sensors

MDPI

Article

An Efficient Resource Allocation Strategy for Edge-Computing Based Environmental Monitoring System

Juan Fang [1,*], Juntao Hu [1], Jianhua Wei [1], Tong Liu [2] and Bo Wang [3]

1 Faculty of Information Technology, Beijing University of Technology, Beijing 100124, China; hujuntaochn@emails.bjut.edu.cn (J.H.); weijianhua@bjut.edu.cn (J.W.)
2 Beijing Computing Center, Beijing 100094, China; liutong@bcc.ac.cn
3 National Computer Network Emergency Response Technical Team, Coordination Center of China, Beijing 100096, China; wb@cert.org.cn
* Correspondence: fangjuan@bjut.edu.cn; Tel.: +86-139-1129-6256

Received: 30 September 2020; Accepted: 23 October 2020; Published: 28 October 2020

Abstract: The cloud computing and microsensor technology has greatly changed environmental monitoring, but it is difficult for cloud-computing based monitoring system to meet the computation demand of smaller monitoring granularity and increasing monitoring applications. As a novel computing paradigm, edge computing deals with this problem by deploying resource on edge network. However, the particularity of environmental monitoring applications is ignored by most previous studies. In this paper, we proposed a resource allocation algorithm and a task scheduling strategy to reduce the average completion latency of environmental monitoring application, when considering the characteristic of environmental monitoring system and dependency among task. Simulations are conducted, and the results show that compared with the traditional algorithms. With considering the emergency task, the proposed methods decrease the average completion latency by 21.6% in the best scenario.

Keywords: environmental monitoring; edge computing; resource allocation; task scheduling

1. Introduction

With the development of industry and the acceleration of urbanization, we must pay attention to environmental monitoring, because air quality has a paramount impact on human health. The collection and processing of environmental data are prerequisites for environmental monitoring and pollution warning. As a novel technology, the Internet of Things (IoT) is considered to be a pollution monitoring solution, which attracts attention from both academia and industry. In addition, cloud computing makes up the lack of computing resources and energy capacity in environmental monitoring terminals devices. In an environmental monitoring system based on IoT and cloud computing technology, the environmental data collected by the sensors are simply processed by the IoT devices and uploaded to the cloud server. Subsequently, the cloud server processes and analyzes the environmental data uploaded from all sensors. However, with the decrease of granularity of environmental monitoring and the increase of data analysis application amount, the centralized cloud computing architecture is facing challenges, such as high latency, low coverage, and lagged data transmission [1].

Edge computing, as a new computational paradigms, reduces the application completion latency and the energy consumption of data transmission by distributing cloud resources closer to where data generation [2]. Fang et al. [3] has proven that edge computing could enhance the real-time performance of service completion, with reducing computing load and power consumption in

the cloud, by offloading computing request to edge servers closer to the IoT device for execution. The authors in [4] proposed a generalized logical sphere(GLS) modeling scheme in order to avoid servers overload.

In edge-computing based environmental monitoring system, sensors collect the environmental data in real time and transmit them to the edge computing server in order to execute necessary processing and analysis, which reduces system energy consumption and network traffic [5]. The accuracy of environmental monitoring is affected by positioning problems of monitoring sensors. To solve this problem, Fei et al. [6] investigated advanced parameter prediction skills and proposed a a smart collaborative tracking scheme to improve particle filter approaches. A number of previous works have proven that resource allocation strategies could reduce task completion latency and energy consumption [7–9]. Fengxian et al. [10] design a genetic algorithm and particle swarm optimization-based algorithm to solve resource allocation problem. However, the dependency between subtasks was not considered in this study. Non-dominated sorting genetic algorithm II was adopted in order to realize multi-objective optimization to reduce the execution time and energy consumption of edge computing devices in paper [11]. Songtao et al. [12] has proven that resource allocation policy is determined by the computing workload of a task and the maximum completion time of its immediate predecessors. In [13], a smart collaborative automation (SCA) scheme was proposed in order to improve resource usage. Du et al. [14] proposed an algorithm, which obtained allocation decision via semidefinite relaxation and randomization, but the communication among subtasks are ignored by this work. The authors in [15] took dependencies among subtasks into account, and proposed an multistage greedy adjustment (MSGA) algorithm to solve the task allocation problem. They minimized the completion time of application by jointly considering the network flows and tasks. Laizhong et al. [16] focus on the task offloading problem in order to find out the optimal tradeoff between task completion latency and energy consumption. They proposed a modified fast and elitist nondominated sorting genetic algorithm to solve the offloading problem. However, in this work, the tasks are considered to be undivided, which wastes the parallel computing capability of edge computing servers. Authors in [17] created un-excuted task queue at the MEC server and proposed an online algorithm to allocate resources. Pereira et al. [18] propose an allocation and management resources mechanism to reduce the model complexity of resource allocation algorithm. The authors in [19] take the dependency between task into account, and proposed a deep reinforcement learning approach to make offloading decision, the difference among tasks is ignored by this work. Xu et al. [20] adopt Non-dominated Sorting Genetic Algorithm II to shorten the resource allocating time of the computing tasks and reduce the energy consumption of the edge computing servers, but this work did not consider the dependencies among tasks.

In most previous studies, the particularity of environmental monitoring applications has not been considered. For example, air pollution tracing analysis can only be performed on certain servers with wind direction information of the region. In addition, as the chromosome size increases, the time complexity of the resource allocation algorithm increases exponentially [10]. In order to reduce the solution space of genetic algorithm, we cluster subtasks in genetic algorithm by k-means algorithm, which makes the resource allocation algorithm converge faster.

This study targets the minimum time cost of task completion in environmental monitoring system by fully utilizing the parallel computing capacity of edge computing. The main contributions of this work are outlined, as follows:

- We first introduce the computing model of task with dependency and formulate resource allocation problem. Taking the communication among subtasks into account, we propose a resource allocation algorithm based on GA. The proposed algorithm can reach a better sollution than GA under a same generation, by clustering the subtasks with heavy dependency and decreasing the size of sollution space.
- The impact of dependency among subtasks on completion time has been discussed in the contribution above. To solve this problem, a task scheduling strategy proposed that reordering

the task queue on edge computing server according to the priority of subtasks can reduce the average task completion latency when considering emergency task.

