applied sciences

MDPI

Article

Task Migration with Partitioning for Load Balancing in Collaborative Edge Computing

Sungwon Moon and Yujin Lim *

Department of IT Engineering, Sookmyung Women's University, Seoul 04310, Korea; sungwon268@sookmyung.ac.kr

* Correspondence: yujin91@sookmyung.ac.kr; Tel.: +82-02-2077-7305

Abstract: Multi-access edge computing (MEC) has emerged as a promising technology to facilitate efficient vehicular applications, such as autonomous driving, path planning and navigation. By offloading tasks from vehicles to MEC servers (MECSs), the MEC system can facilitate computation-intensive applications with hard latency constraints in vehicles with limited computing resources. However, owing to the mobility of vehicles, the vehicles are not evenly distributed across the MEC system. Therefore, some MECSs are heavily congested, whereas others are lightly loaded. If a task is offloaded to a congested MECS, it can be blocked or have high latency. Moreover, service interruption would occur because of the high mobility and limited coverage of the MECS. In this paper, we assume that the task can be divided into a set of subtasks and computed by multiple MECSs in parallel. Therefore, we propose a method of task migration with partitioning. To balance loads, the MEC system migrates the set of subtasks of tasks in an overloaded MECS to one or more underloaded MECSs according to the load difference. Simulations have indicated that, compared with conventional methods, the proposed method can increase the satisfaction of quality-of-service requirements, such as low latency, service reliability, and MEC system throughput by optimizing load balancing and task partitioning.

Keywords: multi-access edge computing; task migration; task partitioning; load balancing

Citation: Moon, S.; Lim, Y. Task Migration with Partitioning for Load Balancing in Collaborative Edge Computing. *Appl. Sci.* **2022**, *12*, 1168. https://doi.org/10.3390/app12031168

Academic Editor: Andrea Prati

Received: 30 November 2021
Accepted: 21 January 2022
Published: 23 January 2022

1. Introduction

In the 5G era, the concept of vehicular networks has extended to the Internet of Vehicles (IoV), in which intelligent and interactive vehicular applications such as autonomous vehicles, path planning and navigation are enabled by communication and computation technologies [1–3]. These computation-intensive applications frequently require ultra-reliability and low-latency (uRLLC). For example, autonomous driving applications need to process huge amounts of data in real time (as high as 2 GB/s) within about 10 ms, which is a very low delay constraint [2]. Applications for efficient driving, e.g., path planning or navigation generate less data than autonomous driving, but tasks from the applications should be performed within a still-low delay constraint of about 100 ms [4]. Despite improving the computing power of vehicles, processing such applications on vehicular terminals is still difficult while ensuring high computing power and quality-of-service (QoS) requirements, such as low latency.

To overcome these limitations and facilitate efficient application processing, multi-access edge computing (MEC) has emerged as a promising technology that can satisfy the demand for the heavy computation of vehicles by providing rapid and sufficient computational resources to vehicles [5]. Thus, by offloading tasks from vehicles to MEC servers (MECSs), the MEC system can facilitate computation-intensive applications with low latency in vehicles with limited computing resources. Meanwhile, compared with conventional centralized cloud computing, offloading computation-intensive tasks to MECSs, which are geographically closer to vehicles, by deploying them at the network edge, can further reduce the network delay.

However, because MECSs have limited computing power, the computing power that the vehicle can request from the MECS depends on the MECS load level, i.e., the number of tasks already allocated to the MECS [6]. Due to the mobility of vehicles, vehicles are not evenly distributed across a MEC system; thus, the number of tasks offloaded to a MECS may not be evenly distributed among MECSs. Therefore, some MECSs are heavily congested, whereas others are lightly loaded. If a task is offloaded to a congested MECS, the task may be blocked or the delay will be prolonged. Thus, the delay requirements may be violated. Moreover, the mobility of vehicles causes uncertainty in downloading results [7]. Owing to the limited coverage of MECSs, the vehicle can travel out of the coverage of a MECS that offloads a task during a service session, resulting in a service interruption. Service interruption occurs because of dynamically changing radio association, which can significantly increase the total service latency, particularly in urban areas with highly dense infrastructure.

To improve the satisfaction with QoS requirements, references [8–13] studied service migration. In [8–10], a service migration that migrates services according to a vehicle's trajectory was proposed to minimize the average completion time of tasks. As they only considered the trajectory of vehicles without considering the load of MECSs, tasks can simultaneously migrate to a specific MECS. Thus, this MECS becomes overloaded, which can result in uneven loads among MECSs and reduce system throughput. Service migration with load balancing was addressed in [11–13]. In [11], a service migration method was proposed that considered the load difference and migration cost in balancing the loads of MECSs in a MEC system. In [12], service migration was determined using the traffic conditions and loads of each MECS to minimize the migration costs and travel times of vehicles. The load of each MECS in [11,12] was measured using the number of tasks in the MECS. In [13], the method distributed the load from an overloaded cell to a lightly loaded cell from its neighbors to increase the utilization of radio resources. The load is represented by the resource block utilization ratio of the cells. The delay constraint of a task was not considered in [11–13] when defining the load. In this paper, we define the load as a computational workload required to complete a task, considering delay constraint. In addition, unlike in [11–13], in which tasks are migrated to one MECS, tasks in overloaded MECS may be migrated to one or more underloaded MECSs to distribute the loads evenly in the MEC system. The authors of [14,15] proposed a load balancing/resource allocation method in a framework, called dew computing, that integrates smart mobile devices into distributed and high-performance computing platforms. In [16], their method determined whether to offload a task or not in order to reduce the processing time, energy consumption of the smartphone and monetary cost. However, we determine whether/where to migrate to and partition among MECSs once the vehicles offload their tasks them in order to reduce the task execution delay and improve the system throughput. Therefore, we address a method in the collaborative edge computing framework that enables tasks to be processed outside of the communication coverage of the MECS offloaded from the vehicle.

Thus, we consider task partitioning in which a task can be divided into a set of subtasks and computed by multiple MECSs in parallel. This can improve service reliability by reducing the task execution delay. In other words, by reducing the execution delay, a vehicle can obtain computing results without service interruption before it moves out of the coverage of a MECS. Resource utilization or energy consumption costs are additionally incurred in the process of partitioning or gathering the computing results, but there are more benefits, such as reducing execution time and improving system throughput.

