therefore, we need a solution that can guarantee rapid processing for latency-sensitive tasks.

We devised an approach to classify tasks and offload latency-tolerant tasks first to reduce the deadline violation incurred by offloading, as shown in Figure 2. Throughout this paper, we assume that each virtual machine (VM) has only one task. Once offloading is decided, the tasks of the applications on a user device are classified as latency-tolerant and latency-sensitive tasks. The proposed LCDA then decides which tasks to be offloaded.

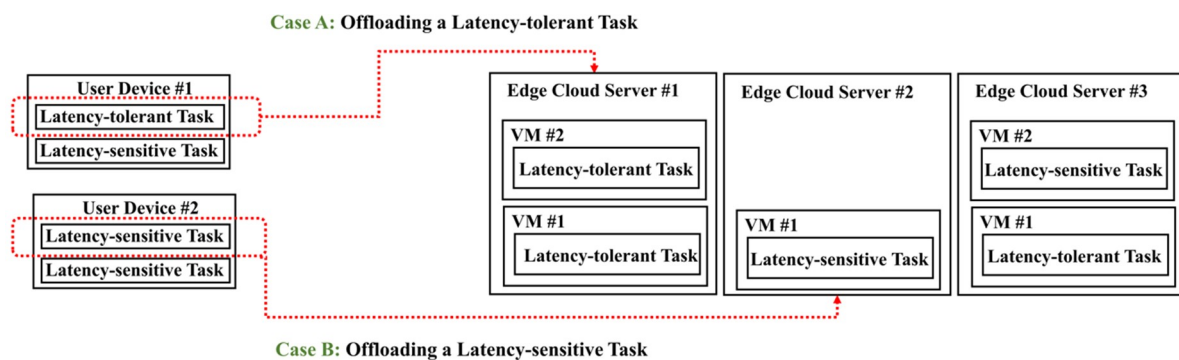


Figure 2. An illustrative example of the LCDA (latency classification based deadline aware task offloading algorithm).

For Case A, in Figure 2, where a latency-tolerant task is selected for offloading, our offloading algorithm chooses the edge cloud server with the smallest number of latency-sensitive VMs. Offloading a latency-tolerant task minimizes any adverse effects on latency-sensitive tasks that are currently running on the edge cloud server. If there are many servers with the same number of latency-sensitive VMs, we consider CPU utilization and the distance to the user device as secondary metrics.

For Case B, where a latency-sensitive task is selected for offloading, our offloading algorithm chooses the server with the lowest CPU utilization. It is not important whether the currently running task on the selected server is latency-sensitive or latency-tolerant: the rapid processing of offloaded tasks is the most important factor. If there are multiple servers with the same CPU utilization, we select the server with the smallest number and size of active VMs.

3. Latency-Classification-Based Deadline-Aware Task Offloading Algorithm (LCDA)

As shown in Figure 3, we consider a typical MEC environment consisting of a user device, an edge cloud server, and a central cloud server. Typically, a user executes tasks on his/her device and offloads tasks to the edge server in accordance with an offloading policy.

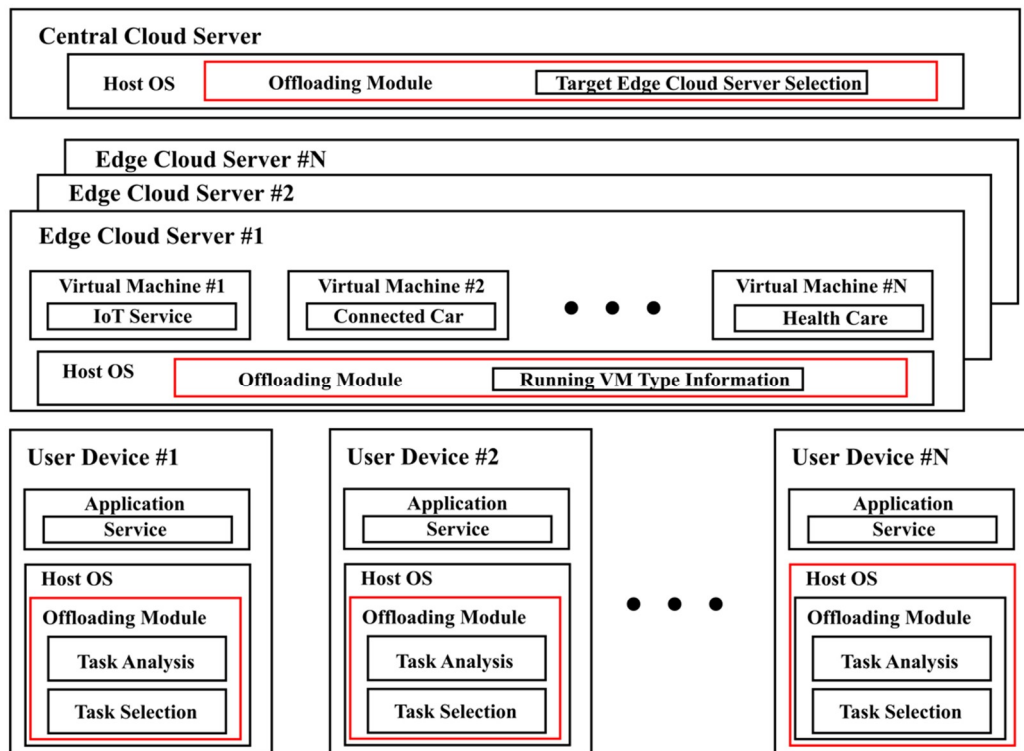


Figure 3. System architecture of the LCDA.

When a task running on a user device is offloaded to an edge cloud server due to a lack of resources, the LCDA performs a task classification process based on latency level. A task is either categorized as latency-sensitive or latency-tolerant. Next, by considering the weight values of tasks, we offload latency-tolerant tasks preferentially. The LCDA periodically checks the status of a given task and the utilization of server resources. The detailed descriptions of proposed algorithms' components are provided below.