The remainder of this paper is organized, as follows. Section 2 introduces the system model and formulates the resource allocation problem. Section 3 introduces the proposed resource allocation algorithm and task scheduling strategy. Section 4 describes a simulation analysis of the proposed algorithm and compares it with the traditional algorithm. Section 5 concludes the study.

2. System and Computation Model

This section describes the system model of edge-computing based environmental monitoring system and formulates the resources allocation problem.

2.1. System Model

In the traditional centralized cloud computing system, the data procession is mainly on cloud computing server. IoT devices collect environmental data periodically and transmit it to the edge gateways and cloud server. After processing on cloud, the results are returned to end devices. By deploying resources (i.e., computing resource and storage resource) on edge gateways, they can process some data locally, then return the processing results to IoT devices directly or transmit data to cloud for further processing.

In this work, we extend the edge-cloud heterogeneous collaborative network architecture in [3], which reduces the network bandwidth occupation, as shown in Figure 1. This network architecture consists of four parts: cloud layer, region layer, roadside layer, and IoT device layer. The devices in IoT layer collect environmental data and upload task to the directly connected roadside server in roadside layer. The roadside servers process data collaboratively or transmit them to a regional server according to resource allocation policy. Regional servers has more computing resources than roadside servers and it is responsible for executing regional tasks. When the devices in the edge network are overloaded, the task will be uploaded to cloud server. After completing all of the tasks uploaded from IoT devices, partial data will recored into databases on cloud server. Fei et al. [21] proposed a smart collaborative distribution scheme to solve the data privacy-preserving problem in such a distribution system.

We assume that there are E environmental monitoring sensors that are located in a region, denoted by a set of $\mathcal{E} = \{1, 2, 3, ..., E\}$, which are installed on roadside or indoors to collect environmental data (i.e., PM2.5, PM10, SO2, NO, CO, and temperature, etc.) and offload environmental monitoring task into the edge computing system.

For each monitoring region, $\mathcal{S}$ represents a set of computing resources, including a set of roadside edge computing servers S_{road}, a set of regional servers S_{region}, and a cloud computing server S_{cloud} as shown in Figure 1. Each server S_i has its own resource (computing resource, bandwidth, and device layer) can be represented, as follow:

$$Res_i = \{f_i, Bw_i, flag_i\}, \tag{1}$$

where $flag_i$ indicates which layer the server S_i located on. Which can be denoted as follow:

$$flag_i = \begin{cases} 0 & \text{if } S_i \in S_{road}, \\ 1 & \text{if } S_i \in S_{region}, \\ 2 & \text{if } S_i \in S_{cloud}. \end{cases} \tag{2}$$

The virtual machines adopts time sharing policy, and execute various task circularly. A space sharing strategy was adopted in order to task processing mechanism for each virtual machine. Before being executed, the arriving task waits in the waiting queue as shown in Figure 2.

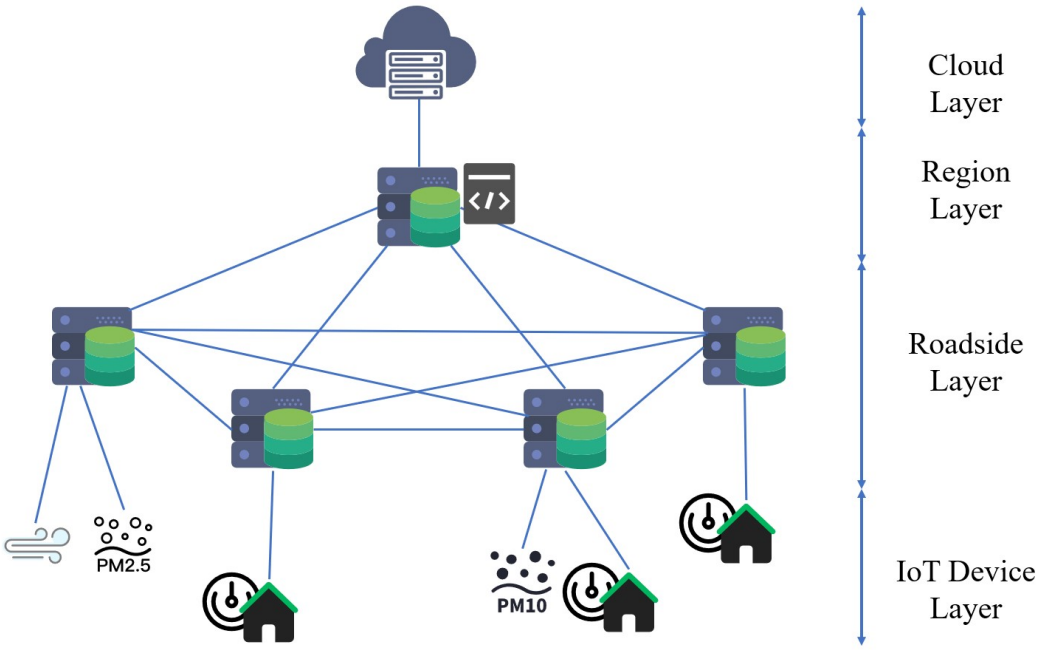

Figure 1. Edge-computing based environmental monitoring system model.

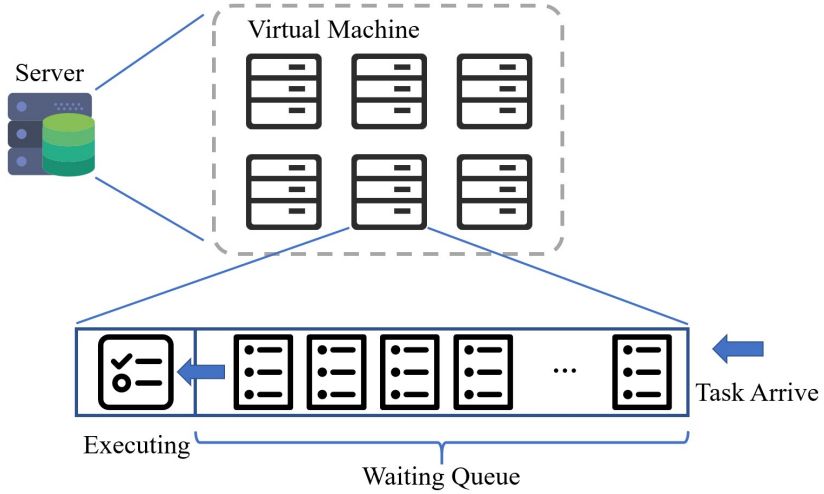

Figure 2. Server Model.