In contrast to the methods used in [8–13] that migrate an entire task to the MECS without partitioning, the methods used in [17–20] addressed task migration with partitioning. In [17], the partitioned tasks were migrated to the serving MECS and cloud server to reduce the average total delay of tasks. In [18], to reduce the service delay, the partitioned tasks were migrated from the MECS in which the task was offloaded to the geographically nearest MECS. The task considered in [17,18] consisted of independent or sequentially dependent subtasks. In practice, a task is generally composed of multiple threads. For

example, navigation applications frequently involve threads of graphics, camera preview, and video processing [21]. Each thread can be considered a subtask, so that computation tasks of the applications can be partitioned and the task can be modeled using a directed acyclic graph (DAG). In [19,20], methods were investigated to optimize performance in terms of delay and energy consumption. However, if the set of subtasks of a task migrates to the MECS without considering the loads of the MECSs, the total service delay may increase. This becomes useless if the subtasks migrate to a congested MECS or a specific MECS simultaneously. By selecting an appropriate MECS to which to migrate the subtasks and determining the amount of workload required of subtasks, the satisfaction of the task's QoS requirements can be improved. Task migration and partitioning methods based on machine learning were proposed in [6,7,11,14,22]. Generally, methods using machine learning require a huge amount of data for learning and analysis. However, since we assume a small clustered area with its own characteristics, the proposed MEC system is built on each area to optimize the performance, reflecting the characteristics of each area.

Therefore, we propose a method of task migration with partitioning using a heuristic algorithm to distribute loads among MECSs and execute the task in parallel in a collaborative edge computing framework. The set of subtasks in an overloaded MECS is migrated to one or more underloaded MECSs depending on the load difference between the overloaded and underloaded MECSs. Balancing the loads in the MEC system through task partitioning and migration can increase both the satisfaction of QoS requirements, such as low latency and service reliability, and MEC system throughput.

The remainder of this paper is organized as follows. In Section 2, we describe the system model and the problem formulation. In Section 3, we introduce the proposed method in detail. In Section 4, we present and discuss the simulation results. Finally, Section 5 concludes the study.

2. System Model and Problem Formulation

The MEC system proposed in this paper is shown in Figure 1. We consider a MEC system comprising a set of MECSs, M, located in roadside units (RSUs) and a set of vehicles, N. A MECS can provide seamless communication and computing service coverage for vehicles on the road. We assume that the MECSs communicate with each other through wired links, so communication latency and bandwidth can be ignored [20,23]. Each MECS uses two queues: a waiting queue and computation queue. The waiting queue uses a limited task buffer to accommodate tasks offloaded from vehicles, and the computation queue operates in a first-in-first-out manner to process tasks. A MECS controller acts as a centralized controller with global knowledge of the MECSs in a system and is responsible for making the decisions for migration and partitioning. Time is divided into slots t, and each time slot has an equal duration of Δs. We assume that the vehicle travels at speed s during the simulation.

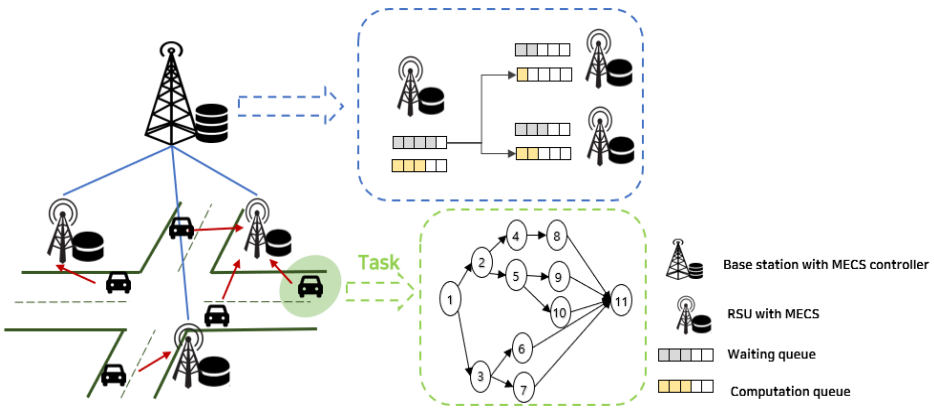


Figure 1. The architecture of the proposed MEC system.

During each time slot, vehicle n offloads its task in which the data size is λ_n (in bits), the computation workload is η_n (in CPU cycles), and the delay constraint is T_n^{max}, which can be divided into several interdependent subtasks. The dependency among subtasks is modeled by a DAG, i.e., $\mathcal{G}(\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of dependent subtasks in task n and $\mathcal{E}$ is the set of dependencies between the subtasks in task n. Each subtask i in task n is associated with a computation workload $\eta_{n,i}$. In this paper, we only consider the tree-structured task graph, where the out-degree of each vertex is equal to 1, except for that of the first vertex. The first and last subtasks compute at the serving MECS, to complete the first subtask and migrate other subtasks simultaneously [18], and to gather and transmit computing results at the end of the service session [13]. For example, in an autonomous driving application, recognizing a situation can be the first subtask and driving guidance can be the last subtask.

We assume that at the beginning of each time slot t, a vehicle offloads its tasks to the serving MECS, giving it the highest signal, as MECS m. The MECS receives the task offloaded from the vehicle, then the task is placed at the waiting queue of the MECS to be partitioned and migrated. At the end of time slot t, the MECS controller gathers the status information from the MECSs in the system. Based on the information, the MECS controller makes a decision of task migration and partitioning whether/where to migrate or partition tasks in the waiting queues of servers. According to the decision made by a MECS controller, at time slot $\tau + 1$, the tasks stored in the waiting queues are placed into the computation queues of the servers to which they belong or the computation queues of other servers. Once tasks or subtasks are placed in the computation queue, they cannot be partitioned or migrated to another MECS additionally. Assuming that some subtasks in MECS m are migrated to another MECS m', the subtasks are performed in MECS m and m' respectively, the computation results of subtasks are merged by MECS m, and sent to the vehicle. These operation of a MEC network are shown in Figure 2.

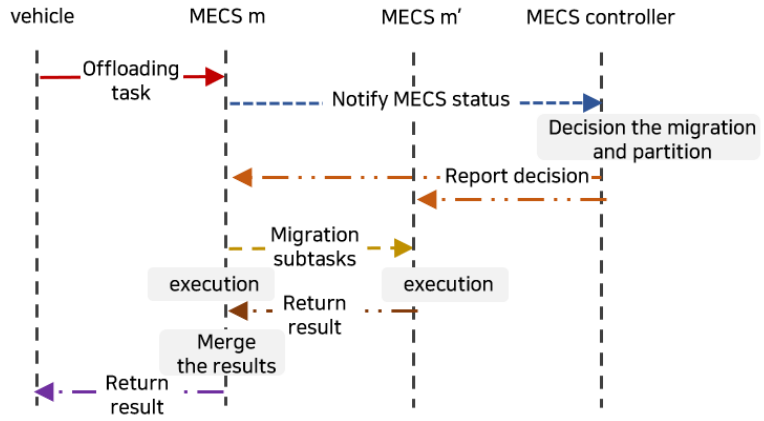


Figure 2. The operation of a MEC network.