3.1. Task Analysis

Unlike previous studies, we consider delay tolerance along with the deadline of a task when performing task offloading. We implement a task analysis function to categorize tasks as latency-sensitive or latency-tolerant tasks on a user device. Before offloading occurs, the task is classified as a latency-sensitive or latency-tolerant task according to the delay tolerance of applications derived by the task analysis function. The task classification is performed based on the user device's load profile or a task's characteristics. If a task is demanding rapid and accurate results or judged to be an urgent task, the task is classified as latency-sensitive. Otherwise, the task is classified as a latency-tolerant. Many research papers classify task types using techniques to identify and classify task characteristics [5–8]. We define the types of tasks running on a user device, as in Equation (1).

$$\text{taskType}(t_i, w_j) = \begin{cases} \text{latency-sensitive,} & \text{Deadline} < \text{Threshold}^{\text{Urgency}} \text{ and } \text{EET} \geq \text{Deadline} \\ \text{latency-tolerant,} & \text{Deadline} \geq \text{Threshold}^{\text{Urgency}} \text{ and } \text{EET} < \text{Deadline} \end{cases} \quad (1)$$

where $\text{taskType}(t, w)$ denotes the type of task (i) at the workflow (j), and the deadline is the value initially set for the task. In practice, when a user rents a virtual machine to use an edge cloud server in the MEC environment, it creates a request and then submits the request to the platform. The request includes the user's maximum allowable latency (i.e., deadline), the size of the input task, and the number of resources required by the virtual machine [2]. We use the deadline value of these. The threshold of

urgency is arbitrary values set by the user, and EET denotes the estimated average execution time of a task. This task analysis function examines a task's characteristics and places the task in the corresponding queue. In other words, if a task has urgency properties, then the task analysis function places the task in the latency-sensitive queue. Otherwise, the task is placed in the latency-tolerant queue. In addition, the type of task can be set directly by the user before the user sends a task to an edge cloud server.

After classifying a task, we calculate the weight of each task to determine the priority for offloading in the subsequent task selection function. In general, the deadline for a task is primarily taken into account when using the workflow for task management in the cloud [9], and the criteria for calculating the weight of a task in the task analysis function are based on recursive techniques [10]. We calculate the weight for the task selection for later offloading, as defined in Equation (2).

$$taskTimeWeight_{t_i}^{w_j} = \frac{D_j - SD(t_i)}{D_j} \quad (2)$$

where *taskTimeWeight* denotes the weight of task (*i*) at the workflow (*j*), *D* denotes the deadline, the user-defined time constraint for each workflow, and *SD* is short for a sub-deadline, denoting the estimated average execution time of each task in the workflow. *SD* is calculated from the execution time of each task recursively from the last exit task of the workflow to the first entry task. Finally, Equation (2) determines the weight of an offloaded task: the smaller is the weight value, the higher priority is assigned to the task. This calculation is performed in the task analysis function in the user device but is executed only when the number of the original tasks in the user device changes by more than 20% in order to reduce the burden of the task analysis operation [11]. The classification information and weight value of tasks are stored for future use.

3.2. Task Selection for Offloading

Selecting tasks to be offloaded to an edge cloud server has a significant impact on overall system performance. This is because offloading an inappropriate task will result in a severe offloading time penalty and will not guarantee the satisfaction of the deadline of a latency-sensitive task. As mentioned previously, when offloading a latency-sensitive task, degradation will occur, and the deadline for the task may not be satisfied. Therefore, the core concept of task selection is to offload tasks in the latency-tolerant queue preferentially, as shown in Algorithm 1.

When selecting a task for offloading, only tasks in the latency-tolerant queue can be candidates. The task with the lowest weight calculated by task analysis function in the latency-tolerant queue is given the highest priority for offloading. This is done to minimize the impact of the performance degradation that is incurred when a task is offloaded on the execution time of any current tasks. If there are multiple tasks with the same weight value, the task with a smaller task size is selected preferentially.

If there are only a few remaining deadlines to be satisfied or if it is determined that offloading a task is more damaging than waiting to execute it on the current user device, then we conclude that there are no appropriate tasks for offloading in the latency-tolerant queue. In this case, or in the case where a user device has only latency-sensitive tasks, we search for tasks to be offloaded in the latency-sensitive queue. In such cases, tasks with the lowest weight calculated by task analysis are selected for offloading. If there are multiple tasks with the same weight value, the smallest task is selected for offloading. Finally, if it is determined that there are no appropriate tasks for offloading in the latency-sensitive queue, then the device will conclude that it is not possible to offload any tasks.

Algorithm 1. *Task_Selection ()*

```

1: for all  $task_i$  in  $task_{latency-tolerant}$ ,  $\forall_i \in \{1, 2, \dots, n\}$ ;
2:   Find the  $task_i$  with lowest weight calculated by task analysis;
3:   if There are tasks with weight of the same value
4:     Find the  $task_i$  with the smallest size;
5:   end if
6:   if 'offloading time of task is greater than local execution time of task' |
       'remaining deadline is lower than threshold'
7:     for all  $task_i$  in  $task_{latency-sensitive}$ ,  $\forall_i \in \{1, 2, \dots, n\}$ ;
8:       Find the  $task_i$  with lowest weight calculated by task analysis;
9:       if There are tasks with weight of the same value
10:        Find the  $task_i$  with the smallest size;
11:      end if
12:      if 'offloading time of task is greater than local execution time of task' |
           'remaining deadline is lower than threshold'
13:        Continue to run on the current user device;
14:      end if
15:    end for
16:  end if
17: end for

```

3.3. Target Edge Cloud Server Selection

After selecting a task to be offloaded, the next step is to select an appropriate edge cloud server to offload that task, as shown in Algorithm 2. To find an edge cloud server for offloading a task, we apply an edge-cloud server selection algorithm differently based on the task type information from the task analysis.