2.2. Application Model

We utilize a directed acyclic graph $A = (M, D)$ to describe the dependency relationship among these subtasks. M represents the modules of an environmental monitoring application, and D indicates the data dependency between two modules. Each module $m_i \in M$ has its own resource requirement (computing resource requirements , bandwidth requirement, and minimum processing layer), which can be denoted as follows:

$$m_i = \{l_i, Bw_i, layer_i\}, \tag{3}$$

where $layer_i \in \{0, 1, 2\}$ indicates module m_i can only be processed on which layer or above.

Each dependency$d_i \in D$ can be denoted as $d_i = \{m_{src}, m_{dest}, data_i\}$, which describe that the processing result with $data_i$ bytes of m_{src} is the input data of m_{dest}. Each $m_i \in M$ represents a subtask and a dependency $d_j = \{m_i, m_k, data_j\}$ indicates the precedence constraint between subtask m_i and m_k, such that subtask m_k cannot start execution until m_i executed.

2.3. Latency Model

We let $f_{n,m}$ denote the computation resource of server S_n which is allocated to subtask m. The execution latency of subtask m is given by:

$$T_{n,m}^{exc} = l_m f_{n,m}^{-1} \tag{4}$$

Let $T_{m,n,w}^{trans}$ denote the subtask m transmission latency from server n to w. Subsequently, we discuss the impact of dependency among subtasks on resource allocation algorithm. The dependencies among subtasks constraining a subtask cannot be processed until all of its predecessors have already been executed. Therefore, resource allocation algorithm must consider the latency of subtasks waiting for their predecessors completed. Subsequently, we give the definition of ready time of a subtask [22].

The ready time of a subtask is defined as the earliest time when all immediate predecessors of the subtask have completed execution. Therefore, the ready time of subtask m which executed on server w is given by

$$RT_m = \max_{k \in pred(m)} \{RT_k + T_{k,j}^{exc} + T_{k,j,w}^{trans}\}, \tag{5}$$

where $pred(m)$ denotes the set of immediate predecessors of subtask m.

2.4. Resource Allocation Problem Formulation

In this section, we formulated the resource allocation problem that was subject to several constraints. Each task t of an environmental monitoring application a can be denoted as $t = \{\mathcal{T}, t_{deadline}\}$, where $\mathcal{T}$ indicates a set of subtask correspond with modules M of a. To ensure the Quality of Service (QoS) of environmental monitoring application, task t should be completed before $t_{deadline}$. In terms of the limited computation resource of edge computing servers, we consider the resource allocation problem: decide which server should execute the subtask $t_i \in \mathcal{T}$. We defined the integer decision variable $x_i \in \mathcal{S}$ that indicates subtask t_i executed on server x_i, where $\mathcal{S}$ represents a set of servers that can serve the region where the sensor located. The variable of resource allocation is as follows: $X = \{x_1, x_2, ..., x_m\}$. We propose minimizing the task completion latency T_{comp}.

Formally, the resource allocation problem can be formulated, as follows:

$$\begin{aligned}
\underset{X}{\text{minimize}} \quad & \max_{k \in \mathcal{T}} \{RT_k\} + T_{k,x_k}^{c} && (6)\\
\text{subject to:} \quad & \forall i \in \mathcal{S}, \forall j \in \mathcal{T}, && (7)\\
& T_{comp} \leq t_{deadline}, && (8)\\
& flag_{x_i} \geq level_j, j \in \mathcal{T}, && (9)\\
& ST_j \geq \max_{k \in pred(j)} \{RT_k + T_{k,s}^{exc} + T_{k,s,x_j}^{trans},\} && (10)
\end{aligned}$$

Constraint (8) is a completion time constraint that restricts the total completion time of all the subtasks of an environmental monitoring application that is limited by the required maximum completion time $t_{deadline}$, which is determined by the type of application. The deadline of routine monitoring application (e.g., generating pollution map) is typically larger than emergency applications (e.g., pollution warning). Constraint (9) states subtask t_j can only be executed on a certain devices on $layer_j$ or above. ST_j in constraint (10) describes that the start time of subtask j is not earlier than the latest time of its predecessors executed and transmitted to server x_j.

3. Resource Allocation Algorithm and Task Scheduling Strategy

As aforementioned, to reduce average task completion time, the resource allocation algorithm and task scheduling policy need to consider the dependencies among subtasks. In this section, an improved GA based resource allocation algorithm and a task scheduling policy were proposed. When a task is

offloaded to the edge computing network, the server implements the resource allocation algorithm for this task, and outputs a resource allocation policy that specifies the processing server of each subtask. During the operation of the edge computing system, each edge server periodically sorts the task queue by implementing the proposed task scheduling strategy.

3.1. GA Based Resource Allocation Algorithm

In term of the cost of genetic algorithm is positively correlated with the amount of subtasks and specific subtasks of environmental monitoring application can only be executed on certain servers. In this paper, we propose a suboptimal algorithm, named combined genetic algorithm (CGA), as shown in Algorithm 1.

The Flow of CGA

A number of works use the Genetic Algorithm to deal with the resource allocation problem. Genetic algorithm adopts the idea of survival of the fittest from the theory of evolution. It performs crossover and mutation operations on a population of solutions until reaching the optimal solution. However, with the increase in the number of subtasks to be allocated, the time cost of genetic algorithm increases rapidly. In addition, resource allocation policies tend to assign more dependent subtasks to the same server. If two subtasks with heavy dependency are divided to be executed on the same server, the communication latency between the two subtasks can be ignored. In addition, calculating the resource allocation result also consumes the resources of edge computing system, and it is usually calculated on roadside server with fewer computing resource. Based on this characteristic, this paper uses the k-means clustering method to cluster subtasks, and take each subtask group as the smallest unit of resource allocation, in order to reduce the size of the solution space. CGA in this work can be described, as follows:

(1) Clustering subtasks: the k-means clustering algorithm first selects the k subtask as initial centroids. Subsequently, it calculates the dependencies among subtasks, and adds a subtask with the heaviest dependency with a certain centroid into the its subtask group. Next, k-means recalculates the centroids of each subtask group and repeats the previous operations until the centroids are stable, as shown in Algorithms 2 and 3. To cluster subtask of an application, we consider divided subtasks with heavy communication dependency into a same group. By clustering $\{t_1, t_2, t_3\}$, $\{t_4, t_5, t_7\}$, each subtask cluster can be the smallest unit in resource allocation algorithm, as shown in Figure 3.

