electronics

Article

Linked-Object Dynamic Offloading (LODO) for the Cooperation of Data and Tasks on Edge Computing Environment

Svetlana Kim [1], Jieun Kang [2] and YongIk Yoon [2,*]

[1] Research & Business Development Foundation, Sookmyung Women's University, Seoul 04310, Korea; xatyna@sookmyung.ac.kr
[2] Department of IT Engineering, Sookmyung Women's University, Seoul 04310, Korea; kje9209@sookmyung.ac.kr
* Correspondence: yiyoon@sookmyung.ac.kr; Tel.: +82-10-5091-0388

Abstract: With the evolution of the Internet of Things (IoT), edge computing technology is using to process data rapidly increasing from various IoT devices efficiently. Edge computing offloading reduces data processing time and bandwidth usage by processing data in real-time on the device where the data is generating or on a nearby server. Previous studies have proposed offloading between IoT devices through local-edge collaboration from resource-constrained edge servers. However, they did not consider nearby edge servers in the same layer with computing resources. Consequently, quality of service (QoS) degrade due to restricted resources of edge computing and higher execution latency due to congestion. To handle offloaded tasks in a rapidly changing dynamic environment, finding an optimal target server is still challenging. Therefore, a new cooperative offloading method to control edge computing resources is needed to allocate limited resources between distributed edges efficiently. This paper suggests the LODO (linked-object dynamic offloading) algorithm that provides an ideal balance between edges by considering the ready state or running state. LODO algorithm carries out tasks in the list in the order of high correlation between data and tasks through linked objects. Furthermore, dynamic offloading considers the running status of all cooperative terminals and decides to schedule task distribution. That can decrease the average delayed time and average power consumption of terminals. In addition, the resource shortage problem can settle by reducing task processing using its distributions.

Keywords: edge computing; offloading computation; distributed collaboration; data processing; dynamic offloading; IoT; gateways

check for updates

Citation: Kim, S.; Kang, J.; Yoon, Y. Linked-Object Dynamic Offloading (LODO) for the Cooperation of Data and Tasks on Edge Computing Environment. *Electronics* **2021**, *10*, 2156. https://doi.org/10.3390/electronics10172156

Academic Editors: Ka Lok Man and Kevin Lee

Received: 30 June 2021
Accepted: 30 August 2021
Published: 3 September 2021

1. Introduction

Nowadays, with the rapid evolution of technology and a vast number of Internet of things (IoT) devices, including individual units, have improved processing ability (robust computing environment) with various embedded sensors. Due to this progress, the IoT devices can perform multiple functions (such as receiving/refining data from sensors in real-time, transferring, processing, and storing) independently and without computing resources in the server (external cloud) [1,2]. Edge computing has been proposed to process existing integrated service platforms by moving computing tasks from cloud servers to IoT devices. Edge computing is the workload of devices and offloads computational tasks to nearby computing devices, significantly reducing processing latency [3]. Existing high-latency and low-reliability cloud computing solutions are challenging to support time-critical cloud/IoT services requirements in transmission data and task processing delay. Edge computing is a centralized architecture where all nearby service requests are directed to a 'central' edge server. However, since the computing power of edge servers is not powerful as cloud-based servers, some issues such as resource limitations and computing latency between multiple competing tasks still occur [4,5].

Various approaches such as Mobile Cloud Computing (MCC) and Multiple Access Edge Computing/Mobile Edge Computing (MEC) support complementary cloud computing solutions using services adjacent to the edge network to cope with this disturbing problem [6–8]. MCC offloading, which uses unlimited resources in a cloud for some tasks, is the method to reduce loads of mobile devices. However, MCC has considerable disadvantages, such as low expandability, long propagation distance between the cloud server and devices, excessive consumption of limited bandwidth, personal protection and security problems [9]. Current MCC offloading methods cannot guarantee real-time data transfer and task delay, making them unsuitable for latency-sensitive applications. On the other hand, MEC is the offload concept of tasks collaboratively by leveraging both the MEC server and the end device (such as a mobile phone). Due to the limited battery life of the mobile device and the growing number of latency-sensitive applications, offloading tasks come with additional overhead in terms of latency and power consumption [10,11]. This problem was partially solved by dividing each task into local task and offload task. Local tasks are processing on the end device, and offload tasks are performing on the MEC server. However, computing resources can increase in some areas, which can exacerbate network problems and even cause issues that affect task execution times.

In recent years, much research has proposed addressing the resource limitations and computing latency issues by scheduling strategies of collaboration task offloading in an edge computing environment. The computing resource on edge is mainly composed of intelligent devices (note: intelligent devices (e.g., smart sensors and smartphones, can access a network, resulting in a considerable amount of network data) are called edge devices/end devices) and edge servers. For example, in [12–14], there is a job scheduling algorithm that utilizes the resources of cloud servers to handle highly overload situations. In [15], leverage resources in edge servers by offloading all computing-intensive tasks of the edge device to the edge server. In this case, computing resources and storage of devices are not utilized properly and wasted. In addition, multiple computation tasks or many devices may access the edge server at the same time. As a result, the workload will increase, long queues, and processing delays of tasks accordingly. In limited computational resources, a multi-MEC system has been proposing by joint communication offloading methods at the ends and edges [16–19]. However, due to the dynamic environment of the computing system, it is not easy to achieve task offload performance. In addition, the resource-rich IoT devices do not collaborate and are fully underutilized, resulting in a waste of their computing resources.

Since task offloading plays a critical role in edge-based service, edge must take full advantage of IoT's communication and computational resources. To this end, the edge server needs a balance task scheduler that, according to the characteristics of the computing task and device status, decides which tasks to offload to which IoT devices. Our previous paper [20] suggested distributed collaboration for computing offloading architectures. The architecture consists of a balanced collaboration system by assigning the master node, the second node, and the action node individually to edge nodes which contribute to all connected and communicable areas. It balanced computing resources through edge-to-edge collaboration and minimized latency due to relatively unbalanced overloads. Based on the [20] architecture, this paper proposes the LODO (Linked-object dynamic offloading) algorithm. LODO focuses on reasonable use of the computation resources of the IoT devices to processing computing tasks efficiently. The IoT devices are called edge nodes in the LODO algorithm. The LODO algorithm receives the compute tasks offloaded by the edge nodes and provides the hybrid state with offloading the tasks according to the resources and state of all edge nodes. The offloading location of computing tasks is uncertain and has various types; so are the compute resources status and performance gaps between nodes. Considers both characteristics, the computing resources of edge nodes can be fully balancing utilized. The main feature here is that the computing task can be reasonably offloaded to different edge nodes by collaborative processing.