The data rate for forwarding the task offloaded from a vehicle to a MECS at time slot t can be calculated as

$$R_{n,m}(t) = B_n \log_2\left(1 + \frac{P \cdot H_{n,m}}{\sigma^2}\right), \tag{1}$$

where B_n is the bandwidth allocated to a channel, P is the transmission power of the vehicle, $H_{n,m}$ is the channel gain between vehicle n and MECS m, and σ^2 is the received noise power. The transmission delay to offload the task from the vehicle to its serving MECS is

$$T_n^{trans}(t) = \frac{\lambda_n}{R_{n,m}}, \tag{2}$$

The load imposed on the MECS by the task is the computational workload required to complete the task within the delay constraint. As the vehicle has already offloaded its task

to the serving MECS and the task is placed in its waiting queue, the task must be completed within a time, excluding the transmission delay under the delay constraint. Therefore, the CPU cycles required to complete the tasks in the waiting queue are

$$l_n = \frac{\eta_n}{T_n^{max} - T_n^{trans}}, \tag{3}$$

The load of each MECS is equal to the sum of the workloads of tasks in the waiting queue $Q_m^w(t)$ of MECS m at the end of time slot t.

$$\zeta_m(t) = \sum_{n \in Q_m^w(t)} l_n \,, \tag{4}$$

To measure whether the load of each MECS is evenly distributed among the MECSs in the MEC system, the MECS controller calculates the average load difference among the MECSs in the MEC system.

$$\partial = \frac{1}{M} \sum_{m=1}^{M} \sum_{m' \neq m}^{M} |\zeta_m - \zeta_{m'}| \,, \tag{5}$$

As the number of vehicles associated with each MECS is not evenly distributed, the number of tasks offloaded to the MECSs is not even. Therefore, because the workloads imposed on MECSs differ, the loads of some MECSs are high and those of others are low. If tasks are migrated to an overloaded MECS or a particular MECS simultaneously, the delay requirements of tasks may be violated because of the increased time spent in the waiting queue of the MECS. Thus, to increase the satisfaction of QoS requirements and system throughput, we propose a task migration with partitioning method to balance the loads among MECSs in the MEC system, as follows:

$$min. \, \partial \,, \tag{6}$$

3. Task Partitioning and Migration Methods

In this section, we propose task migration with partitioning methods. To balance the loads among MECSs in a MEC system, the set of subtasks of tasks in an overloaded MECS migrate to underloaded MECSs. In addition, we determine the amount of workload of subtasks that must be migrated to a target MECS to reduce execution delay and improve system throughput. To increase the satisfaction of QoS requirements of tasks, we try to reduce the computation time of a task by increasing the concurrency of the clusters' computations. This is because the constraint of computation time for each cluster is required to guarantee the satisfaction of the delay constraint of a task but it is difficult to obtain the information for each cluster.

3.1. Task Partitioning Method

The following describes a task partitioning that divides tasks into a set of subtasks, i.e., clusters. The task is divided into a set of subtasks, and offloading them increases the parallelism of task execution, minimizing latency [24]. Therefore, we will apply a scenario in which the set of subtasks can be migrated to other MECSs, which can balance the loads among MECSs in the MEC system. In addition, the subtasks are executed collaboratively by MECSs in the system, which can improve the QoS requirements of tasks by minimizing the delay for computing and improving system throughput.

We assume that the task can be divided into a set of subtasks. We define the set of clusters as C, which consists of the subclusters C_i, e.g., $C = \{C_2, C_3\}$ (Figure 3). Cluster C_i is a set that consists of subtasks i and all its successors in the task graph, except for the last subtask. For example, we have $C_2 = \{2, 4, 5, 8, 9, 10\}$ and $C_3 = \{3, 6, 7\}$. We also define cluster C_i, consisting of subcluster C_j, e.g., C_5 is the subcluster of C_2 ($C_5 \subset C_2$). At this time,

subtask j is the direct successor of subtask i. As the clusters in C are not related to each other, each cluster can be migrated independently.

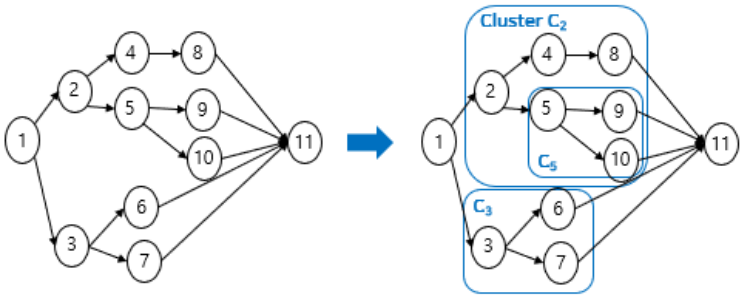


Figure 3. Example of DAG task graph and clusters.

C may have more than one cluster; therefore, the order of clusters' migration should be determined. We select the order of clusters with large computational workloads, which have a large number of CPU cycles to compute the subtasks [20]. We represent the CPU cycles required to complete the set of subtasks in a task as l_i^c, which is calculated as $\eta_{n,i}/(T_n^{max} - T_n^{trans})$. To balance the load among MECSs, particularly between overloaded and underloaded MECSs, the amount of workload of clusters is migrated using the load difference of MECSs. By migrating them in descending order cluster size, the load of the overloaded MECS and underloaded MECS can be narrowed. If the condition that the cluster workload is lower than the load difference between the overloaded and underloaded MECS is not satisfied, the large cluster will be split into smaller ones.

3.2. Task Migration Method

The proposed task migration method is presented in Algorithm 1. The objective of our method is to balance the loads among MECSs in the MEC system, particularly between the overloaded and underloaded MECSs. The MECS controller must determine which MECS m to migrate from and select the target MECS m' to migrate to. If MECS m migrates its task to target MECS m', the MECS controller must determine the task and the amount of workload to be migrated to MECS m'. At the end of time slot t, the MECS controller monitors the load status of the MECSs. Subsequently, the MECS controller calculates the average load of the MECSs in the MEC system as follows:

$$\zeta(t) = \frac{1}{M} \sum_{m=1}^{M} \zeta_m(t), \tag{7}$$

The MECS controller classifies the overloaded or underloaded MECSs based on their average load. The MECSs whose loads are higher than $\zeta(t)$ are overloaded compared with other MECSs as a set of α. Similarly, the MECSs whose loads are equal to or lower than $\zeta(t)$ are underloaded compared with other MECSs as a set β. To balance the loads among MECSs in the MEC system, particularly between the overloaded and underloaded MECSs, tasks in overloaded MECSs migrate to underloaded MECSs. Therefore, the MECS controller determines that each MECS m in set α migrates its tasks to MECS m' in set β. The MECS controller calculates the load difference between the MECS in set α and MECS in the set β. This is expressed as follows:

$$\varphi_{m,m'}(t) = \zeta_m(t) - \zeta_{m'}(t), \;\; m \in \alpha, \;\; m' \in \beta, \tag{8}$$

Now, the MECS controller determines a target MECS m' with the highest $\varphi_{m,m'}(t)$ for MECS m. The MECS m then migrates the amount of workload to the target MECS m' as according to the load difference between the MECS m and MECS m'. Here, the amount of workload of subtasks that MECS m migrates to MECS m' should be less than $\varphi_{m,m'}(t)$ Tasks to be migrated in the waiting queue of MECS m are selected in descending order.