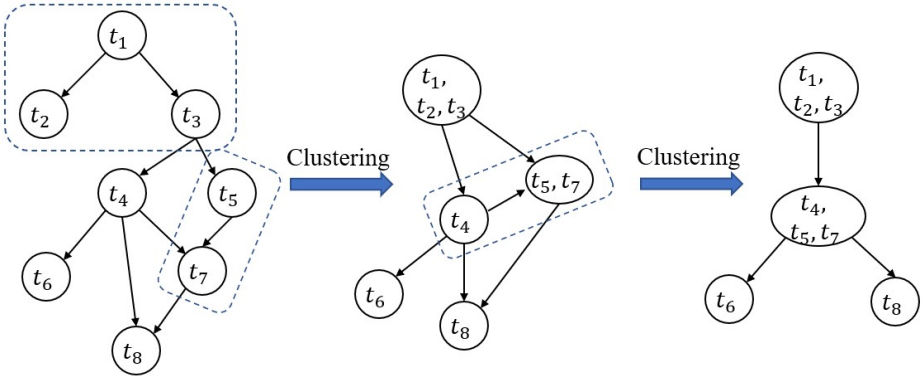

Figure 3. Clustering substaks.

By grouping the subtasks, the cost of genetic algorithm and the communication among subtasks can be reduced. However, subtask grouping makes us consider the feasibility of solution in the generation initial population, crossover, and mutation operations. Because a certain subtask in a group

can only be executed on specific server, the whole group should be scheduled to specific server, which can be described, as follows:

$$flag_x \geq \max\{layer_i\}, i \in \mathcal{G}_j, \tag{11}$$

where $\mathcal{G}_j$ denotes a subtask group and $flag_x$ denotes the server layer of solution of group $\mathcal{G}_j$.

CGA optimizes a set of initial random solutions to a acceptable solution, by evolutionary operations (selection, crossover, and mutation). The crossover and mutation operations of CGA can maintain solutions keep diversity avoid falling into local optimal trap.

(2) Chromosome and Fitness Function: in this paper, we use real coded GA. Each individual is defined by the chromosome, which describes a solution of problem (6). Figure 4 shows the chromosome structure of an individual. x_i in chromosome structure denotes a target server that is assigned to subtask cluster c_i.

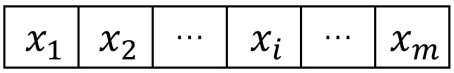

Figure 4. The chromosome structure.

To evaluate the feasibility of an individual, the fitness function is defined, as follows:

$$Fitness(X) = \max_{i \in ot}\{RT_i + T^c_{i,x_i}\}, \tag{12}$$

where ot is a set of subtask without outdegree in task DAG. It describes that the end task of an application.

(3) Initialization and Selection: as shown in Algorithm 1, first get the subtask cluster of an environmental monitoring Application a by Algorithm 2. Subsequently, generate the initial solution population randomly, but constrained by (11). The selection operation of CGA restricts partially individuals that fare better than others can survive.

Algorithm 1: Get subtasks cluster.

Input: $\mathcal{T}$
Output: subtask cluster
Initial subtask centers *center*[]<- Algorithm($\mathcal{T}$);
$c[1], c[2], ..., c[k]$; ; `// Store subtasks clustring result`
for $i = 1$ *to* k **do**
    add center[i] into $c[i]$;
while *Not converged* **do**
    for *subtask t in* $\mathcal{T}$ **do**
        Choose maximum dependency d of subtasks in *center*[];
        Cluster with this centroid;
    Select the subtask with the heaviest dependency in each group as the centroid;
return $c[1], c[2], ..., c[k]$;

Algorithm 2: Get initail subtask center.

Input: DAG with subtasks $\mathcal{T}$
Output: k centroids

```
center[];                                          // Centroid to be determined
count=0;
for subtask t in T with outdegree=0 or indegree=0 do
    Add t into center[];
if count = k then
    return center[];
if count>k then
    choose k largest degree with the heavier dependency subtasks;
    remove other count-k tasks from center[];
else
    choose k-count smallest degree with the heavier dependency subtasks;
    add these k-count tasks into center[];
return center[];
```

Algorithm 3: CGA.

Input: $\mathcal{S}$,$\mathcal{T}$
Output: resource allocation policy X

```
population size α;
elitism rate β;
mutation rate γ;
iteration times δ;
subtasks cluster = Algorithm(DAG with subtasks T)
generate α feasible solution randomly and add them to Pop;
for i = 1toδ do
    for each solution p in Pop do
        calculate fitness for p;
    new population Pop1;
    new population Pop2;
    number of elitism ne = α · β;
    select the best ne solutions in Pop and add to Pop1;
    number of crossover nc = (α − ne)/2;
    for j=1 to nc do
        select two solutions x_A and x_B from Pop;
        generate x_C and x_D by crossover from x_A and x_B;
        add x_C to Pop2, add x_D to Pop2;
    for solution x in Pop2 do
        if random(0 : 1) ≤ δ then
            generate a new solution x' randomly;
            while Algorithm3(x') = false) do
                x' = generate a new solution randomly;
            add x' into Pop2;
    Pop = Pop1 + Pop2;
return the best solution x_best in Pop;
```

(4) Crossover and Mutation: to maintain the diversity of population and reach a better solution to resource allocation problem, CGA applied crossover and mutation operations iteratively to the population. In each iteration, the selection chooses suitable individuals for crossover operation. Subsequently, the crossover operation is applied to these individual in order to generate their offsprings. Because subtasks are already clustered, in this work we only consider crossover one bit in a chromosome, in order to maintian local search capability. As shown in Figure 5, crossover operation select one bit randomly from chromosome of Parent α and Parent β, and generates offspring α and offspring β. Because the crossover operation switch the same index on chromosome that indicates a same subtask group, the solutions of offsprings generated by crossover are also feasible.