First, if the type of task to be offloaded is latency-tolerant, we select the edge cloud server with the smallest number of latency-sensitive VMs for offloading. This is done to minimize interference on edge cloud servers running latency-sensitive VMs when a task is offloaded. If multiple edge cloud servers running the same number of latency-sensitive VMs are identified, we select the edge cloud server with the lowest CPU utilization. In other words, the server with the most available resources is always selected. If there are edge cloud servers with the same utilization, the server closest to the user device is selected. This is consistent with the concept of "data locality," which means that a VM and the disk to be accessed by that VM are placed physically close to each other to improve the performance of cloud computing. This minimizes network congestion and increases the overall throughput of a system. If there is no edge cloud server with sufficient resources in the edge pool, the central cloud server with the most resources in the cloud pool is selected.

Second, if the type of the task to be offloaded is latency-sensitive, we do not consider different types of VMs when selecting an edge cloud server. Latency-sensitive tasks are always offloaded into the edge cloud server with the greatest available resources without considering the task state of the target server. It is because processing the offloaded latency-sensitive task as quickly as possible is most important. In such cases, the server with the lowest CPU utilization is selected for offloading. If there are multiple servers with the same CPU utilization, the server with the smallest number and size of running VMs is selected as the final target. If all of these conditions are not met, the task cannot be offloaded, and execution continues on the user device. This decision is to minimize the performance degradation of the latency-sensitive tasks during offloading.

Algorithm 2. *Server_Selection ()*

```

1: for all  $server_i$  in  $edge_{pool}$ ,  $\forall_i \in \{1, 2, \dots, n\}$ ;
2:   if the type of task to be offloaded is latency-tolerant
3:     Find the  $server_i$  with the smallest number of latency-sensitive VMs;
4:     if There are servers with the same number of latency-sensitive VMs
5:       Find the  $server_i$  with the lowest CPU utilization;
6:       if There are  $server_i$  with the same utilization
7:         Find the  $server_i$  closest to the user device;
8:       end if
9:     end if
10:  else // the type of task to be offloaded is latency-sensitive
11:    Find the  $server_i$  with the lowest CPU utilization;
12:    if There are multiple servers with the same CPU utilization
13:      Find the  $server_j$  with the smallest number and size of running VMs;
14:    end if
15:  end if
16: end for

```

3.4. Latency Classification Based on Deadline-Aware Task Offloading Algorithm

We combine the task analysis, task selection, and server selection functions described above to create the proposed LCDA. Algorithm 3 represents the complete offloading and scheduling process. The task analysis and selection functions are responsible for analyzing and selecting tasks in the LCDA, respectively. The process in lines 1 to 6 is executed when the number of existing tasks changes by more than 20%, and the process in lines 7 to 10 is executed only when offloading occurs. The task analysis and selection functions are responsible for the task management for offloading in the LCDA. The server selection function is responsible for finding a target server for offloading. If an appropriate task cannot be identified by the task analysis function or task selection function, or if the server selection function cannot find a suitable edge cloud server, the LCDA determines that offloading a task to the edge cloud server is not possible—the LCDA will try to offload the task to the central cloud server or continue executing the task on the current user device. In the next section, we validate the LCDA's latency performance. Throughout this technique, we assume that LCDA does not take into account the scheduling of VMs in edge and central cloud servers. Because we focus on the rapid processing of latency-sensitive tasks, it is assumed that all VMs in the edge and cloud servers are running concurrently.

Algorithm 3. LCDA ()

```

1: for all  $task_i$ , where  $device_i \in Device, \forall_i \in \{1, 2, \dots, n\}$ ;
2:    $Task\_analysis\_using$  (1) and (2);
3:    $Task\_Selection$  ();
4: end for
5: Update the status of each task;
6: Store task status information;
7: for all  $server_i$  in  $edge_{pool}, \forall_i \in \{1, 2, \dots, n\}$ ;
8:    $Server\_Selection$  ();
9:   if there is no suitable  $server_i$  for offloading
10:    if  $task_i$  is latency-sensitive
11:      for all  $server_j$  in  $cloud_{pool}, \forall_j \in \{1, 2, \dots, n\}$ ;
12:        Find the  $server_j$  with lowest CPU utilization;
13:      end for
14:    else //  $task_i$  is latency-sensitive
15:      Continue to run on the current user device;
16:    end if
17:  end if
18: end for

```

4. Performance Evaluation

In this section, we present experimental results that demonstrate the performance of the LCDA in terms of reducing the processing time by managing task offloading while guaranteeing task deadline satisfaction. The experiments evaluate how different offloading techniques affect the service time of the MEC systems, and analyze the impact on its functions, namely the task failure rate and processing time for different numbers of devices, tasks, task ratios, and execution times. We simulate various scenarios using the EdgeCloudSim simulator [12]. This simulator extends the CloudSim toolkit [13] with additional features that allow networking resources and edge node modeling to enable accurate simulations of the physical edge infrastructure.

For the experiments, we assume that there are four VMs and one host running in the central cloud center, 14 hosts, and eight VMs running in the edge cloud center, and 500 devices (users) unless specified otherwise. The user can specify the type of the application such as task length, data transmission speed, and delay sensitivity. To verify the effectiveness of the LCDA, we consider four different kinds of applications; augmented reality, health, heavy computation, and infotainment. The augmented reality and health applications have a short task length, and the increase of CPU utilization of the edge cloud servers is relatively low. However, as it is important to obtain a quick response, we have set these two applications to be latency-sensitive. The heavy computation and infotainment have long task lengths, and the increase of CPU utilization of the edge cloud server is relatively high. Since the response time and accuracy are relatively insignificant for playing games, we set them as latency-tolerant applications. As for the default application setting, we use two latency-sensitive applications and two latency-tolerant applications. The parameters of this simulation are listed in Table 1.

Table 1. Application parameter setting in the simulation.