After partitioning and migrating the subtasks, the MEC controller removes m and m' from α and β if their loads become similar to the average load of all MECSs or if the load difference between m and m' is similar. The MECS controller repeats this procedure until α or β is empty, or all of the tasks in m are partitioned and migrated.

Algorithm 1. Proposed method

at time slot t, initialize $\zeta(t)$ initialize set $\alpha = \{m|\zeta_m(t) > \zeta(t)\}$
initialize set $\beta = \{m'|\zeta_{m'}(t) \leq \zeta(t)\}$
if $|\alpha| \neq \varnothing$ and $|\beta| \neq \varnothing$ **then**
  MECS $m = argmax_{m\in\alpha}\zeta_m(t)$
  select task i in the waiting queue of MECS m
  a set of subtasks in task i, $C = \{C_i\}$, sort by the amount of workload in descending order
  for the number of subtasks in C, $|C|$ **do**
  MECS $m' = argmax_{m\in\alpha,\ m'\in\beta}\varphi_{m,m'}(t)$
  if $l_i^c \leq \varphi_{m,m'}(t)$ **then**
    migrate C_i to MECS m'
    update $\zeta_{m(t)} = \zeta_m(t) - l_i^c$, $\zeta_{m'} = \zeta_{m'}(t) + l_i^c$
  else
    break
  update $\varphi_{m,m'}(t) = \zeta_m(t) - \zeta_{m'}(t)$
  if $\varphi_{m,m'}(t) \leq threshold$ **then**
    continue
else
$\alpha = \alpha\backslash\{\mathrm{m}\}$
$\beta = \beta\backslash\{m'\}$
else
  tasks in the waiting queue are placed into its computation queue

4. Simulation Results and Discussion

In this section, we verify the performance of the proposed method compared with other methods. It is assumed that the MEC system is deployed in a clustered area grouped by the units of administrative districts in the urban environments where MECSs are densely deployed and vehicles are not driving at high speed. We randomly distributed 10 MECSs within a MEC system. The capacity of each MECS was 10 GHz. Vehicles were distributed in a Poisson distribution on the road. All vehicles had mobility within a random walk model in an area of 1000 m $\times$ 1000 m using the SUMO simulator [25]. The input size of the task followed a random distribution with 2–10 Kbits. The task graph used in the simulation was randomly generated, where the workload of each subtask followed a random distribution with 0.1–0.2 Gcycles. The mobility speed of the vehicle also followed a random distribution of 1–30 m/s. The parameters used in the simulation are summarized in Table 1 according to [26–28]. The path loss between the RSU and vehicle was modeled as $128.1 + 37.6\log_{10}(distance(\mathrm{km}))$. When vehicles offloaded their tasks simultaneously, the vehicles performed data transmission in different spectral bands via orthogonal frequency-division multiple access technology, and there was mutual interference among different subcarrier allocations.

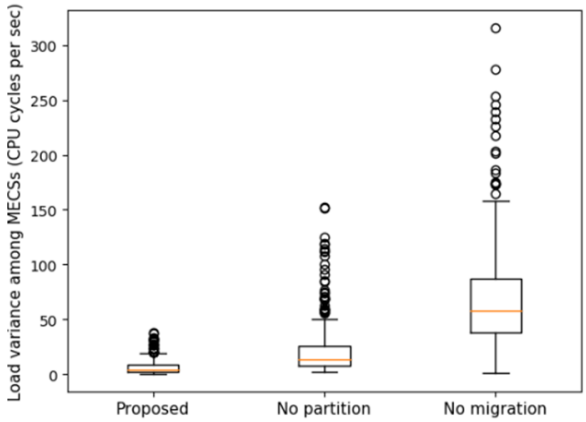
To demonstrate the reason for using task partition and migration in our method, we compared the *Proposed* method with *No partition* and *No migration* strategies. The *Proposed* method uses both task partitioning and migration. The *No partition* method migrates an entire task without any task partitioning to the target MECS, and the *No migration* method executes a task on the MECS in which the vehicle's task is offloaded without any task migration.

The load variance among the MECSs is shown in Figure 4. It shows the load variance of each time slot when the vehicle arrival rates were 0.3, 0.6 and 0.9. In other words, it is an indicator of the distribution of the load among MECSs during the simulation. In the *Proposed* method, the set of task subtasks in an overloaded MECS was migrated to

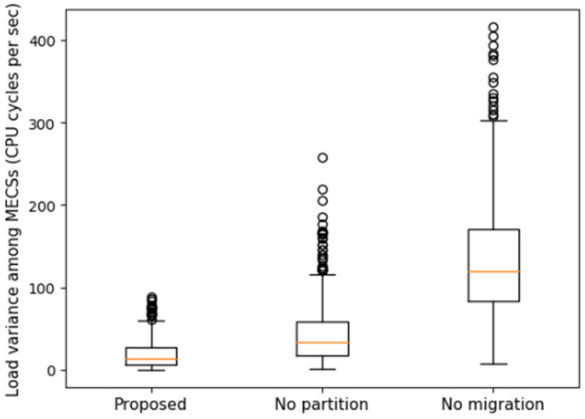
underloaded MECSs. Under *No partition* method, the all tasks in the overloaded MECS were migrated to the underloaded MECS. Thus, the *Proposed* method further reduced the load difference between overloaded and underloaded MECSs compared with the *No partition* method. *No migration* method had a large load variance, because vehicles were not evenly distributed geographically and had high mobility; thus, the loads may have been concentrated in specific MECSs.

Table 1. Parameters used in the simulation.