The main contributions of this article are summarized as follows:

- We propose a Dynamic offloading method (DOM) with hybrid states that contains the resource requirements of offloading tasks and real-time resource availability information of each edge node. According to the current computational task execution state, the hybrid state is formed based on information related to all edge nodes (e.g., compute, storage, and network state). The edge computing reasonably offloaded the task to suitable edge nodes according to the hybrid state.
- We propose a linked-object algorithm that, according to offloading task status, provides two cooperation offloading options. If the computing power of an edge node cannot meet the task requirements, performing the task-linked option. When the edge node's memory/storage computing resource is not more available, executing the data-linked option. Each of these options helps solve the problem of load imbalance among computing nodes.
- We also investigate the application of forest fire that requires real-time sensing data and various time-critical tasks. For example, real-time data (such as temperature, humidity, wind, slope, and others) is necessary to predict the possibility of forest fires moving to other areas and the diffusion speed. If the task processing time is a large percentage of the total time, thus resulting in a processing delay of other tasks. Moreover, the total service time may become unacceptable when performing large tasks at the close but slow (low computing power) edge nodes. In this case, the LODO algorithm can offload computation tasks to suitable edge node according to the real-time computing resources status of the edge nodes.

The rest of this paper is organized as follows. Section 2 explains DOM (Dynamic Offloading Method) by hybrid states is formulating. Section 3, introduces details of the LODO properties approach for collaboration offloading. Scenario and results are presented in Section 4. Section 5 presents the discussion. Finally, Section 6 concludes this.

2. Dynamic Offloading Method (DOM) with Hybrid States

Based on our previous paper [20], this section identifies the reason for loads according to data-linked and task-linked, which are the major processes of an edge node. On edge node have edge gateways, end devices, and schedulers. Edge gateways can provide computing (*CPU*), storage (memory), bandwidth, and other system resources for edge computing operations. It is essential to reasonably use these computing resources depending on the offloading configuration to solve computation offloading efficiently. For example, suppose the goal is to reduce the load for data-linked processing. In this case, memory utilization is more important than *CPU* ratio, so utilizing the edge with more storage resources is necessary. On the other hand, task-linked processing increases *CPU* usage; it should use an edge with free computing resources. Using edge resources according to processing status can increase edge resource utilization and reduce the average execution time of computational tasks.

The resource range that can process in the edge node is definition 80% of *CPU* and 90% of memory. We define this as the reference overload threshold. If one exceeds the existing threshold range using a monitoring process, it determines the overload of data-linked and task-linked. The ratio is classified as a task-linked overload with high *CPU* usage and categorized as a data-linked overload with high memory usage (ref. Figure 1).

Depending on the hybrid state of the edge node, the dynamic offloading method (DOM) defines an overload range that considers the correlation/relationship between data and tasks. The following section describes whole activity states (hybrid states) through discussion and formulation about overload issues according to data and task of an edge node using DOM.

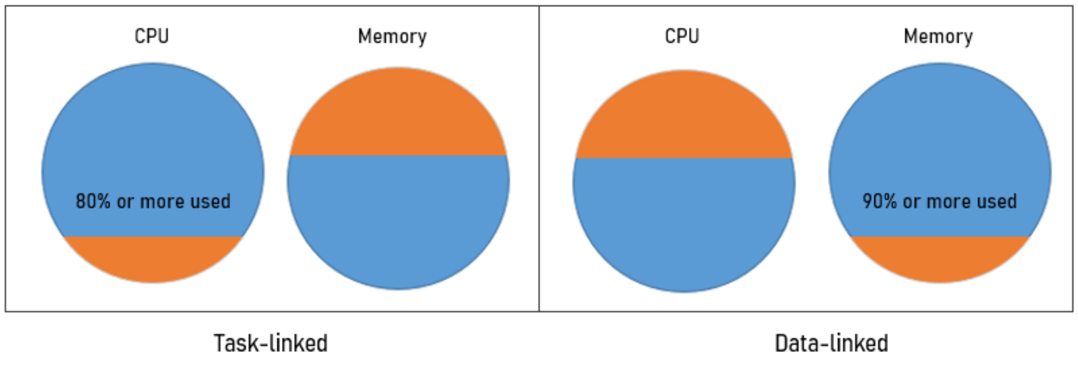


Figure 1. The resource range for data-linked and task-linked.

2.1. Expression of Hybrid States on Edge Node

As shown in Figure 2, edge computing is a process of collaboration offloading by monitoring the state of all edge nodes that change dynamically. Edge node consists of end devices that receive, store, and preprocesses the sensing data, and edge gateways analyze and outputs the results following the purpose of the domain. Since edge nodes have different resources and characteristics, real-time monitoring is a requirement in an ever-changing environment. In addition, there are different data and performance characteristics during the analysis depending on the purpose of the domain, so the offload characteristics may also differ. Representative offloading characteristics can classify into "offloading due to data," "offloading due to task," or "offloading due to Data & Task," and the offloading range need determine according to each characteristic. According to the current offloading execution state, the edge node must also provide suitable edge nodes.