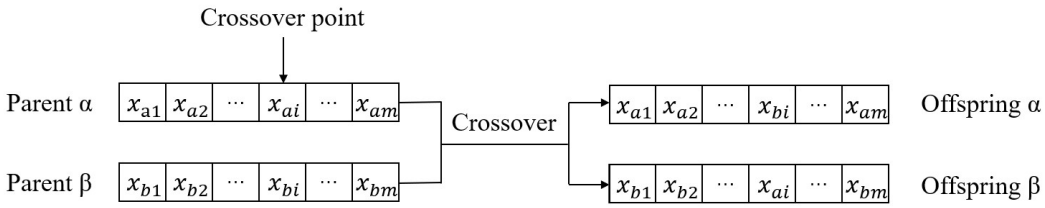

Figure 5. Crossover operation.

To avoid the population falling into local optimal trap, the mutation operation alter chromosome of individual randomly, but constrained by (11).

3.2. Task Scheduling Strategy

As aforementioned in Section 2, the ready time is an important factor of the completion time of a task. Additionally, the earliest start time of a subtask is when the predecessors of this subtask completed and transmitted to the server that this subtask assigned. Clearly, reducing the completion time of the latest predecessor task could reduce the completion time of the whole task. However, the ready time and deadline of task are ignored by traditional first come first service (FCFS) task shceduling strategy. In order to reduce the cost of ready time and take deadline into account, we propose a task scheduling strategy shown as Algorithm 4. When a subtask is completed, the server records the subtask Id into completion table that maintained on each server. Additionally, servers sort the waiting queue periodically according to the priorities of subtasks in waiting queue. The sorting operation follows that, if a subtask being executed earlier could reduce the completion time of whole task, it should be served in advance.

Subtasks t_1, t_2, t_3 and t_1^* with green describes these subtask has been executed, as shown in Figure 6. Therefore, serving t_4 ahead of time can fully use the parallel computing capacity of edge computing system to reduce the waiting time.

In addition, the proposed task scheduling strategy also tends to serve the emergency task. The strategy sort the task queue on server according to priority p of subtask, which can be calculated by Equation (13).

$$p = \alpha(t_{deadline} - t_{cur})^{-1} \cdot c, \tag{13}$$

where $t_{deadline}$ denotes the deadline of the task that depends on the type of environmental monitoring application. t_cur describes Current system time. c describes the number of completed subtasks in the set of peer subtasks, which denoted by pt in Figure 6. Peer subtasks are defined as the predecessors of successors of t_i, which can be described, as follows:

$$\forall t_j \in pt, \ \exists d^{'} = \{m_i, m_k\} \ and \ d^{''} = \{m_j, m_k\}, \tag{14}$$

where m_i and m_j denote the application modules of t_i and t_j in DAG, respectively. Because resource allocation results are carried by data packages in edge network. The task completion record can be requested quickly according to the allocation result, even if it is not maintained on the current server.

Algorithm 4: Task scheduling strategy.

Input: $q[\,]$,$\mathbb{C}$
Output: $q^{'}[\,]$
for *each subtask* t_i *in* $q[\,]$ **do**
  peer subtask $pt[\,] = pred(sucr(t_i))$;
  c = amount of completed subtask in $pt[\,]$;
  Calculate the priority p of t_i by Equation (13);
$q^{'}[\,]$ = sort $q[\,]$ by p;
return $q^{'}[\,]$;

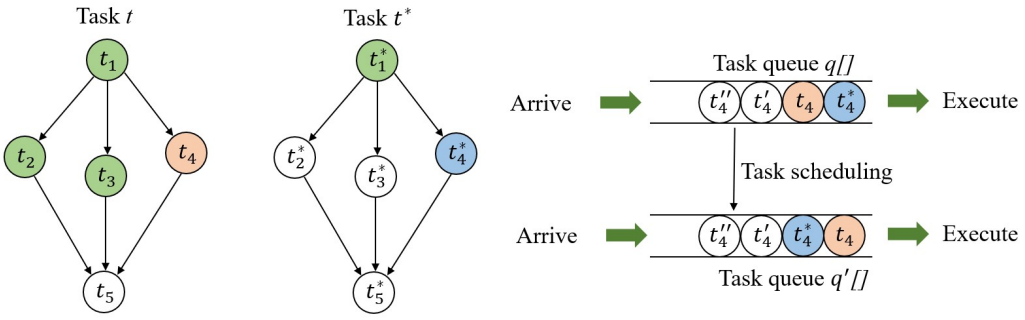

Figure 6. Task scheduling.

4. Simulation and Result

4.1. Experimental Environment

In this work, we extend iFogsim to support the architecture in Section 2. The *Tuple* class is modified to implement successors to inherit the resource allocation result and task deadline. The modified *FogDevice* class communication with other devices located on a same layer to implement collaborative computing and load balancing. The resource allocation algorithm interface is placed inside the *FogDevice*.

Servers settings: environmental monitoring sensors are installed on roadside monitoring station or vehicles, and connected to roadside edge server. The CPU capacity of each roadside edge server is in the range of 3000–6000, and the Ram range of 4000–8000. The CPU capacity, Ram, and bandwidth of region server are 8000–12,000, 20,000, and 10,000, respectively. We set the CPU capacity of cloud server as 44,000, the Ram as 40,000, and the bandwidth as 10,000. Additionally, we set two type of connection between end device (monitoring station and vehicles) and edge server: fixed number, random number. The detailed settings are shown in Table 1.

Table 1. Experimental simulation parameter settings for servers.

	CPU (MIPS)	RAM (MB)	Up/Down Bandwidth (bytes/ms)	Busy/Idle Power (W)
Cloud server	44,000	40,000	10,000/10,000	1648/1332
Region server	8000–12,000	8000	10,000/10,000	107.34/83.43
Roadside server	3000–6000	4000–8000	5000/10,000	87.43/60.43

After setting up the network architecture and devices on each layer. The application model is set, as shown in Figure 7.