Parameter	Value
subcarrier bandwidth	12.5 kHz
transmission power of a vehicle (P_n)	1 W
received noise power (σ^2)	10^{-11} mw
duration of a time slot (Δ)	0.1 s
delay constraint of a task (t_n^{max})	0.1–0.6 s
radius of a MECS	250 m
waiting queue size of a MECS	5 tasks
computation queue size of a MECS	5 tasks

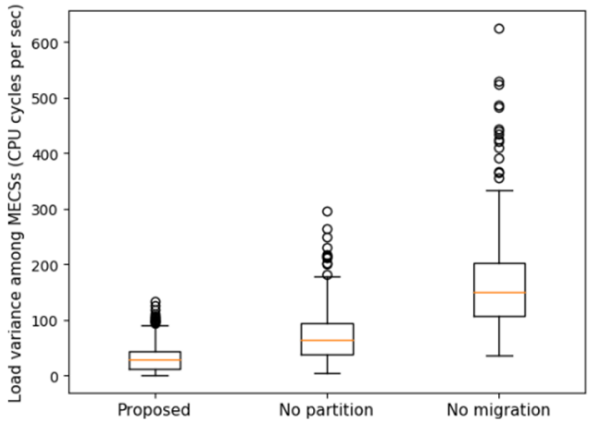


(**a**)



(**b**)

Figure 4. *Cont.*



(**c**)

Figure 4. Load variance among MECSs of the three methods; (**a**) vehicle arrival rate = 0.3; (**b**) vehicle arrival rate = 0.6; (**c**) vehicle arrival rate = 0.9.

The task completion rates within the delay constraints and task reliability rates are shown in Figures 5 and 6, respectively. The rate of task completion within the delay constraint is defined as the ratio of the number of completed tasks while satisfying the delay constraint to the total number of offloaded tasks. The task reliability rate is defined as the ratio of the number of completed tasks to the total number of offloaded tasks while receiving the result without the service interruption. The task completion rates within the delay constraints and task reliability rates are affected by the task execution delay. For example, for a congested MECS, the load of the MECS becomes overloaded. This can increase the waiting time of the MECS queue, resulting in an increase in a task execution delay. As shown in Figure 5, depending on the load balancing among MECSs, the task completion rates within the delay constraints had a high rate. Thus, if the task execution delay is prolonged, service interruption occurs when receiving its result. To receive the result without service interruption, the vehicle must obtain the result quickly before it moves out of the coverage of the MECS to which the task is offloaded. If the load balancing among MECSs is appropriate, the task execution delay will be reduced, and the reliability rate will be high. Therefore, the *Proposed* method had the highest rate, and the *No migration* method had the lowest rate in terms of the task completion rates within the delay constraints and task reliability rates. Therefore, it was better to perform migration than not, and migration with partitioning performed better than full migration. Therefore, in this study, we considered task migration with partitioning.

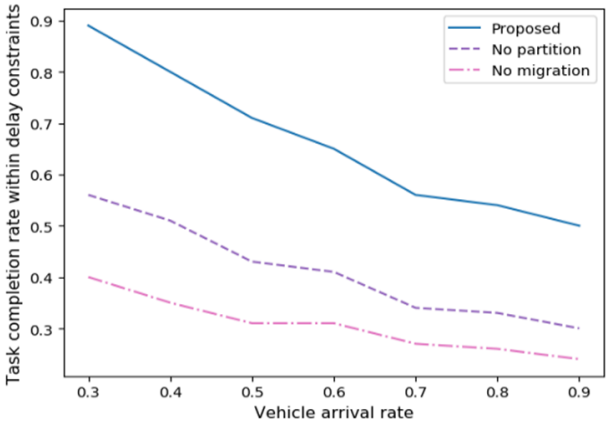


Figure 5. Task completion rates of the three methods.

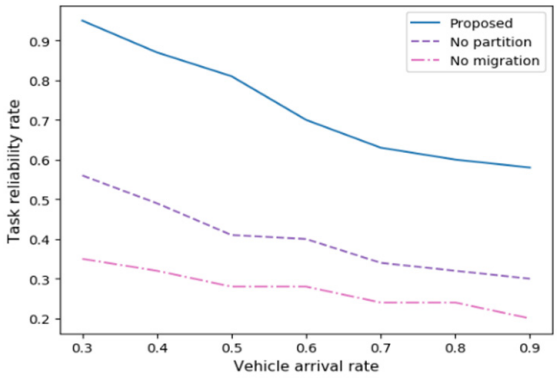


Figure 6. Task reliability rates of the three methods.

To verify the performance of our proposed method, we compared it with four other methods: *IPSO* [20], *Nearest*, *Least* and *Random*. *IPSO* aims to minimize the service failure probability by considering task execution delay. It is a method of migrating tasks to MECSs that meet the latency constraint of the subtasks. *IPSO* defines the workload of the subtasks as the ratio of the sum of the required CPU cycles to the sum of its input and output data. *Nearest* is a method of migrating tasks in overloaded MECSs to the nearest MECS among underloaded MECSs. *Least* is a method of migrating tasks in overloaded MECSs to the least loaded MECS among underloaded MECSs. *Random* is a method of migrating tasks in overloaded MECSs to randomly selected MECSs among underloaded MECSs.

The task completion rates within the delay constraints under different values of delay constraints are shown in Figure 7 for a vehicle arrival rate of 0.6. *Proposed* achieved the highest completion rate for different values of the delay constraint, a value of approximately 35–60% higher than that of other methods. The task reliability rate under different values of mobility speed of vehicles is shown in Figure 8 for a vehicle arrival rate of 0.6. Service interruption occurs when the vehicle can travel out of the coverage of the MECS that is offloaded during a service session, which significantly increases the total service latency. We observed that *Proposed* achieved the highest task reliability rate for different values of mobility speed, approximately 20–51% higher than that of other methods. A comparison of the environment with high and low speeds indicated differences of approximately 8% for *Proposed*, 11% for *IPSO*, 15% for *Nearest*, 17% for *Least*, and 13% for *Random*. Under conditions with different delay constraints and mobility speed, *Proposed* performed better than that of others by optimizing the load balancing, which reduces the task execution delay. Therefore, we conducted the experiment by setting the delay constraints to 0.1–0.6 s and the mobility speed to 1–30 m/s.

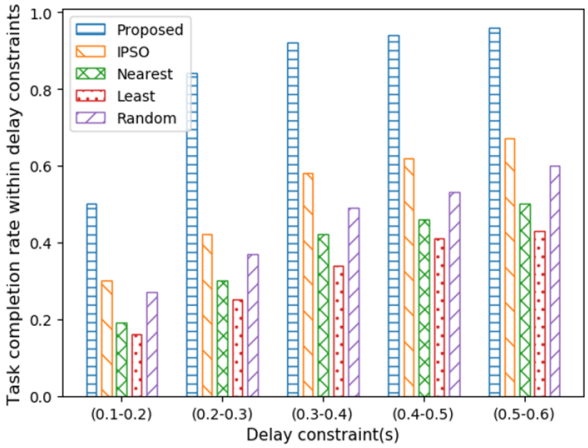


Figure 7. Task completion rates for delay constraints.