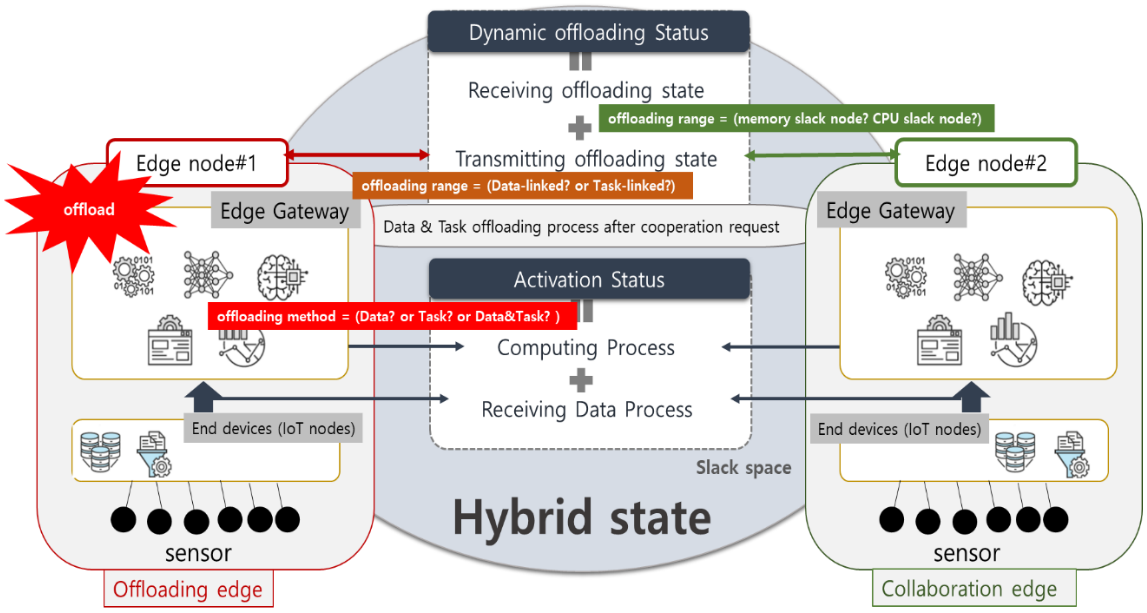

Dynamic offloading Status
Receiving offloading state
offloading range = (memory slack node? CPU slack node?)
Transmitting offloading state
offloading range = (Data-linked? or Task-linked?)
Edge node#1
offload
Edge Gateway
Data & Task offloading process after cooperation request
Edge node#2
Edge Gateway
Activation Status
offloading method = (Data? or Task? or Data&Task?)
Computing Process
Receiving Data Process
End devices (IoT nodes)
End devices (IoT nodes)
Slack space
Hybrid state
sensor
sensor
Offloading edge
Collaboration edge

Figure 2. Monitoring process with hybrid states on edge node.

The hybrid state to reasonably use the computing resource, and computing task is divide into three statuses, the "Activation status", "Dynamic offloading status," and "Slack space." The activation status means accurate assessment and incorporating the edge node state consumption due to the offloaded tasks. Receiving data and computing processes are the functions that derive an offloading task method to the due to data or/and task according to the problem computing resource. Offloading data method is when the edge node uses a memory value is more than 90%, and the task offload method is when the *CPU* value is more than 80%.

The dynamic offloading between edges nodes is one of the essential factors in determining task range based on offloading task method. According to the task-method attributes, the transmitting offloading state extract the range of data-linked or task-linked offloading. Transmitting offloading state extracts the range of data-linked or task-linked offloading according to the offloading method. The Receiving offloading state selects the offloading range that can tolerate in the slack space of the cooperative edge node, excluding 10% memory and 20% *CPU*. Here the slack space means a value obtained by subtracting activation status and dynamic offloading status from the hybrid state. This value can describe the state of the edge node like a hybrid state, should have more than 10% memory or 20% *CPU* basically to prevent overload.

The following Equation (1) computes the hybrid state of the edge node's dynamic active and slack space, depending on the data and task computation function. Equation (1) indicates the total performance state of an edge node. Based on data and task, the hybrid state is the sum of Equation (1a) the current performance (activation) state, Equation (1b) the offloading state, and the slack space.

$$\begin{gathered} Hybrid\ state\{CPU,\ Memory\} \\ = Activation\ Status\{C_i,\ M_i\} + Dynamic\ offloading\ Status\{C_i, M_i\} \\ +Slack\ Space\{C_i, M_i\}, \end{gathered} \tag{1}$$

Equation (1a) shows the current activation state, where C_i is *CPU* and M_i is Memory. This definition is a sum of the data received by the process of edge node and the computing process of data and tasks.

$$\begin{gathered} Activation\ status\{CPU,\ Memory\} \\ = Receiving\ Data\ Process\{C_i,\ M_i\} + Data\&Task\ Computation\ Process\{C_i, M_i\} \end{gathered} \tag{1a}$$

Equation (1b) shows that the dynamic offloading state is defined as the sum of the offloading state by overloads in the edge node and the collaboration offloading state for other nodes.

$$\begin{gathered} Dynamic\ offloading\ Status\{CPU,\ Memory\} \\ = Transmission\ offloading\ Process\{C_i,\ M_i\}+Receiving\ offloading\ Process\{C_i, M_i\}, \end{gathered} \tag{1b}$$

2.2. Definition of Assigning an Offloading Range

To assign the offloading range, the group should divide as being data-linked or task-linked. First, the offloading range based on memory is grouped in data connectivity and minimizes its overlapped offloading. In this computation, let D $= \{d_1,\ d_2,\ \ldots d_n\}$, where D is the set of edge gateway (device), and d_i is the *i*-th edge gateway (device). Equation (2) means the energy E generated by moving data from edge node $d1$ to $d2$. The $S(d1, d2)$ is the total amount of data that must transmit between nodes; hence, the group's energy consumption based on data connectivity is proportional to the amount of data. Therefore, the minimum range of transmitted energy is extracting by the amount of data after selecting collaboration edge nodes.

$$E(d1, d2) = ((w \times S(d1, d2)) \div bandwidth) \times Power_{wifi}, \tag{2}$$

Second, minimizing the optimum edge node's performance time includes offloading by grouping task connectivity based on *CPU* for the offloading range. Because of various *CPU* capacities according to the slack space, each edge node has different processing speeds. Therefore, it is essential to assign the offloading range because the total processing time is dependent on how to use the *CPU*. Equation (3) shows the difference of processing time between the existing node and the collaborating node of the group based on task connectivity for offloading. The $t_{o(i)}$ is the consumed time in an overloaded edge node, $\mathrm{t}_{\mathrm{c}(i)}$ is the consumed time in a collaborated node. The w is a weighted factor that depends on the slack space of the collaboration node. As long as the slack space increases, the weighted factor (w) also increases, so its consumed time becomes rapidly faster than the overload edge. Therefore, the range with a maximum weighted factor must extract as the offloading range by selecting the collaboration edge node.

$$t_{c(i)} = t_{o(i)} \div w, \tag{3}$$

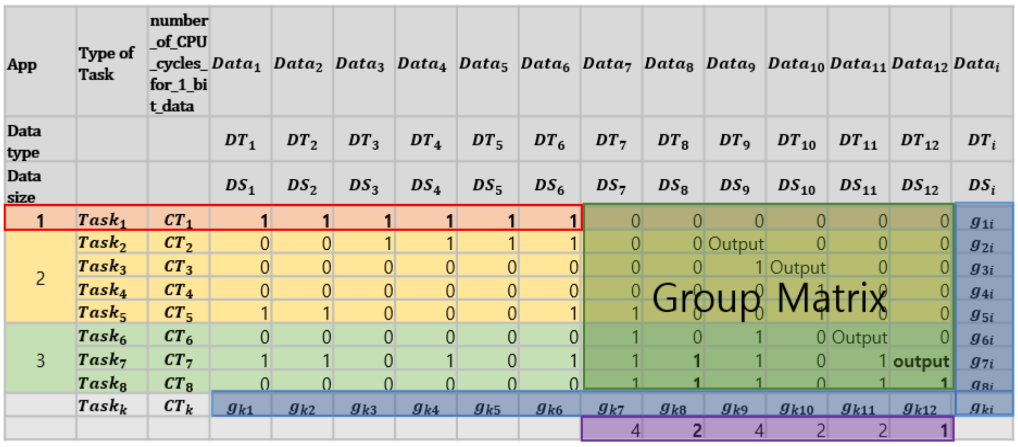
Therefore, the defined Equation needs to compromise (CPU, Memory) in the range of data and task. It aims to extract the range of minimizing consumption energy and final time for offloading.