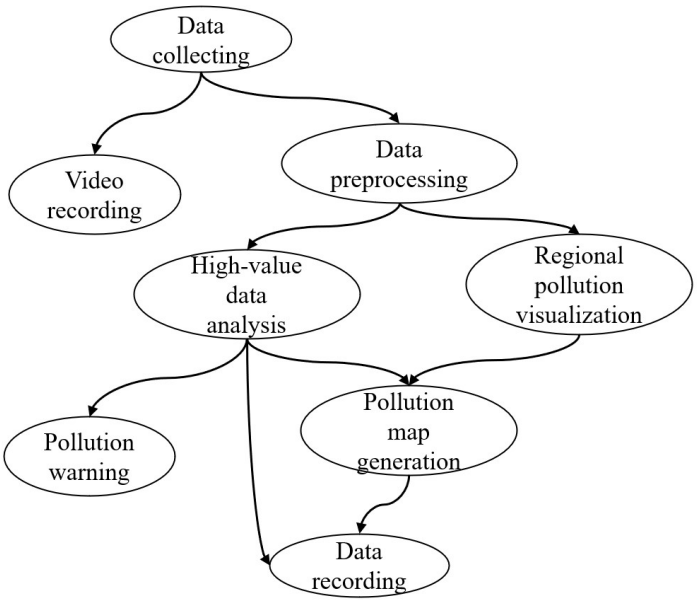

Figure 7. Environmental monitoring application modules.

The mapping relationship between application modules and servers on different layers are shown in Table 2.

Table 2. Mapping between application modules and servers.

Layer	Application Module
Cloud	Data recording Pollution map generation Regional pollution visualization High-value data analysis Vedio recording
Region	Data preprocessing High-value data analysis Regional pollution visualization Vedio recording
Roadside	Data preprocessing High-value data analysis Vedio recording
IoT Device	Data collecting Pollution warning

4.2. Simulation Result

In this section, we evaluate the performance of the proposed method in the iFogsim simulation environment. We compare our method with the basic genetic algorithm to prove the effect of our resource allocation algorithm and task scheduling strategy. In our experiment, we first observe the convergence of GA and CGA in a same time of resource allocation. Subsequently, we compare the average task completion latency with different sensor amount deployed on a roadside server, in order to ascertain a suitable number of sensor per roadside server to serve. In addition, the simulation result of average task latency with a different amount of request per sensor offload in unit time prove the minimum time granularity for environmental monitoring in this experimental environment. In the following results, GA represents basic Genetic Algorithm, CGA denotes the Combined Genetic Algorithm we proposed, and combined genetic algorithm with task scheduling strategy is denoted by CGA&Q.

4.3. Convergence Analysis

Figure 8 shows the different convergence between GA and CGA. On account of grouping subtasks, CGA converged earlier than GA in earlier iterations, but GA could reach a better sollution than CGA in later iterations. However, consider the roadside server's lack of computing resource, the iteration is set to the range 50–200 times in this work. Obviously, in this range, the CGA we proposed could reach a better sollution than GA.

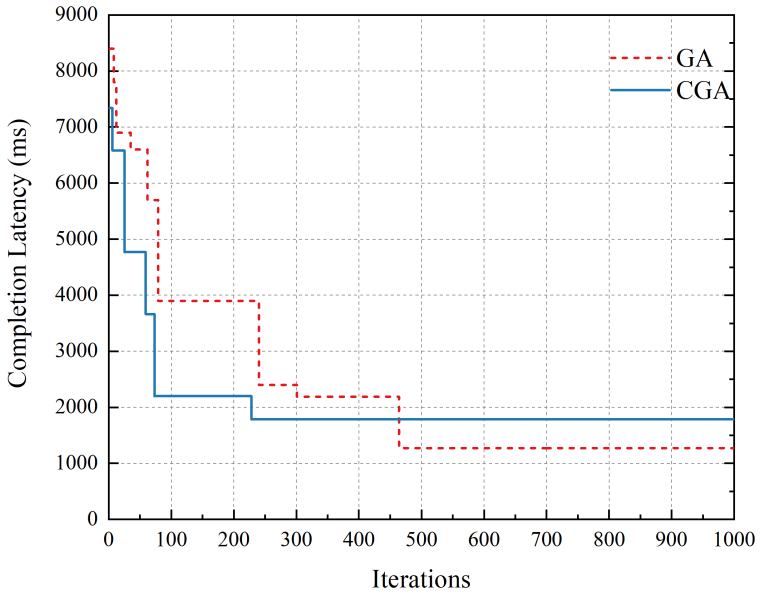

Figure 8. Convergence of GA vs. CGA.

4.4. Performance Analysis

In this section, we first show the performance of proposed algorithm versus GA with different sensor amount under a roadside server. Figure 9 plots the average task completion latency as the result in different methods. Each sensor offloads 16 environmental monitoring tasks in a time unit. When less than seven sensors are deployed on a roadside server, the effect of CGA&Q is not obvious, due to the size of task queue on each server being small. With the increase of sensor number, the performance of proposed method becomes higher. In the ten scenarios, the average completion latency optimization ratio (GA vs. CGA&Q) is 14.1%, 10.0%, 10.5%, 9.8%, 7.6%, 13.6%, 12.0%, 22.4%, 16.4%, and 13.0%. As the sensor number grows above 10, it has been hard for completion latency to meet the demand of environmental monitoring application. Therefore, in following simulation, we set the number of sensors that are deployed on roadside server to 10.

Figure 10 shows the average completion latency with different task number per sensor offloaded in a unit time. In the eight scenarios of different task amount, the average completion latency optimization ratio (GA vs. CGA&Q) is 18.4%, 16.4%, 15.5%, 18.1%, 21.6%, 21.1%, 18.7%, and 6.9%. Because each bit in the chromosome only denotes one subtask, the sollutions in the initail population of GA will cost more time on communication than CGA. When we decrease the number to 8, the performance of CGA is better than CGA&Q. Because the priority of subtask in each task queue are non-real time (behind the current system state). As task number decreases, the task queue refresh speed increases. It degrades the performance of the task scheduling strategy.

To ascertain the ratio of subtasks number executed by the different layer servers, We deploy 10 sensors on each roadside server, and set region server connected with eight roadside servers. Figure 11 summarizes the utilization of server on different level. As the amount of tasks increases, the proportion of tasks allocated to cloud computing increases from 14.06% to 31.41%. Meanwhile, the average completion latency increased from 1458 ms to 2910 ms, because more subtasks were allocated to the cloud server. Even if the less task offloaded into edge computing system, the region

server and cloud server still execute the task partially, because a certain specific subtask can only be executed on the cloud or region server.