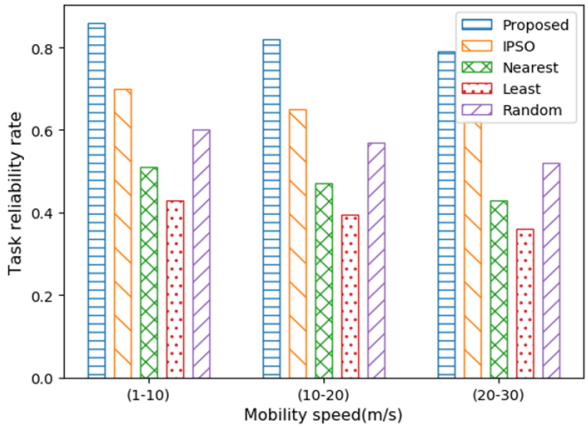


Figure 8. Task reliability rates for mobility speed.

Figure 9 shows the load variance among the MECSs when the arrival rates were 0.3, 0.6 and 0.9. In *Proposed*, the set of subtasks in the task was migrated to one or more target MECSs depending on the load difference. In *Nearest* and *Random*, the set of subtasks in the task was migrated to the target MECSs based on distance and randomness. In *Least*, the set of subtasks in the task is migrated to the least loaded MECS, which indicated a scenario in which the least loaded MECS was concentrated. Therefore, depending on which MECS was selected as the target MECS, the loads of the MECSs varied. In *IPSO*, the set of subtasks was migrated based on delay constraint. *Proposed* shows better performance than *IPSO* in terms of load variance because proposed considers load difference to determine the task migration. *IPSO* shows better performance than *Nearest*, *Random* and *Least*, because it is affected by the delay, especially the queueing delay associated with the load of the MECS.

Figure 10 depicts the average blocking rate and standard deviation of the blocking rate with varying arrival rates. The blocking rate is defined as the ratio of the number of tasks blocked by the MECS to the total number of offloaded tasks. The task is blocked by the MECS when the task is offloaded to a MECS whose queue is full. In *Nearest* and *Random*, the target MECS may be another overloaded MECS or may become overloaded owing to concentrating tasks on it. In *Least*, because tasks in overloaded MECSs are migrated to the least loaded MECS, the least-loaded MECS becomes overloaded. Therefore, because some MECSs have a high load, the task may be blocked when it is offloaded to the MECS with a high load. In *IPSO*, when all MECSs in MEC system do not meet the delay constraints for the subtasks, the subtasks migrate to the MECS that minimizes the computing delay. In this case, since only the processing delay is considered, there is likely that the overloaded MECS is selected. *Proposed* not only had the lowest blocking rate but also the smallest standard deviation compared with the other methods because the load was relatively well distributed. The system throughput with varying arrival rates is shown in Figure 11. The system throughput is defined as the multiplication of the number of tasks completed per time slot by the average task size. When the loads are balanced among the MECSs, the block rate decreases. In addition, if the block rate decreases, more tasks can be performed; thus, the system throughput increases.

Figures 12 and 13 show the rate of task completion within the delay constraints and task reliability rate, respectively. The task completion rate within the delay constraints is affected by the task execution delay consisting of transmission, queueing, execution, and migration delay. Similarly, task reliability is affected by the task execution delay. A migration delay is the time a task is migrated to another MECS, which does not account for most of the total delay because it uses wired links. The queueing delay is affected by the load of the MECS, that is, the length of the queue of the MECS. Increasing the parallelism of task execution reduces the execution delay. Therefore, the better the loads of MECSs are distributed, the less the queuing delay affected by the MECS load and queue length,

increasing the task completion rate within the delay constraints. The task completion rate of *Proposed* was approximately 18–58% higher than that of other methods. If the task execution delay is prolonged, the vehicle may move out of the coverage of the MECS, which is offloaded to the task during a service session. Thus, service interruption may occur more often in receiving its result. To receive the result without the service interruption, the loads of the MECS must be appropriately distributed to reduce the task execution delay. The reliability rate of *Proposed* was approximately 12–46% higher than that of the other methods because it distributed the loads of MECSs well.

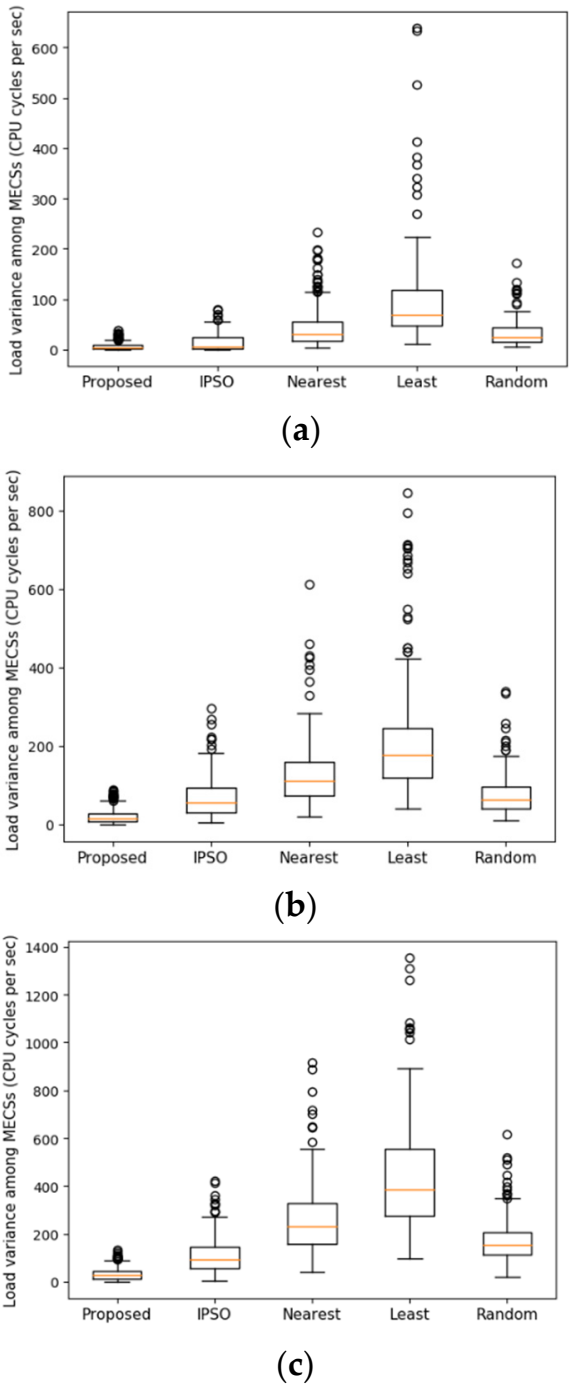


Figure 9. Load variance among MECSs of the five methods; (**a**) vehicle arrival rate = 0.3.; (**b**) vehicle arrival rate = 0.6; (**c**) vehicle arrival rate = 0.9.