3. LODO (Linked-Object Dynamic Offloading) Algorithm

The LODO algorithm provides two cooperation offloading options depending on the cause of the overload. In Section 2.1, the cause of overload according to the state of an overloaded edge node through monitoring was definition. When a memory overload problem occurs, executing the algorithm using the data-linked offload method described in Section 3.1. If the cause of the overload is in the CPU, execute the task-linked offloading option described in Section 3.2.

3.1. Data-Linked Algorithm

The Data-linked offloading (ref. Algorithm 1), which concerns data correlations, improves the energy efficiency by choosing the data redundancy based on memory, the range in minimizing energy during transmitting, and the collaboration node. Create a collaboration data group based on the currently executed data. Figure 3 explains how to create a group when performing offloading based on a scenario that allows Task1 to be performing at the existing edge. Derivation of the task-based offloading range is the least frequent of the remaining data, excluding the data used in Task1. It is possible to set the task priority according to the existing domain and create a list expression task offloading according to the criteria for the task and the data that is reducing due to offloading. For example, offloading the least frequent Task8 reduces the number of data12 on existing edge nodes. Offloading with Task7 to free up additional memory space will free up the amount of data8. A list of expression tasks is an input to the algorithm's data dependency groups. The output categorizes the minor offloading group, the collaboration node, and the total consumed memory.



App	Type of Task	number_of_CPU_cycles_for_1_bit_data	$Data_1$	$Data_2$	$Data_3$	$Data_4$	$Data_5$	$Data_6$	$Data_7$	$Data_8$	$Data_9$	$Data_{10}$	$Data_{11}$	$Data_{12}$	$Data_i$
Data type			DT_1	DT_2	DT_3	DT_4	DT_5	DT_6	DT_7	DT_8	DT_9	DT_{10}	DT_{11}	DT_{12}	DT_i
Data size			DS_1	DS_2	DS_3	DS_4	DS_5	DS_6	DS_7	DS_8	DS_9	DS_{10}	DS_{11}	DS_{12}	DS_i
1	$Task_1$	CT_1	1	1	1	1	1	1	0	0	0	0	0	0	g_{1i}
2	$Task_2$	CT_2	0	0	1	1	1	1	0	0	Output	0	0	0	g_{2i}
	$Task_3$	CT_3	0	0	0	0	0	0	0	0	1	Output	0	0	g_{3i}
	$Task_4$	CT_4	0	0	0	0	0	0	0	0	0	1	0	0	g_{4i}
	$Task_5$	CT_5	1	1	0	0	0	1	1	0	0	1	0	0	g_{5i}
3	$Task_6$	CT_6	0	0	0	0	0	0	1	0	1	0	Output	0	g_{6i}
	$Task_7$	CT_7	1	1	0	1	0	1	1	1	1	0	1	output	g_{7i}
	$Task_8$	CT_8	0	0	0	0	0	0	1	1	1	0	1	1	g_{8i}
	$Task_k$	CT_k	g_{k1}	g_{k2}	g_{k3}	g_{k4}	g_{k5}	g_{k6}	g_{k7}	g_{k8}	g_{k9}	g_{k10}	g_{k11}	g_{k12}	g_{ki}
									4	2	4	2	2	1	

Figure 3. List expression task based on data-centric offloading.

$$\text{Amount of memory in } List_j = \sum_{i=1}^{list[2]} g_{ki} * DT_{ki} * DS_{ki}, \tag{4}$$

$$\text{Amount of } CPU \text{ in } List_j = \sum_{\prime i=1}^{list[2]} g_{ki} * DT_{ki} * DS_{ki} * CT_{ki}, \tag{5}$$

Each group's memory and *CPU* usage can calculate via (Equation (4)) and (Equation (5)). g_{ki} is whether the *i*-th data used in the *k*-th task is used, and DT_{ki} is the data type. DS_{ki} is the amount of sensing data that is instantaneously accumulated as a data size. CT_{ki} is a 'number of *CPU* cycles for 1-bit data' value and is the *CPU* value used in each task. The calculated Equations (4) and (5) values changes depending on the task and data included in the corresponding list. Through the calculated value, the "possible offloading list" is extracting by matching the *CPU* usage rate that is not larger than the idle space of the collaboration edge node. In other words, depending on the cause of the overload, if the memory is insufficient, the data relevance list that can secure the maximum memory is selected in the offload range. The selected data relevance list becomes the "data dependency group," which is the input to the algorithm. The list of feasible groups is sort by examining their executions in collaboration nodes, and the offloading is processed in each node sequentially. If slack energy in the collaboration node is not enough, it could skip out to the next node for offloading. The Data-centric Offloading stops in the case of solving the overload issue. Otherwise, it keeps progressing to offloading.

Algorithm 1 Data-linked offloading

```
   Input: Next Execution, Data Dependency Group Output: Offloading Group,
   Collaboration Edge Node, TotalMemory
1  Step 1: Overload detection and determination of cause.
2  IF Status of EN = Out of memory problem ## result of activation state
3     NextExecution = Data-linkedOffloading
4     OffloadingApply = True
5  ELSE
6     OffloadingApply = False
7  Step 2) Calculate group capabilities based on data
8  FOR I = 1 to length(data dependency group)
9        $g_j$ = data dependency group
10       GroupList(Data_list, Total_memory) = $g_j$
11 END FOR
12 Step 3) Perform offloading after extracting possible groups based on collaboration nodes
13 WHILE OffloadingApply
14       FOR $j$ = 1 to $k$
15       IF g_j(Total_memory) $\leq$ Collaboration_EN(Slack Memory) $-$ (100-threshold)
16       possiblelist = $g_j$(Data_list, Total_memory
17 END FOR
18 DO max(possiblelist) offloading to Colaboration_EN
19       IF EN(slack_memory) $\leq$ threshold
20       OffloadingApply = False
21 RETURN {possible list, collaboration edge node, Total_memory, Total_energy}
```

3.2. Task-Linked Algorithm

The task-linked algorithm, which concerns task correlations, reduces the overall time by selecting the most considerable *CPU*, the range in increasing energy efficiency during processing, and the collaboration node (ref. Algorithm 2). Create collaboration groups based on tasks to performing and data to be using. Figure 4 shows a subordinate system consisting of 8 tasks. In this way, the application is complexly configuring. Some tasks use the output value of each task as an input value. A dependency system is a workflow made up of dependent tasks that interact between modules. Task2 creates data9, which is the input value of Task3, Task6, Task7, and Task8. Based on task connectivity, Task3, Task6, Task7, and Task8 cannot start execution until Task2 is executing and the output is displayed.