	Augmented Reality	Health	Heavy Computation	Infotainment
active_period (sec)	40	45	60	30
idle_period (sec)	20	90	120	45
avg_data_upload (KB)	1500	20	2500	25
avg_data_download (KB)	25	1250	200	1000
avg_task_length (MI)	9000	3000	45000	15000
utilization_on_edge [0–100]	6	2	30	10
utilization_on_cloud [0–100]	0.6	0.2	3	1
delay_sensitivity [0, 1]	0.9	0.7	0.1	0.3

In this experiment, we conduct a performance analysis by comparing five offloading algorithms, including our LCDA. First, a network-based offloading algorithm performs the task on the edge cloud server and offloads the tasks to the central cloud if the network capacity is exceeded. As mentioned earlier in the paper, it is assumed that the WAN environment serves as the connection between the edge cloud server and the central cloud server. We measure the delay that occurs when sending a 1-Mbit dummy task to the cloud in a WAN environment and then divide the delay value by one to set the bandwidth (Mbps) of the WAN. When the value of this bandwidth exceeds 6 (that is, when the delay time is less than 0.165 s), we offload the task to the cloud; otherwise, we operate the task on the edge cloud server. Second, a utilization-based offloading algorithm performs the task offloading from the edge server to the central cloud server when a certain amount of CPU utilization is exceeded. In this case, the CPU threshold of the edge cloud server is 80%. Third, a hybrid offloading algorithm combines these two cases. Finally, a Mobile+Edge algorithm runs a task on the user device and offloads the task to the edge cloud server when the amount of necessary mobile resources exceeds the amount of available resources. In this case, if the requested CPU capacity of a task exceeds that of the current user device, the task offloads to the edge server otherwise, the execution continues on the current user device. For all those offloading techniques, we evaluate the task failure rates and average processing times, which are important performance criteria in distributed computing.

In Figure 4, the x-axis represents the scalability of devices (users), and the y-axis denote the task failure rate and processing time, respectively. Overall, the task failure rate and processing time tend to increase slightly, with an increase in the number of devices. The task failure rate is the lowest for the Mobile+Edge algorithm, and the LCDA has the lowest task failures, except for the Mobile+Edge algorithm. Since the LCDA offloads a latency-tolerant task with large size and length preferentially, the failure rate of the LCDA is slightly higher than that of the Mobile+Edge algorithm. On the other hand, the processing time is the worst for the Mobile+Edge algorithm across all algorithms, and the LCDA shows relatively good results. This implies that the LCDA is a sufficient alternative to the existing MEC offloading techniques, which are likely to violate the deadline of a task because of the long processing time due to offloading. Even though the task failure rate of the LCDA is slightly higher than the others, the costs are negligible.

To verify the effectiveness of scalability in terms of the number of applications, we conducted another experiment showing the percentage of failed tasks and processing time with respect to the number of applications, as shown in Figure 5. Note that in this experiment, even though the number of applications increases, the ratio of latency-sensitive to latency-tolerant applications is always 5:5 (the number 8 in the x-axis means that the numbers of latency-sensitive and latency-tolerant applications are both 4, and the number 20 in the x-axis means that the numbers of latency-sensitive and latency-tolerant applications are each 10). In terms of the task failure rate, the Mobile+Edge algorithm shows the best performance, and when compared to the other algorithms, the LCDA shows 35.3% better performance on average. In terms of the processing time, the Mobile+Edge algorithm has the worst performance, and the network-based algorithm has the best performance. The LCDA

shows the second-best performance and consumes approximately 83.3% less time on average than the Mobile+Edge algorithm.

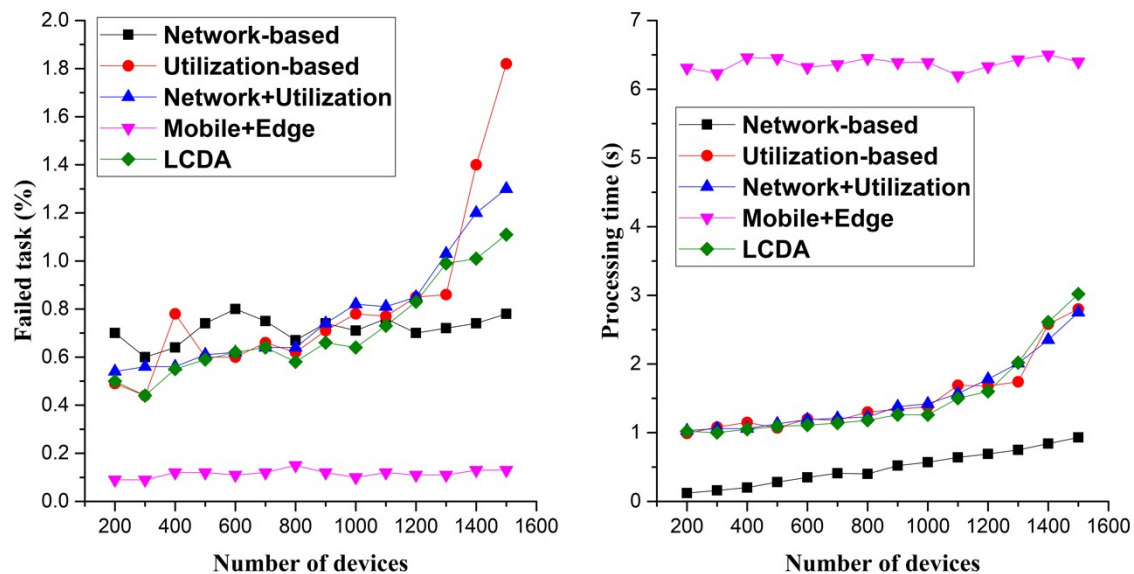


Figure 4. Scalability based on the number of users (devices). LCDA stands for latency classification deadline aware task offloading algorithm.