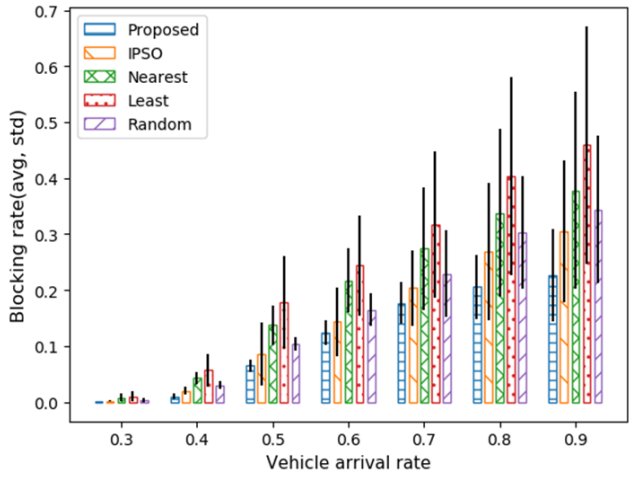


Figure 10. Blocking rates for vehicle arrival rate.

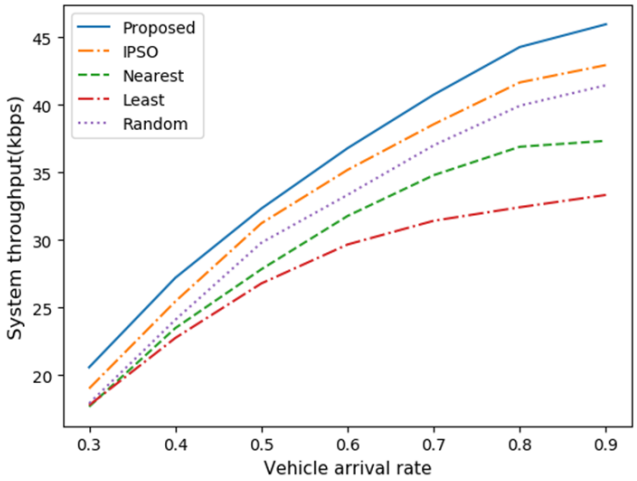


Figure 11. System throughput for vehicle arrival rate.

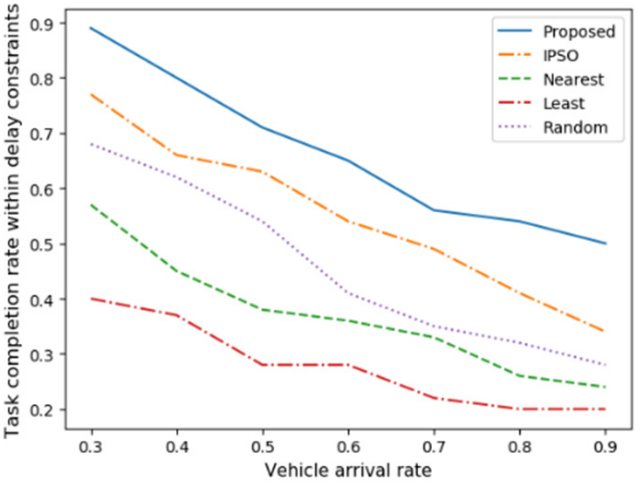


Figure 12. Task completion rates for vehicle arrival rate.

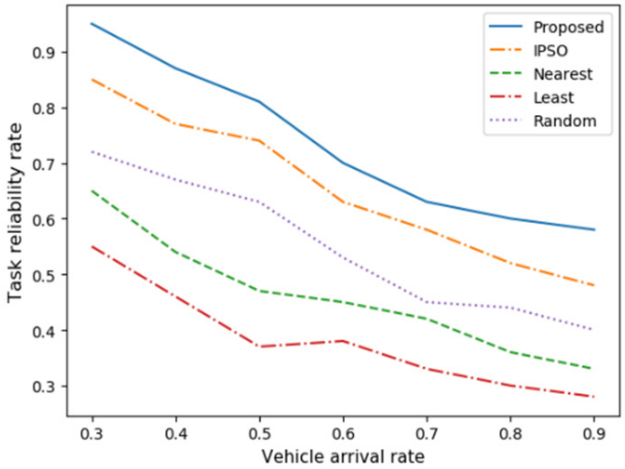


Figure 13. Task reliability rates for vehicle arrival rate.

5. Conclusions

In this paper, we investigated task migration with partitioning method in vehicular networks to balance loads among MECSs in a MEC system. We considered a scenario in which the vehicle can offload its task to a serving MECS and the task could be divided into a set of subtasks. To balance the load among MECSs, overloaded MECSs migrated the set of subtasks to one or more underloaded MECSs, depending on the load difference of the overloaded and underloaded MECSs. We compared the proposed method with conventional methods through simulations, and the results indicated that the proposed method distributed the load appropriately among MECSs. In addition, the proposed method increased the completion rate within the delay constraints and system throughput. In addition, the blocking rate and service interruption caused by vehicle mobility were reduced. In future research, we will consider task scheduling when selecting a task to be partitioned or when executing the task to reduce the task execution delay. Furthermore, we will expand the proposed method in a distributed manner to reduce the complexity of the MECS controller in a more dynamic environment. We will investigate the multi-agent learning or federated learning methods to determine the optimal migration and partitioning strategy in the extended MEC system.

Author Contributions: Conceptualization, S.M. and Y.L.; Methodology, S.M.; Software, S.M.; Writing—Review Editing, S.M. and Y.L.; Supervision, Y.L. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No.2021R1F1A1047113).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

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

References

1. Raza, S.; Wang, S.; Ahmed, M.; Anwar, M.R. A Survey on Vehicular Edge Computing: Architecture, Applications, Technical Issues, and Future Directions. *Hindawi Wirel. Commun. Mob. Comput.* **2019**, *2019*, 3159762. [CrossRef]
2. Liu, S.; Liu, L.; Tang, J.; Yu, B.; Wang, Y.; Shi, W. Edge Computing for Autonomous Driving: Opportunities and Challenges. *Proc. IEEE* **2019**, *107*, 1697–1716. [CrossRef]
3. Al-Fuqaha, A.; Guizani, M.; Mohammadi, M.; Aledhari, M.; Ayyash, M. Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. *IEEE Commun. Surv. Tutor.* **2015**, *17*, 2347–2376. [CrossRef]
4. 3GPP TR 22.886 Version 15.3.0 Release 15. Study on Enhancement of 3GPP Support for 5G V2X Services. Available online: https://www.3gpp.org/DynaReport/22886.htm (accessed on 28 November 2021).