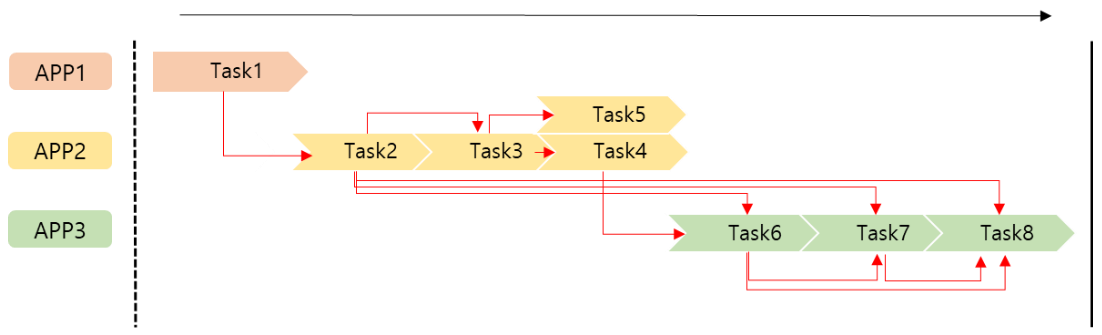


Figure 4. Workflow and Task Connectivity.

When performing tasks, an offloading group is creating in consideration of task relevance. Groups that consider task relevance are composed of one or more tasks, as shown in Figure 5.

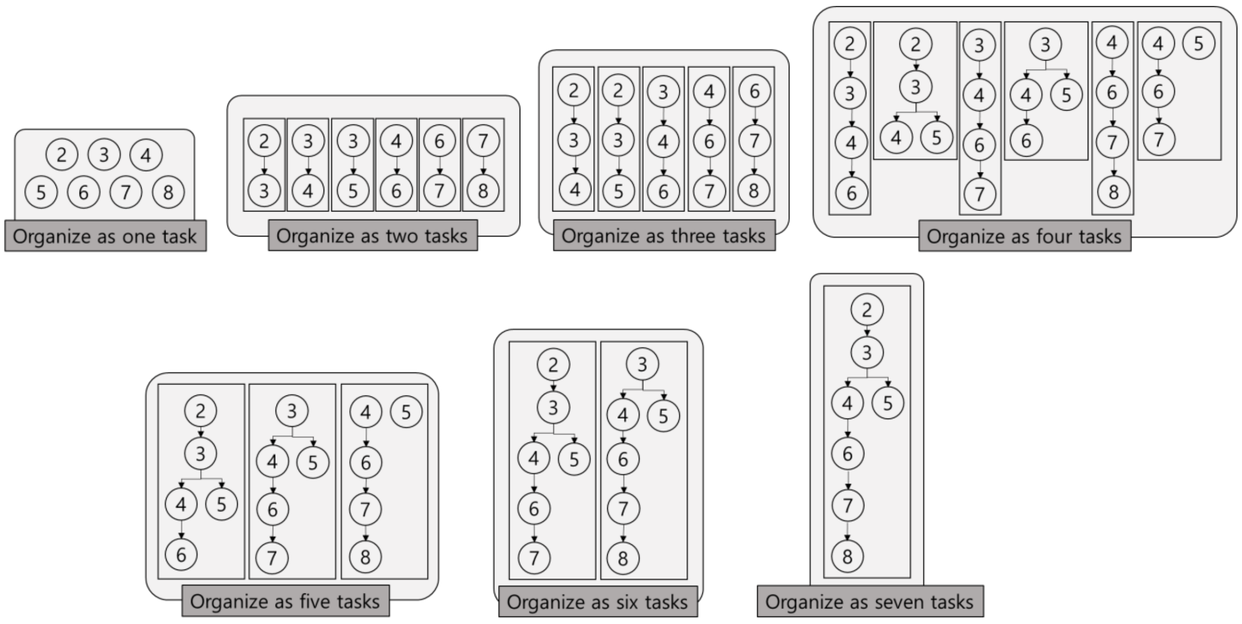


Figure 5. Offloading group of Task relevance.

To select an offloading range is necessary to calculate the mobility of data generated during offloading. Figure 6 illustrates the importance of task connectivity group-based offloading through task mobility. In Case 1, Task3 and Task4 are offloading, and in Case2, Task3 and Task6 are offloading. Task3 and Task4 use the result of Task2 together, and the output of Task3 becomes the input of Task4 and proceeds sequentially. Also, since Task4's output is used by Task5 and Task6 simultaneously, the total number of movements is two times. However, the input value of Task3 and Task6 is common to Task2, but Task6 requires the output value of Task4 as an input value. Each output value is also using as an input value for other tasks. Thus, the total number of moves is four. Even if the same amount of data is using, the delay time increases as the total number of movements increases. Hence, the available memory and *CPU* are calculating through Equations (4) and (5), appropriate lists are extracting, and the group with the least mobility is select as the offloading range.

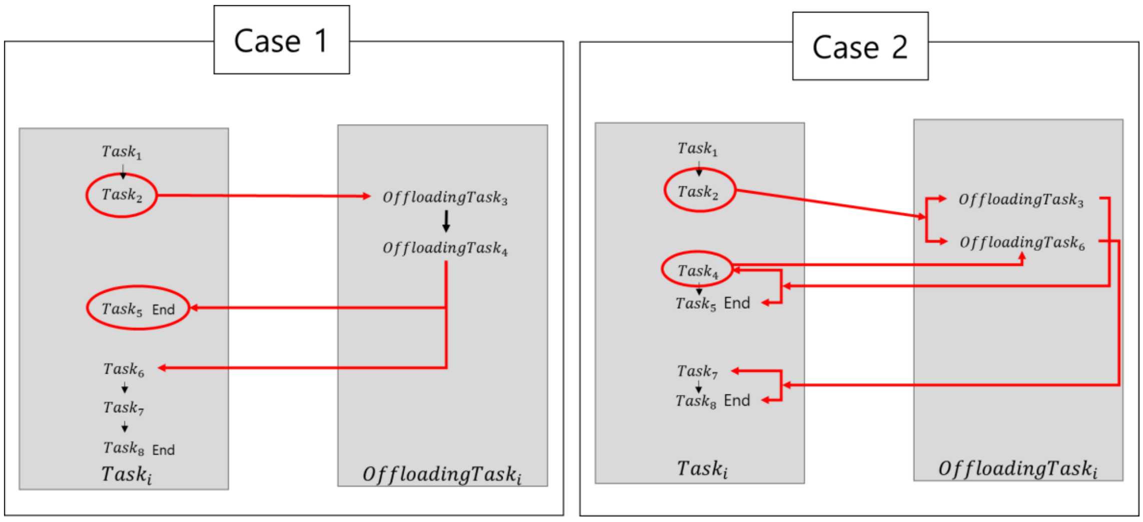


Figure 6. Task connectivity group-based offloading through task mobility.