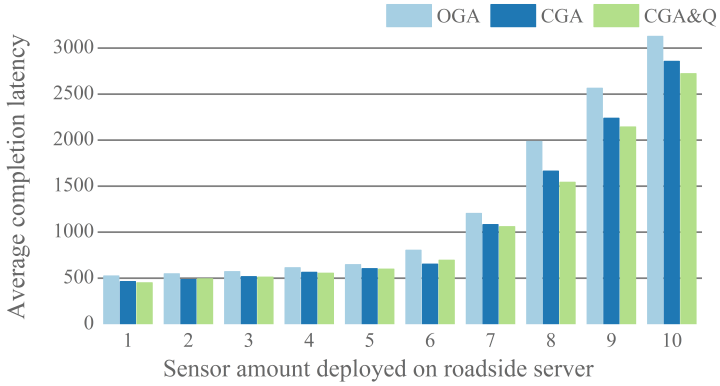

Figure 9. Impact of sensor amount on average completion latency.

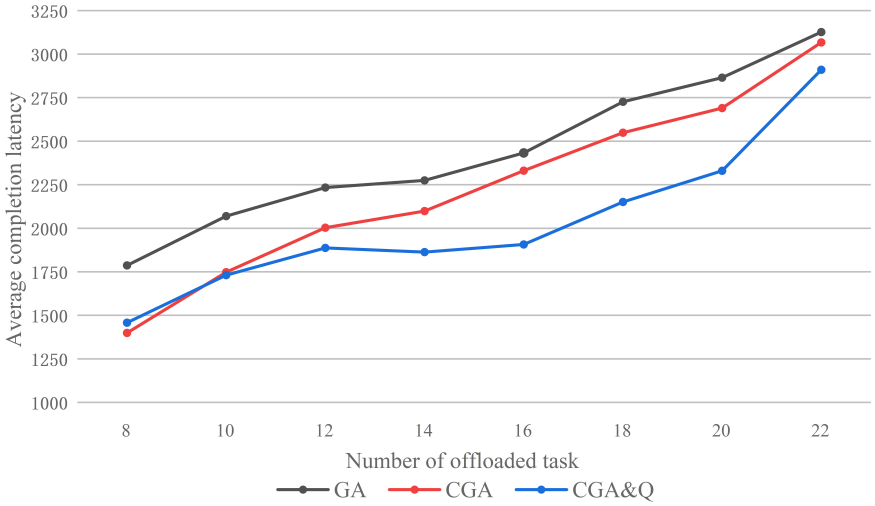

Figure 10. Average completion latency of application by GA, CGA, and CGA with task scheduling.

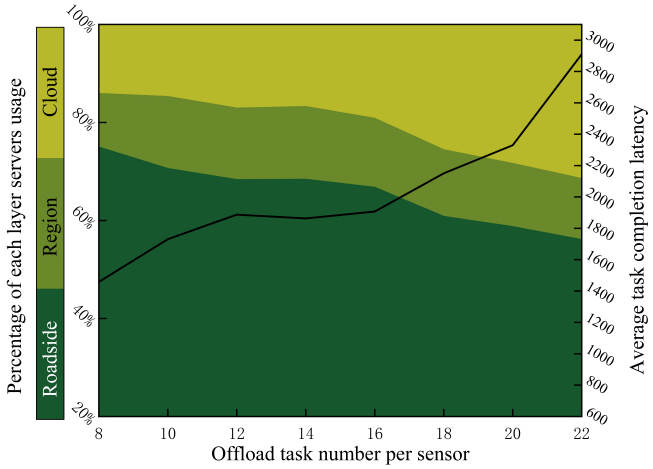

Figure 11. The utilization of servers in each layer.

5. Conclusions

This paper takes the characteristics of environmental monitoring applications into account, and presents a resource allocation algorithm and task scheduling strategy for an edge-computing based environmental monitoring system. By clustering subtasks with heavy communication dependency, the proposed method reduces the cost of calculating results of resource allocation and accelerates the convergence of the genetic algorithm in the early iterations. In addition, we propose a task scheduling strategy to fully use the parallel computing capacity of edge computing system. Subsequently, we ascertain the suitable setting of environmental monitoring system in the environment of this work. However, the clustering subtask should be dynamically adjusted to the status (i.e., real-time resources and load of edge servers) of edge computing system. Additionally, the additional communication cost (e.g., if completion information are not on current server, it will be transmitted by edge network) caused by task scheduling strategy has not been investigated. In future work, we plan to improve the proposed methods to adjust the allocation policy dynamically, and implement a real-world testbed in order to improve the proposed method and system model in terms of cost of resources and availability of monitoring service.

Author Contributions: Conceptualization, J.F.; Data curation, J.H.; Funding acquisition, J.F.; Project administration, J.F.; Resources, T.L.; Supervision, J.F.; Writing—original draft, J.H.; Writing—review & editing, J.F., J.H., J.W., T.L. and B.W. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by Beijing Natural Science Foundation, grant number 4192007.

Acknowledgments: This work is supported by Beijing Natural Science Foundation (4192007), and supported by the National Natural Science Foundation of China (61202076), along with other government sponsors. The authors would like to thank the reviewers for their efforts and for providing helpful suggestions that have led to several important improvements in our work. We would also like to thank all teachers and students in our laboratory for helpful discussions.