5. Dai, Y.; Xu, D.; Maharjan, S.; Zhang, Y. Joint Load Balancing and Offloading in Vehicular Edge Computing and Networks. *IEEE Internet Things J.* **2019**, *6*, 4377–4387. [CrossRef]
6. Tang, M.; Wong, V. Deep Reinforcement Learning for Task Offloading in Mobile Edge Computing Systems. *IEEE Tran. Mob. Comput.* **2020**, *1*, 19810254. [CrossRef]
7. Li, M.; Gao, J.; Zhao, L.; Shen, X. Deep Reinforcement Learning for Collaborative Edge Computing in Vehicular Networks. *IEEE Trans. Cogn. Commun. Netw.* **2020**, *6*, 1122–1135. [CrossRef]
8. Li, J.; Shen, X.; Chen, L.; Pham, D.; Ou, J.; Wosinska, L.; Chen, J. Service Migration in Fog Computing Enabled Cellular Networks to Support Real-Time Vehicular Communications. *IEEE Access* **2019**, *7*, 13704–13714. [CrossRef]
9. Taleb, T.; Ksentini, A.; Frangoudis, P.A. Follow-Me Cloud: When Cloud Services Follow Mobile Users. *IEEE Trans. Cloud Comput.* **2019**, *7*, 369–382. [CrossRef]
10. Liu, C.; Tang, F.; Hu, Y.; Li, K.; Tang, Z.; Li, K. Distributed Task Migration Optimization in MEC by Extending Multi-Agent Deep Reinforcement Learning Approach. *IEEE Trans. Parallel Distrib. Syst.* **2021**, *32*, 1603–1614. [CrossRef]
11. Moon, S.; Park, J.; Lim, Y. Task Migration Based on Reinforcement Learning in Vehicular Edge Computing. *Hindawi Wirel. Commun. Mob. Comput.* **2019**, *10*, 1–24. [CrossRef]
12. Yuan, Q.; Li, J.; Zhou, H.; Lin, T.; Luo, G.; Shen, X. A Joint Service Migration and Mobility Optimization Approach for Vehicular Edge Computing. *IEEE Trans. Veh. Technol.* **2020**, *69*, 9041–9052. [CrossRef]
13. Addali, K.; Kadoch, M. Enhanced Mobility Load Balancing Algorithm for 5G Small Cell Networks. In Proceedings of the 2019 IEEE Canadian Conference of Electrical and Computer Engineering (CCECE), Edmonton, AB, Canada, 5–8 May 2019; pp. 1–5.
14. Hirsch, M.; Mateos, C.; Zunino, A.; Majchrzak, T.A.; Grønli, T.-M.; Kaindl, H. A Task Execution Scheme for Dew Computing with State-of-the-Art Smartphones. *Electronics* **2021**, *10*, 2006. [CrossRef]
15. Hirsch, M.; Mateos, C.; Zunino, A. Augmenting Computing Capabilities at the Edge by Jointly Exploiting Mobile Devices: A Survey. *Future Gener. Comput. Syst.* **2018**, *88*, 644–662. [CrossRef]
16. Muslim, N.; Islam, S.; Grégoire, J. Offloading Framework for Computation Service in the Edge Cloud and Core Cloud: A Case Study for Face Recognition. *Int. J. Netw. Manag.* **2020**, *31*, 2146. [CrossRef]
17. Feng, M.; Krunz, M.; Zhang, W. Joint Task Partitioning and User Association for Latency Minimization in Mobile Edge Computing Networks. *IEEE Trans. Veh. Technol.* **2021**, *70*, 8108–8121. [CrossRef]
18. Liu, J.; Zhang, Q. Adaptive Task Partitioning at Local Device or Remote Edge Server for Offloading in MEC. In Proceedings of the 2020 IEEE Wireless Communications and Networking Conference (WCNC), Seoul, Korea, 25–28 May 2020; pp. 1–6.
19. Kong, Y.; Zhang, Y.; Wang, Y.; Chen, H.; Hei, X. Energy Saving Strategy for Task Migration Based on Genetic Algorithm. In Proceedings of the 2018 International Conference on Networking and Network Applications (NaNA), Xi'an, China, 12–15 October 2018; pp. 330–336.
20. Liu, J.; Zhang, Q. Code-Partitioning Offloading Schemes in Mobile Edge Computing for Augmented Reality. *IEEE Access* **2019**, *7*, 11222–11236. [CrossRef]
21. Mahmoodi, S.E.; Uma, R.N.; Subbalakshmi, K.P. Optimal Joint Scheduling and Cloud Offloading for Mobile Applications. *IEEE Trans. Cloud Comput.* **2019**, *7*, 301–313. [CrossRef]
22. Cheng, Z.; Min, M.; Liwang, M.; Huang, L.; Gao, Z. Multiagent DDPG-Based Joint Task Partitioning and Power Control in Fog Computing Networks. *IEEE Internet Things J.* **2022**, *9*, 104–116. [CrossRef]
23. Anwar, M.R.; Wang, S.; Akram, M.F.; Raza, S.; Mahmood, S. 5G-Enabled MEC: A Distributed Traffic Steering for Seamless Service Migration of Internet of Vehicles. *IEEE Internet Things J.* **2022**, *9*, 648–661. [CrossRef]
24. Zhang, Y.; Liu, H.; Jiao, L.; Fu, X. To offload or not to offload: An Efficient Code Partition Algorithm for Mobile Cloud Computing. In Proceedings of the 2012 IEEE 1st International Conference on Cloud Networking (CLOUDNET), Paris, France, 28–30 November 2012; pp. 80–86.
25. Lopez, P.A.; Behrisch, M.; Walz, L.B.; Erdmann, J.; Flötteröd, Y.P.; Hilbrich, R.; Lücken, L.; Rummel, J.; Wagner, P.; Wiessner, E. Microscopic Traffic Simulation using SUMO. In Proceedings of the 2018 21st International Conference on Intelligent Transportation Systems (ITSC), Maui, HI, USA, 4–7 November 2018; pp. 2575–2582.
26. Zhang, J.; Guo, H.; Liu, J.; Zhang, Y. Task Offloading in Vehicular Edge Computing Networks: A Load-Balancing Solution. *IEEE Trans. Veh. Technol.* **2020**, *69*, 2092–2104. [CrossRef]
27. Cheng, K.; Teng, Y.; Sun, W.; Liu, A.; Wang, X. Energy-Efficient Joint Offloading and Wireless Resource Allocation Strategy in Multi-MEC Server Systems. In Proceedings of the 2018 IEEE International Conference on Communications (ICC), Kansas City, MO, USA, 20–24 May 2018; pp. 1–6.
28. Yang, C.; Liu, Y.; Chen, X.; Zhong, W.; Xie, S. Efficient Mobility-Aware Task Offloading for Vehicular Edge Computing Networks. *IEEE Access* **2019**, *7*, 26652–26664. [CrossRef]