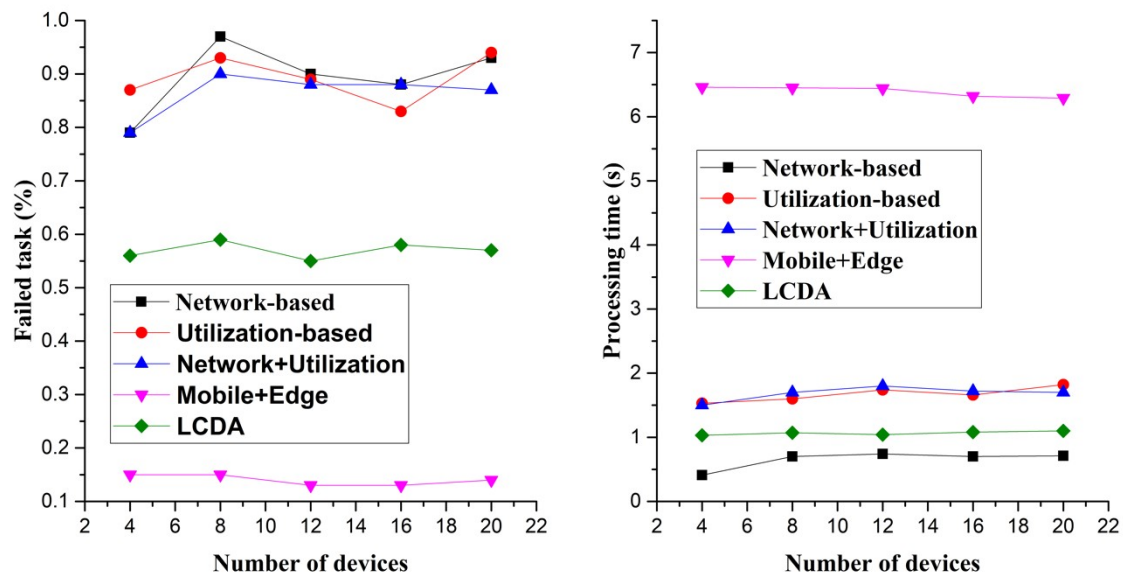


Figure 5. Performance comparison for the number of applications.

In Figure 6, we conducted an experiment to investigate the performance change of the MEC system by varying the ratio of application types. To validate the effect of the ratio of latency-sensitive to latency-tolerant applications, we performed experiments by varying the ratio of the application types from 10:0 to 0:10. The application tagged as latency-sensitive is the augmented reality application, and the application tagged as latency-tolerance is the infotainment application. For instance, if the ratio of the application is 3:7, the experiment is performed for the ratio of three augmented reality applications and seven infotainment applications. Since the latency-tolerant applications are relatively heavy, it can be seen that both task failure rate and processing time of the latency-tolerant tasks are higher than those of the latency-sensitive tasks. Similar to other experiments, the Mobile+Edge

technique shows the best performance in terms of task failure rate, and the LCDA has about 43.3% lesser failure rate than those of the other three algorithms. In terms of the processing time, the Mobile+Edge technique shows the worst performance, and the LCDA shows a performance improvement of 77.9% compared to the performance of the Mobile+Edge technique.

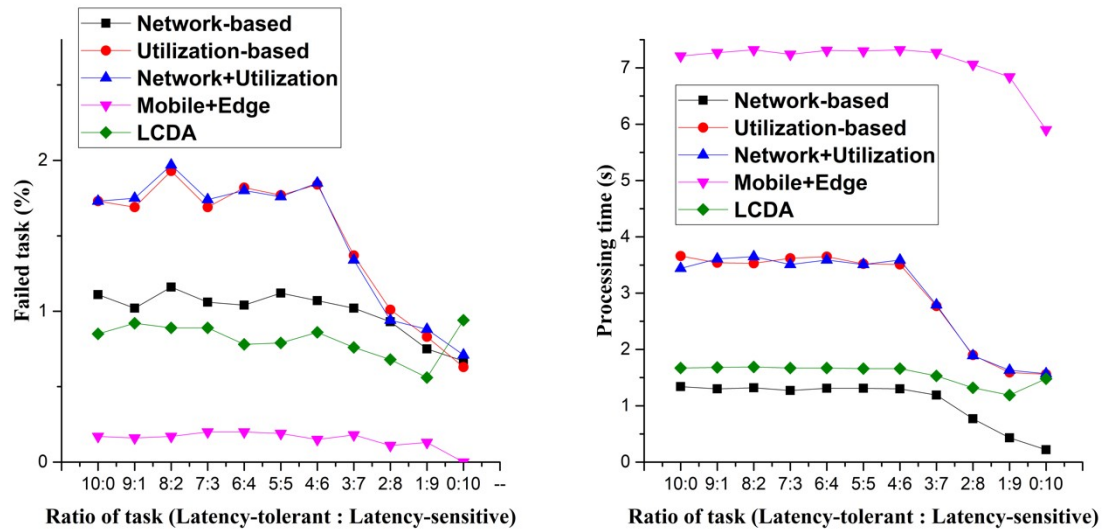


Figure 6. Performance comparison for latency-tolerant to latency-sensitive ratio.

To measure the efficiency of the LCDA algorithm from various perspectives, we evaluated its performance by increasing simulation execution time, as shown in Figure 7. The time was measured at intervals of 10, 30, 60, and 120 min. As expected, the performance of all the approaches seems similar to the previous experiments. The Mobile+Edge approach showed the best and worst performers in terms of task failure rate and processing time, respectively. In terms of the task failure rate, the LCDA reduces the task failure rate by 32.8% compared to the other three methods. In terms of processing time, the LCDA shows 83.1% better performance compared to the Mobile+Edge approach.