A grouped list of task dependencies is the input to the algorithm. The task connectivity list described above becomes the "task dependency group," which is the input to the algorithm. The output classifies the least offloading group, the collaboration node, and the overall time. The list of feasible groups is sorting by examining their executions in collaboration nodes, and the offloading is processed in each node sequentially. Offloading continues until the overload problem is resolving, and if slack energy in the collaboration node is not enough, it could skip out to the next node for offloading. Total output time means the time until the end of offloading for a specific application and is calculated from the number of moves and the amount of data.

Algorithm 2 Task-Linked offloading

```
    Input: NextExecution, Task Dependency Group
    Output: Offloading Group, Total Time(EndPoint)
1   Step 1) Overload detection and determination of the cause.
2   IF Status of EN = Out of CPU problem
3       NextExecution = Task-linkedOffloading
4       OffloadingApply = True
5   ELSE
6   OffloadingApply = False
7   Step 2) Calculate group capabilities based on task
8   FOR I = 1 to length(task dependency group)
9       $g_j$ = task dependency group
10      GroupList(Task_list, Total_CPU) = $g_j$
11  END FOR
12  Step 3) Perform offloading after extracting possible groups based on collaboration nodes
13  WHILE OffloadingApply
        FOR $j$ = 1 to k
            IF g_j(Total_Time) $\leq$ Collaboration_EN(Slack space) $-$ (100-threshold)
              possiblelist = $g_j$(Task_list, Total_Time)
    END FOR
    DO max(possiblelist) offloading to Collaboration_EN
        IF EN(slack_space) $\leq$ threshold
        OffloadingApply = False
    RETURN{possiblelist, collaboration edge node, Total_CPU, Total_time}
```

4. Scenario and Results

This section provides an experimental scenario and the LODO algorithm's performance that proposes in data-linked and task-linked algorithms. The performance of the algorithm is tested based on the variables and tasks used in the forest fires scenario and the relationship between them. Based on the algorithm experiment results, the evaluation scenario includes memory usage and execution time. Table 1 lists tasks and variables used in forest fire response scenarios. In practice, the quantity and size of a variable can be the same or different.

Table 1. Tasks for forest fire response and data types used.

Application	Task	Data	Output
APP1. Fire Probability Prediction	Task 1. Fire probability and probability prediction	temperatures (data1), humidity (data2), fuel (data3), mount.terrain (data4), weather (data5), fuel (data3), for geography (data6)	Forest fire Probability (data13)
APP2. Diffusion Range Prediction	Task 2. Diffusion rate	fuel (data3), mount.terrain (data4), weather (data5), for geography (data6)	Diffusion rate (data9)
	Task 3. Forest fire intensity	diffusion rate (data9)	Fire intensity (data10)
	Task 4. Flame height, Fire type prediction	fire intensity (data10)	Fire type (data14)
	Task 5. Fire direction, Diffusion area prediction	temperatures (data1), humidity (data2), for.geography (data6), wind speed (data7), fire intensity (data10)	End
APP3. Diffusion Location Prediction	Task 6. Flame length	fire type (data14), wind speed (data7), diffusion rate (data9)	Flame length (data11)
	Task 7. Non-combustible material lift height calculation	temperatures (data1), humidity (data2), mount.terrain (data4), for.geography (data6), wind speed (data7), wind direction (data8), diffusion rate (data9)	Flame height (data12)
	Task 8. Distance for non-combustible mater. Fireworks Landing Position Prediction	wind speed (data7), wind direction (data8), diffusion rate (data9), flame length (data11), flame height (data12)	End

Tasks are prioritized based on what is essential or should do first. Task 1 is the most basic analysis that starts in the forest fire scenario, so it should be performing on the edge node. In addition, the priority is high because it is performing in real-time. Before the data processing process and algorithm apply, the offloading range list becomes a subset of 127 for all tasks except for Task 1. As the number of tasks and variables increases, the selectable range of offloading becomes more diverse.

In case of memory overload, a list extracted based on data association is using. Table 2 is a data correlation list that derives an offloading range that can quickly acquire memory based on data.

Table 2. Data-linked list.

Task	Acquire Memory
Task8	data12
Task8, Task7	data12, data11, data8
Task8, Task7, Task5 Task6	data12, data11, data8, data7
Task8, Task7, Task3, Task6	data12, data11, data8, data9
Task8, Task7, Task4, Task5	data12, data11, data8, data10

In case of *CPU* overload, a list extracted based on task association is using. As with the data center, except for Task 1, the connectivity between the remaining tasks creates a group that reduces mobility between edges and secure *CPU* space. Table 3 is a task connectivity list that derives an offloading range that reduces mobility while quickly obtaining *CPU* space centered on tasks. Figure 7 shows the total number of moves and the task connectivity list (red dots). In the figure, the ox is each group corresponding to the task connectivity list, and oy is the number of moves that occur when offloading each group.

Table 3. Task-linked list.