Conflicts of Interest: The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

Abbreviations

The following abbreviations are used in this manuscript:

$\mathcal{E}, E$	set and number of environmental monitoring sensors
$\mathcal{S}$	Set of servers
S_i	server i
$f_i, Bw_i, flag_i$	computing capability, bandwidth, and device layer of server i
A	application
m_i	application module i
$l_i, Bw_i, layer_i$	computing resource requirements, bandwidth requirement, and minimum processing layer of m_i
d_i	subtask dependency
$m_{src}, m_{dest}, data_i$	source module, destination module, and input data
$T^c_{n,m}$	execution time of subtask m on server S_n
$T^{trans}_{m,n,w}$	transmission latency of m from server n to w
RT_m	ready time of subtask m
ST_j	start time of subtask j
T_{comp}	completion time
$t_deadline$	deadline of task
X	solution of resource allocation
p	priority of subtask

References

1. Abbas, N.; Zhang, Y.; Taherkordi, A.; Skeie, T. Mobile Edge Computing: A Survey. *IEEE Internet Things J.* **2018**, *5*, 450–465. [CrossRef]
2. Capra, M.; Peloso, R.; Masera, G.; Ruo Roch, M.; Martina, M. Edge Computing: A Survey On the Hardware Requirements in the Internet of Things World. *Future Internet* **2019**, *11*, 100. [CrossRef]
3. Fang, J.; Ma, A. IoT Application Modules Placement and Dynamic Task Processing in Edge-Cloud Computing. *IEEE Internet Things J.* **2020**. [CrossRef]
4. Song, F.; Zhou, Y.; Chang, L.; Zhang, H. Modeling Space-Terrestrial Integrated Networks with Smart Collaborative Theory. *IEEE Netw.* **2019**, *33*, 51–57. [CrossRef]
5. Idrees, Z.; Zou, Z.; Zheng, L. Edge Computing Based IoT Architecture for Low Cost Air Pollution Monitoring Systems: A Comprehensive System Analysis, Design Considerations & Development. *Sensors* **2018**, *18*, 3021. [CrossRef] [PubMed]
6. Song, F.; Zhu, M.; Zhou, Y.; You, I.; Zhang, H. Smart Collaborative Tracking for Ubiquitous Power IoT in Edge-Cloud Interplay Domain. *IEEE Internet Things J.* **2020**, *7*, 6046–6055. [CrossRef]
7. Wang, C.; Liang, C.; Yu, F.R.; Chen, Q.; Tang, L. Computation Offloading and Resource Allocation in Wireless Cellular Networks With Mobile Edge Computing. *IEEE Trans. Wirel. Commun.* **2017**, *16*, 4924–4938. [CrossRef]
8. Zhang, H.; Chen, Z.; Wu, J.; Deng, Y.; Xiao, Y.; Liu, K.; Li, M. Energy-Efficient Online Resource Management and Allocation Optimization in Multi-User Multi-Task Mobile-Edge Computing Systems with Hybrid Energy Harvesting. *Sensors* **2018**, *18*, 3140. [CrossRef]
9. 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]
10. Guo, F.; Zhang, H.; Ji, H.; Li, X.; Leung, V.C.M. An Efficient Computation Offloading Management Scheme in the Densely Deployed Small Cell Networks With Mobile Edge Computing. *IEEE/ACM Trans. Netw.* **2018**, *26*, 2651–2664. [CrossRef]
11. Xu, X.; Xue, Y.; Qi, L.; Yuan, Y.; Zhang, X.; Umer, T.; Wan, S. An edge computing-enabled computation offloading method with privacy preservation for internet of connected vehicles. *Future Gener. Comput. Syst.* **2019**, *96*, 89–100. [CrossRef]
12. Guo, S.; Liu, J.; Yang, Y.; Xiao, B.; Li, Z. Energy-Efficient Dynamic Computation Offloading and Cooperative Task Scheduling in Mobile Cloud Computing. *IEEE Trans. Mob. Comput.* **2019**, *18*, 319–333. [CrossRef]
13. Song, F.; Ai, Z.; Zhou, Y.; You, I.; Choo, K.R.; Zhang, H. Smart Collaborative Automation for Receive Buffer Control in Multipath Industrial Networks. *IEEE Trans. Ind. Inform.* **2020**, *16*, 1385–1394. [CrossRef]
14. Du, J.; Zhao, L.; Feng, J.; Chu, X. Computation Offloading and Resource Allocation in Mixed Fog/Cloud Computing Systems With Min-Max Fairness Guarantee. *IEEE Trans. Commun.* **2018**, *66*, 1594–1608. [CrossRef]
15. Sahni, Y.; Cao, J.; Yang, L. Data-Aware Task Allocation for Achieving Low Latency in Collaborative Edge Computing. *IEEE Internet Things J.* **2019**, *6*, 3512–3524. [CrossRef]
16. Cui, L.; Xu, C.; Yang, S.; Huang, J.Z.; Li, J.; Wang, X.; Ming, Z.; Lu, N. Joint Optimization of Energy Consumption and Latency in Mobile Edge Computing for Internet of Things. *IEEE Internet Things J.* **2019**, *6*, 4791–4803. [CrossRef]
17. Zhang, Z.; Yu, F.R.; Fu, F.; Yan, Q.; Wang, Z. Joint Offloading and Resource Allocation in Mobile Edge Computing Systems: An Actor-Critic Approach. In Proceedings of the 2018 IEEE Global Communications Conference (GLOBECOM), Abu Dhabi, UAE, 9–13 December 2018; pp. 1–6.
18. Pereira, R.S.; Lieira, D.D.; Silva, M.A.C.d.; Pimenta, A.H.M.; da Costa, J.B.D.; Rosário, D.; Villas, L.; Meneguette, R.I. RELIABLE: Resource Allocation Mechanism for 5G Network using Mobile Edge Computing. *Sensors* **2020**, *20*, 5449. [CrossRef] [PubMed]
19. Qi, Q.; Wang, J.; Ma, Z.; Sun, H.; Cao, Y.; Zhang, L.; Liao, J. Knowledge-Driven Service Offloading Decision for Vehicular Edge Computing: A Deep Reinforcement Learning Approach. *IEEE Trans. Veh. Technol.* **2019**, *68*, 4192–4203. [CrossRef]
20. Xu, X.; Li, Y.; Huang, T.; Xue, Y.; Peng, K.; Qi, L.; Dou, W. An energy-aware computation offloading method for smart edge computing in wireless metropolitan area networks. *J. Netw. Comput. Appl.* **2019**, *133*, 75–85. [CrossRef]

21. Song, F.; Zhou, Y.T.; Wang, Y.; Zhao, T.M.; You, I.; Zhang, H.K. Smart collaborative distribution for privacy enhancement in moving target defense. *Inf. Sci.* **2019**, *479*, 593–606. [CrossRef]
22. Lin, X.; Wang, Y.; Xie, Q.; Pedram, M. Task Scheduling with Dynamic Voltage and Frequency Scaling for Energy Minimization in the Mobile Cloud Computing Environment. *IEEE Trans. Serv. Comput.* **2015**, *8*, 175–186. [CrossRef]