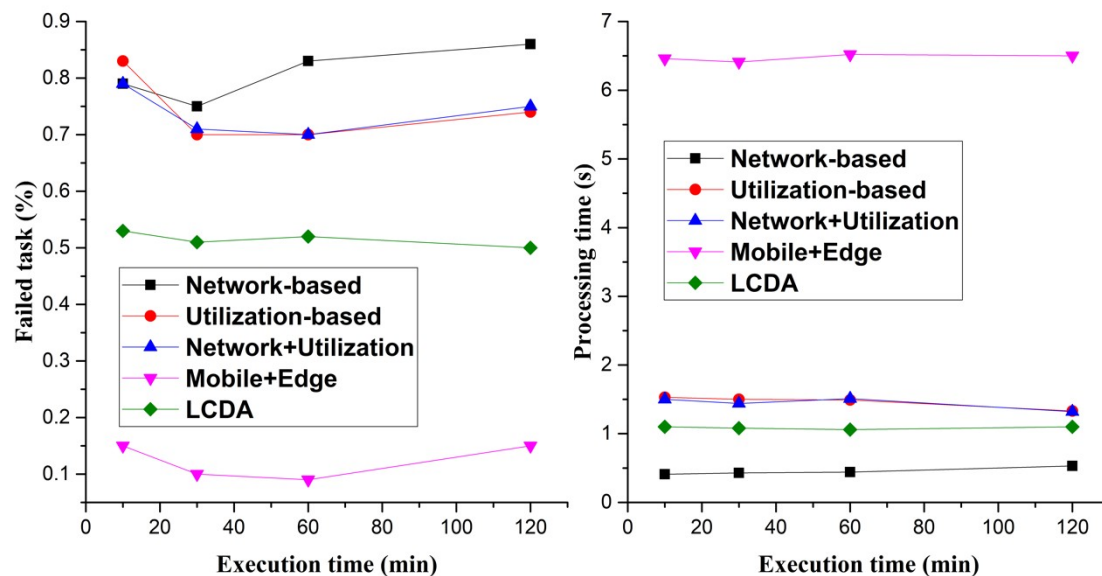


Figure 7. Performance comparison with respect to execution time.

To evaluate the progressing improvement for the latency-sensitive applications, we checked the results of the experiments from two perspectives for each application. Figure 8 shows the results of

comparing the performance of the failed task rate for four applications. The LCDA and Mobile+Edge algorithms are both employed by increasing the number of devices from 200 to 1600 for 10 min. From the task failure rate perspective, the overall performance is better with the Mobile+Edge algorithm than the LCDA. However, in terms of accurate processing, the performance of the augmented reality and health applications, which are latency-sensitive applications, outperform those of heavy computation and infotainment applications, which are latency-tolerant, and satisfy our experimental goal (Figure 8a). On the other hand, Figure 8b shows that the task failure rate for the augmented reality applications where fast and accurate execution is most important and is worse than the infotainment, which is latency-tolerant. In Figure 8a, the failed task rate range of the heavy computation application is about 1–5% and has the worst performance compared to the other three applications with about a 0–1.5% failed task rate range. This result shows that even though the LCDA is applied, the performance is low for the application with a long task length. In Figure 8b, there are fluctuations in the failed task rate of the latency-tolerant applications. This result shows that offloading without considering the task type provides unstable performance for latency-tolerant applications.

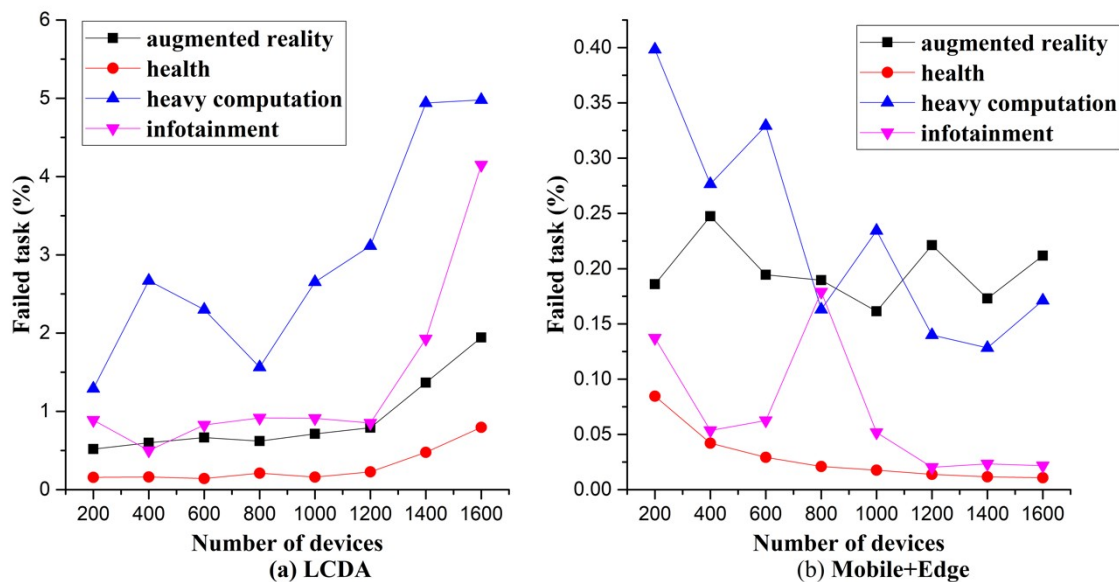


Figure 8. Performance comparison of the failed task rate for different application types.

The experimental procedures for the graphs in Figure 9 are the same as those for Figure 8, except that the performance comparison target is the processing time and not task failure rate. When comparing the LCDA and the Mobile+Edge application, the average performance seems to have improved by 70.3% for the LCDA. Even though heavy computation and infotainment applications use a lot of resources than augmented reality and health applications, we have guaranteed short processing time for the latency-sensitive applications in situations where multiple applications are running concurrently. In particular, from observing the processing time for augmented reality, which is a latency-sensitive application, the priority of the augmented reality application over the infotainment application can be guaranteed when the LCDA is applied.