	Task	Number of Moves
1	Task2	2
	Task3	2
	Task4	2
	Task5	2
	Task6	3
	Task7	3
	Task8	4
2	Task2-Task3	2
	Task3-Task4	2
	Task3-Task5	2
	Task4-Task6	3
	Task6-Task7	3
	Task7-Task8	3
3	Task2-Task3-Task4	2
	Task2-Task3-Task5	2
	Task3-Task4-Task6	2
	Task4-Task6-Task7	3
	Task6-Task7-Task8	3
4	Task2-Task3-Task4-Task6	2
	Task2-Task3-Task4-Task5	3
	Task3-Task4-Task6-Task7	2
	Task3-Task4-Task5-Task6	3
	Task4-Task6-Task7-Task8	3
	Task4-Task5-Task6-Task7	4
5	Task2-Task3-Task4-Task5-Task6	3
	Task3-Task4-Task5-Task6-Task7	3
	Task4-Task5-Task6-Task7-Task8	4
6	Task2-Task3-Task4-Task5-Task6-Task7	3
	Task3-Task4-Task5-Task6-Task7-Task8	3
7	Task2-Task3-Task4-Task5-Task6-Task7-Task8	3

Table 4 compares the range to which offloading is applying within the range of possible collaboration usage (Memory, *CPU* 70%) and the entire range based on the lists in Tables 2 and 3. "Offloading Transmission" is the memory size calculated based on the data included in the offloading group. "Offloading performance" is a value calculated based on the number of *CPU* cycles for 1-bit data, and the value varies depending on the data used in the task and the function of the task. With "Data-linked algorithm groups," the average data movement is 1,574,156 (Kb), which offloads more data than before applying the algorithm, but the average memory securing is 8.15%, which is faster than before applying the algorithm. Suppose the "task-oriented algorithm group" is used. In that case, the offloading task group sends a smaller amount than before applying the algorithm at an average of 4.19 (GHz). It is possible to secure free space faster, and the average number of moves is an average of 2.63 Cycles enable fast processing.

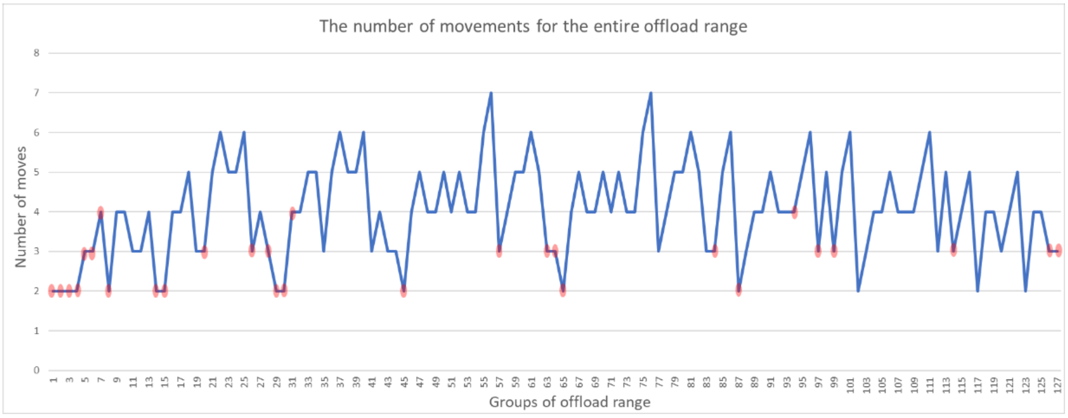


Figure 7. Graph the number of movements for the entire offload range.

Table 4. Data & Task algorithm performance evaluation.

	All Group	Data-Linked Algorithm Group	Task-Linked Algorithm Group
Offloading Transmission (Memory)	1,199,392 (Kb)	1,574,156 (Kb)	921,015.1 (Kb)
Offloading Performance(*CPU*)	5.72 (GHz)	8.67 (GHz)	4.19 (GHz)
Percentage of memory available	45,332.69 (3.78%)	212,775.5 (8.15%)	30,394.8 (3.30%)
The average number of moves	3.72	3.55	2.63

5. Discussion

Spatial dependence has two meanings. The first is that the data generated by one sensor is not used in only one task or app and is can use in multiple spaces. For example, diffusion direction data is generating by combining wind, slope, and temperature data with circumstances such as earthquakes affecting different regions at time intervals. The second is task dependencies. Task dependencies include the order of tasks as well as data correlation between tasks. In refs. [21,22], the data dependency, the output of the task located in Area A, can be entered in Area B.

They should process several tasks, such as knowing the types of forest fires, predicting how the wind changes, and identifying how ignition materials are distributing in fire areas by season, temperature, humidity. Additionally, some cases have data with two or more tasks used simultaneously or including complicatedly mixed order of tasks. For example, to determine the diffusion range with the location of forest fires. However, as the accurate diffusion range and location could identifying many tasks in various areas could be linked and applied to each other. In other words, multiple tasks do not follow in a series of workflows but are spread in many branches or entangled like spider webs.

However, most offloading does consider only a series of processing in [23] and does not include correlations and interactions of data and tasks. One or more outputs can be applying to the next task after starting four different tasks simultaneously, or new input that changes the result executed already. Because output data transferred from a server to a local device is much smaller than input data, the time overhead of a backhaul link could be ignorant [24,25]. This is considered only a series of dynamic applications, not static tasks. If one output data can be the input of many tasks, the increase of task number cause to increase both redundant data and output data; hence the overhead due to another delay and energy consumption from a transmission cannot be disregarded.

Correlations of data and tasks have to consider for dividing tasks of granularity and dependency in various circumstances. Therefore, we proposed a collaboration LODO algorithm that determines idle space and offloads data and tasks based on spatial dependencies.

6. Conclusions

This paper considers collaboration edge computing in offloading with an edge node that can collaborate with tasks. We proposed an energy-efficient LODO algorithm to extract the scope and offload of collaboration nodes to save energy and reduce execution time at the edge nodes. Formulated hybrid states in an edge node could predict overloads through monitoring and applied in the LODO algorithm. Furthermore, in selecting an offload range by considering data correlations and task connectivity, the LODO algorithm reduces data redundancy and delays and minimizes energy consumptions during offloading. Therefore, a collaboration offloading model based on the LODO algorithm minimizes the energy of the entire edge node so that it is more efficient to execute within a short time.

Author Contributions: Conceptualization, S.K., J.K. and Y.Y.; methodology, S.K.; formal analysis, S.K. and J.K.; investigation, J.K.; resources, S.K. and J.K.; data curation, J.K.; writing—original draft preparation, S.K. and J.K.; writing—review and editing, S.K. visualization, J.K.; supervision, Y.Y.; project administration, S.K.; funding acquisition, Y.Y. and S.K. This J.K. author contributed equally to this study as co-first authors. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (2019R1I1A1A01064054). This research was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (2018R1D1A1B07047112).

Data Availability Statement: Data is describing within the article. The data that support the findings of this study are available from the corresponding author upon reasonable request.

Acknowledgments: The writers would like to thank the editor and anonymous reviewers for their helpful comments for improving the quality of this paper.