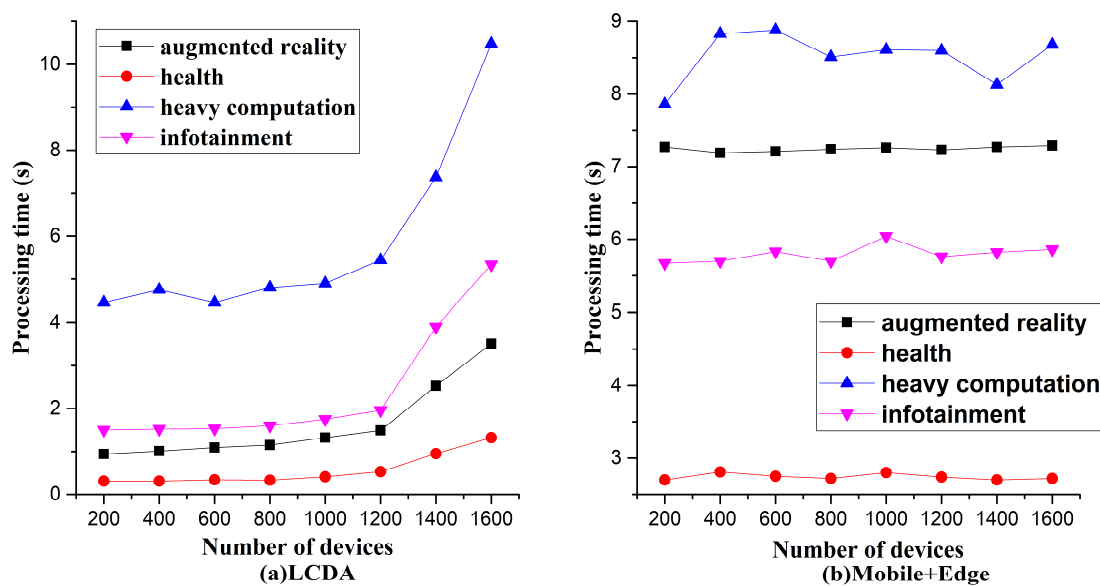


Figure 9. Performance comparison of processing time according to the different application types.

5. Discussion

We will summarize every related research using two perspectives: scheduling to minimize the delay of offloading and scheduling to guarantee the deadline satisfaction of services in an IoT environment.

In [14], the authors developed a task-scheduling policy based on network size and user node mobility to minimize delay in a fog network environment. They considered both a stationary network scenario with a fixed average delay and a non-stationary scenario with frequent changes in average delay based on the rapid movement of users. On the other hand, the authors of [15] proposed an optimization problem solution to minimize the delay when multiple users offload tasks to multiple edge cloud nodes performing complex IoT event processing, such as speech and face recognition. The author of [16] researched a voluntary-node-based task scheduling technique that autonomously determines which resources to allocate to user nodes according to their current resource allowance. This method can effectively increase idle resource utilization. In [17], the authors developed online algorithms for dynamic service caching at the edge computing environment. This technique quickly handled the load of unknown patterns and minimized the cost of operating the edge cloud server, including the cost of forwarding requests to the central cloud server and downloading the new service. The author of [18] proposed a dispatching method that determines which job is offloaded from the user device, and a scheduling method that determines which task is processed first on the edge and central cloud servers to minimize the average response time. In particular, latency sensitivity was assigned a larger weight to give a task higher priority. However, these studies focused only on reducing the offloading delay or the total response time of a system, and the problem of satisfying the deadlines of each task was not considered. In this paper, we considered both satisfying deadlines and minimizing the delay in IoT environments. None of the studies mentioned above considered the deadline satisfaction objectives in the context of task scheduling.

In [19], the authors proposed a multi-decision mobile computation offloading scheme that can minimize energy consumption while guaranteeing deadline satisfaction. The authors of [20] investigated a resource provisioning problem that can optimize the cost of a system while ensuring the deadline satisfaction of IoT service. The authors of [21] designed a deadline-aware rate allocation scheme that guarantees the deadline satisfaction of services by assigning higher priority to IoT services with deadline constraints when allocating network bandwidth at the data center. Various other studies have considered deadline satisfaction, but our goal was to solve multi-purpose problems, such as

energy and cost minimization or fair resource allocation, while satisfying deadlines. Based on our goal of minimizing both latencies during the offloading and task execution time, as well as satisfying task deadlines, we have developed a method that guarantees faster task processing performance than previous methods.

In addition, the authors of [22] developed a resource allocation strategy for performing user request approval, scheduling, and placement of service virtual machines to maximize user QoS (quality of service). In particular, they used the remaining deadline information to schedule edge server selection and used the LRU (Least Recently Used) policy, assuming that urgent services were not necessarily popular services. However, during the experiment, they ignored the size and transmission delay of the data. In the MEC environment, data size and transmission delay are more critical than propagation delays of links and queuing delays of user requests. In this paper, we divided the network into WLAN, MAN, and WAN by type, and we calculated the delays for each scenario. Moreover, considering failures caused by the CPU capacity of nodes, network delay, and user mobility, derived more accurate experimental results.

6. Conclusions

As IoT quickly becomes a reality, there is an increased need for the MEC paradigm. The MEC is a network architecture that supports computing, analysis, and storage capacity at the edge of the network. In particular, it can provide performance benefits for applications that require IoT environments. However, providing short-latency connections and ensuring the appropriate execution of latency-sensitive tasks without incurring deadline violations have remained as challenges to be addressed. In order to overcome these challenges, we propose a latency-classification-based deadline-aware task offloading algorithm based on the urgency of tasks while avoiding deadline violations. We establish a simulation environment to evaluate the performance of the LCDA approach and compare the proposed algorithm to the existing offloading algorithms. Based on the simulation output, we can prove that the LCDA achieves significant processing time reduction with a reasonable number of task failures for latency-sensitive applications. As future work, we intend to apply machine learning techniques to calculate efficiently the deadlines of tasks as well as the edge cloud server's resources. In addition, we intend to continue expanding the algorithm to reduce task failure rates of the LCDA.

Author Contributions: All the authors contributed equally to work. All authors read and approved the final manuscript.

Funding: This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. NRF-2019R1A2C1006754).

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

References

1. Plageras, A.P.; Psannis, K.E.; Stergiou, C.; Wang, H.; Gupta, B. Efficient IoT-based sensor BIG Data collection–processing and analysis in smart buildings. *Future Gener. Comput. Syst.* **2018**, *82*, 349–357. [\[CrossRef\]](#)
2. Gao, G.; Xiao, M.; Wu, J.; Huang, H.; Wang, S.; Chen, G. Auction-based VM Allocation for Deadline-Sensitive Tasks in Distributed Edge Cloud. *IEEE Trans. Serv. Comput.* **2019**. [\[CrossRef\]](#)
3. Giust, F.; Sciancalepore, V.; Sabella, D.; Filippou, M.C.; Mangiante, S.; Featherstone, W.; Munaretto, D. Multi-Access Edge Computing: The Driver Behind the Wheel of 5G-Connected Cars. *IEEE Commun. Stand. Mag.* **2018**, *2*, 66–73. [\[CrossRef\]](#)
4. Alam, M.G.R.; Hassan, M.M.; Uddin, M.Z.; Almogren, A.; Fortino, G. Autonomic computation offloading in mobile edge for IoT applications. *Future Gener. Comput. Syst.* **2019**, *90*, 149–157. [\[CrossRef\]](#)
5. Zhang, F.; Ge, J.; Li, Z.; Li, C.; Wong, C.; Kong, L.; Luo, B.; Chang, V. A load-aware resource allocation and task scheduling for the emerging cloudlet system. *Future Gener. Comput. Syst.* **2018**, *87*, 438–456. [\[CrossRef\]](#)

6. Lyu, X.; Tian, H.; Jiang, L.; Vinel, A.; Maharjan, S.; Gjessing, S.; Zhang, Y. Selective Offloading in Mobile Edge Computing for the Green Internet of Things. *IEEE Netw.* **2018**, *32*, 54–60. [\[CrossRef\]](#)
7. Dai, H.; Zeng, X.; Yu, Z.; Wang, T. A scheduling algorithm for autonomous driving tasks on mobile edge computing servers. *J. Syst. Arch.* **2019**, *94*, 14–23. [\[CrossRef\]](#)
8. Yang, L.; Cao, J.; Cheng, H.; Ji, Y. Multi-user Computation Partitioning for Latency Sensitive Mobile Cloud Applications. *IEEE Trans. Comput.* **2014**, *64*, 1. [\[CrossRef\]](#)
9. Sahni, J.; Vidyarthi, D.P. A Cost-Effective Deadline-Constrained Dynamic Scheduling Algorithm for Scientific Workflows in a Cloud Environment. *IEEE Trans. Cloud Comput.* **2015**, *6*, 2–18. [\[CrossRef\]](#)
10. Arabnejad, H.; Barbosa, J.G. Multi-QoS constrained and Profit-aware scheduling approach for concurrent workflows on heterogeneous systems. *Future Gener. Comput. Syst.* **2017**, *68*, 211–221. [\[CrossRef\]](#)
11. Choi, H.; Kang, J.; Lee, D.; Chung, K.; Yu, H. Resource Utilization-Aware Scheduling Technique Based on Dynamic Cache Refresh Scheme in Large-Scale Cloud Data centers. In Proceedings of the 2017 International Conference on Computational Science and Computational Intelligence (CSCI), Las Vegas, NV, USA, 14–16 December 2017; pp. 1560–1565. [\[CrossRef\]](#)
12. Sonmez, C.; Ozgovde, A.; Ersoy, C. Edgecloudsim: An environment for performance evaluation of edge computing systems. *Trans. Emerg. Telecommun. Technol.* **2018**, *29*, e3493. [\[CrossRef\]](#)
13. Calheiros, R.N.; Ranjan, R.; Beloglazov, A.; De Rose, C.A.; Buyya, R. CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software* **2011**, *41*, 23–50. [\[CrossRef\]](#)
14. Tan, Y.; Wang, K.; Yang, Y.; Zhou, M.T. Delay-Optimal Task Offloading for Dynamic Fog Networks. In Proceedings of the ICC 2019–2019 IEEE International Conference on Communications (ICC), Shanghai, China, 20–24 May 2019. [\[CrossRef\]](#)
15. Guan, G.; Dong, W.; Zhang, J.; Gao, Y.; Gu, T.; Bu, J. Queec: QoE-aware edge computing for complex IoT event processing under dynamic workloads. In Proceedings of the ACM Turing Celebration Conference-China, Chengdu, China, 17–19 May 2019. [\[CrossRef\]](#)
16. Zhang, G.; Shen, F.; Chen, N.; Zhu, P.; Dai, X.; Yang, Y. DOTS: Delay-Optimal Task Scheduling Among Voluntary Nodes in Fog Networks. *IEEE Internet Things J.* **2018**, *6*, 3533–3544. [\[CrossRef\]](#)
17. Zhao, T.; Hou, I.H.; Wang, S.; Chan, K. ReD/LeD: An asymptotically optimal and scalable online algorithm for service caching at the edge. *IEEE J. Sel. Areas Commun.* **2018**, *38*, 1857–1870. [\[CrossRef\]](#)
18. Tan, H.; Han, Z.; Li, X.Y.; Lau, F.C. Online job dispatching and scheduling in edge-clouds. In Proceedings of the IEEE INFOCOM 2017—IEEE Conference on Computer Communications, Atlanta, GA, USA, 1–4 May 2017. [\[CrossRef\]](#)
19. Hekmati, A.; Teymoori, P.; Todd, T.D.; Zhao, D.; Karakostas, G. Optimal Multi-Decision Mobile Computation Offloading with Hard Task Deadlines. *IEEE Trans. Mob. Comput.* **2019**, *99*, 1. [\[CrossRef\]](#)
20. Yao, J.; Ansari, N. Reliability-Aware Fog Resource Provisioning for Deadline-Driven IoT Services. In Proceedings of the 2018 IEEE Global Communications Conference (GLOBECOM), Abu Dhabi, United Arab Emirates, 9–13 December 2018. [\[CrossRef\]](#)
21. Shen, B.; Chilamkurti, N.; Wang, R.; Zhou, X.; Wang, S.; Ji, W. Deadline-aware rate allocation for IoT services in data center network. *J. Parallel Distrib. Comput.* **2018**, *118*, 296–306. [\[CrossRef\]](#)
22. Ascigil, O.; Phan, T.K.; Tasiopoulos, A.G.; Surlas, V.; Psaras, I.; Pavlou, G. On uncoordinated service placement in edge-clouds. In Proceedings of the 2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom), Hong Kong, China, 11–14 December 2017. [\[CrossRef\]](#)