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

References

1. El-Sayed, H.; Sankar, S.; Prasad, M.; Puthal, D.; Gupta, A.; Mohanty, M.; Lin, C.-T. Edge of Things: The big picture on the Integration of Edge, IoT and the cloud in a distributed computing environment. *IEEE Access* **2018**, *6*, 1706–1711. [CrossRef]
2. Li, X.; Qin, Y.; Zhou, H.; Cheng, Y.; Zhang, Z.; Ai, Z. Intelligent Rapid adaptive offloading algorithm for computational services in dynamic internet of things system. *Sensors* **2019**, *19*, 3423. [CrossRef] [PubMed]
3. Premsankar, G.; Di Francesco, M.; Taleb, T. Edge computing for the Internet of Things: A case study. *IEEE Internet Things J.* **2018**, *5*, 1275–1284. [CrossRef]
4. Wang, J.; Pan, J.; Esposito, F.; Calyam, P.; Yang, Z.; Mohapatra, P. Edge cloud offloading algorithms: Issues, methods, and perspectives. *ACM Comput. Surv.* **2019**, *52*, 2. [CrossRef]
5. Yousefpour, A.; Fung, C.; Nguyen, T.; Kadiyala, K.; Jalali, F.; Niakanlahiji, A.; Kong, J.; Jue, J.P. All one needs to know about fog computing and related edge computing paradigms: A complete survey. *J. Syst. Archit.* **2019**, *98*, 289–330. [CrossRef]
6. Cui, Y.; Ma, X.; Wang, H.; Stojmenovic, I.; Liu, J. A survey of energy efficient wireless transmission and modeling in mobile cloud computing. *Mobile Netw. Appl.* **2013**, *18*, 148–155. [CrossRef]
7. Mehrabi, M.; You, D.; Latzko, V.; Salah, H.; Reisslein, M.; Fitzek, F.H.P. Device-enhanced MEC: Multi-access edge computing (MEC) aided by end device computation and caching: A survey. *IEEE Access* **2019**, *7*, 166079–166108. [CrossRef]
8. Kai, C.; Zhou, H.; Yi, Y.; Huang, W. Collaborative Cloud-Edge-End Task Offloading in Mobile-Edge Computing Networks With Limited Communication Capability. *IEEE Trans. Cogn. Commun. Netw.* **2021**, *7*, 624–634. [CrossRef]
9. Huang, W.; Huang, Y.; He, S.; Yang, L. Cloud and edge multicast beamforming for cache-enabled ultra-dense networks. *IEEE Trans. Veh. Technol.* **2020**, *69*, 3481–3485. [CrossRef]

10. Ren, J.; Yu, G.; Cai, Y.; He, Y. Latency optimization for resource allocation in mobile-edge computation offloading. *IEEE Trans. Wireless Commun.* **2018**, *17*, 5506–5519. [CrossRef]
11. Kao, Y.; Krishnamachari, B.; Ra, M.; Bai, F. Hermes: Latency optimal task assignment for resource-constrained mobile computing. *IEEE Trans. Mobile Comput.* **2017**, *16*, 3056–3069. [CrossRef]
12. Auluck, N.; Azim, A.; Fizza, K. Improving the schedulability of real-time tasks using fog computing. *IEEE Trans. Serv. Comput.* **2019**. [CrossRef]
13. Li, H.; Ota, K.; Dong, M. Learning IoT in edge: Deep learning for the Internet of Things with edge computing. *IEEE Netw.* **2018**, *32*, 96–101. [CrossRef]
14. Tran, T.X.; Pompili, D. Joint task offloading and resource allocation for multi-server mobile-edge computing networks. *IEEE Trans. Veh. Technol.* **2019**, *68*, 856–868. [CrossRef]
15. Zhao, J.; Li, Q.; Gong, Y.; Zhang, K. Computation Offloading and Resource Allocation for Cloud Assisted Mobile Edge Computing in Vehicular Networks. *IEEE Trans. Veh. Technol.* **2019**, *68*, 7944–7956. [CrossRef]
16. Mao, S.; Leng, S.; Zhang, Y. Joint communication and computation resource optimization for NOMA-assisted mobile edge computing. In Proceedings of the IEEE International Conference Communication (ICC), Shanghai, China, 20–24 May 2019; pp. 1–6.
17. Haber, E.; Nguyen, T.M.; Assi, C.; Ajib, W. Macro-cell assisted task offloading in MEC-based heterogeneous networks with wireless backhaul. *IEEE Trans. Netw. Serv. Manag.* **2019**, *16*, 1754–1767. [CrossRef]
18. Hossain, M.D.; Sultana, T.; Nguyen, V.; Nguyen, T.D.; Huynh, L.N.; Huh, E.N. Fuzzy Based Collaborative Task Offloading Scheme in the Densely Deployed Small-Cell Networks with Multi-Access Edge Computing. *Appl. Sci.* **2020**, *10*, 3115. [CrossRef]
19. Ren, J.; Yu, G.; Cai, Y.; He, Y. Collaborative cloud and edge computing for latency minimization. *IEEE Trans. Veh. Technol.* **2019**, *68*, 5031–5044. [CrossRef]
20. Kang, J.; Kim, S.; Kim, J.; Sung, N.; Yoon, Y. Dynamic Offloading Model for Distributed Collaboration in Edge Computing: A Use Case on Forest Fires Management. *Appl. Sci.* **2020**, *10*, 2334. [CrossRef]
21. Zhang, Z.; Wu, J.; Chen, L.; Jiang, G.; Lam, S.K. Collaborative task offloading with computation result reusing for mobile edge computing. *Comput. J.* **2019**, *62*, 1450–1462. [CrossRef]
22. Zhang, D.; Ma, Y.; Zheng, C.; Zhang, Y.; Hu, X.H.; Wang, D. Cooperative-competitive task allocation in edge computing for delay-sensitive social sensing. In Proceedings of the 2018 IEEE/ACM Symposium on Edge Computing (SEC), Seattle, WA, USA, 25–27 October 2018; pp. 243–259.
23. Ning, Z.; Dong, P.; Kong, X.; Xia, F. A cooperative partial computation offloading scheme for mobile edge computing enabled Internet of Things. *IEEE Internet Things J.* **2018**, *6*, 4804–4814. [CrossRef]
24. Yang, L.; Dai, Z.; Li, K. An offloading strategy based on cloud and edge computing for industrial Internet. In Proceedings of the 2019 IEEE 21st International Conference on High Performance Computing and Communications, IEEE 17th International Conference on Smart City, IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS), Zhangjiajie, China, 10–12 August 2019.
25. He, B.; Bi, S.; Xing, H.; Lin, X. Collaborative computation offloading in wireless powered mobile-edge computing systems. In Proceedings of the 2019 IEEE Globecom Workshops (GC Wkshps), Waikoloa, HI, USA, 9–13 December 2019; pp. 1–7.