future internet

MDPI

Article

An Efficient Resource Scheduling Strategy for V2X Microservice Deployment in Edge Servers

Yanjun Shi *, Yijia Guo, Lingling Lv and Keshuai Zhang

School of Mechanical Engineering, Dalian University of Technology, Dalian 116024, China; gyijia@mail.dlut.edu.cn (Y.G.); lvlingling@mail.dlut.edu.cn (L.L.); zks_work@163.com (K.Z.)

* Correspondence: syj@ieee.org; Tel.: +86-411-84709130

Received: 9 September 2020; Accepted: 14 October 2020; Published: 15 October 2020

Abstract: The fast development of connected vehicles with support for various V2X (vehicle-to-everything) applications carries high demand for quality of edge services, which concerns microservice deployment and edge computing. We herein propose an efficient resource scheduling strategy to containerize microservice deployment for better performance. Firstly, we quantify three crucial factors (resource utilization, resource utilization balancing, and microservice dependencies) in resource scheduling. Then, we propose a multi-objective model to achieve equilibrium in these factors and a multiple fitness genetic algorithm (MFGA) for the balance between resource utilization, resource utilization balancing, and calling distance, where a container dynamic migration strategy in the crossover and mutation process of the algorithm is provided. The simulated results from Container-CloudSim showed the effectiveness of our MFGA.

Keywords: containerized microservice; resource scheduling; edge computing

1. Introduction

Connected and autonomous vehicle (CAV) technology involves the application of the Internet of Things (IoT) technology in intelligent transportation. CAV has become a vital component of the intelligent transportation system with the rapid deployment of 5G networks [1]. In the future, vehicle-to-everything (V2X) communications will become a reality [2], which consists of vehicle-to-vehicle (V2V), vehicle-to-infrastructure (V2I), vehicle-to-pedestrian (V2P), and vehicle-to-network (V2N). Through V2X, we can break through the technical bottlenecks of vehicle non-line-of-sight perception and information sharing. As a result, various CAV applications have also been spawned, such as platooning, advanced driving, and remote driving. However, the delay response, computing speed, and bandwidth of these application services have a high demand for service requirements. For example, the allowable delay for remote driving is 5–10 ms [3], undoubtedly increasing the communication burden. In this study, we try to propose a resource scheduling strategy to improve the utilization, which aims to meet the V2X service demand.

The combination of edge cloud computing and V2X technology has become a new solution to satisfy the above demand, as shown in Figure 1. The edge cloud is a comprehensive elastic cloud platform with computing, storage, network, and security capabilities [4]. Edge cloud computing has six advantages: low latency, self-organization, definable, schedulable, high security, and open standards. The most significant benefits are the unified coordination and service capabilities, which will effectively reduce response time, reduce bandwidth, and improve service quality when deploying the V2X applications on the edge cloud. However, the resource capacity of edge computing servers is less than the traditional cloud computing servers [5]. Thus, it is crucial to make full use of the edge cloud hardware resources within the limit of resource conditions to ensure the high reliability, flexibility, and high performance of the V2X applications.

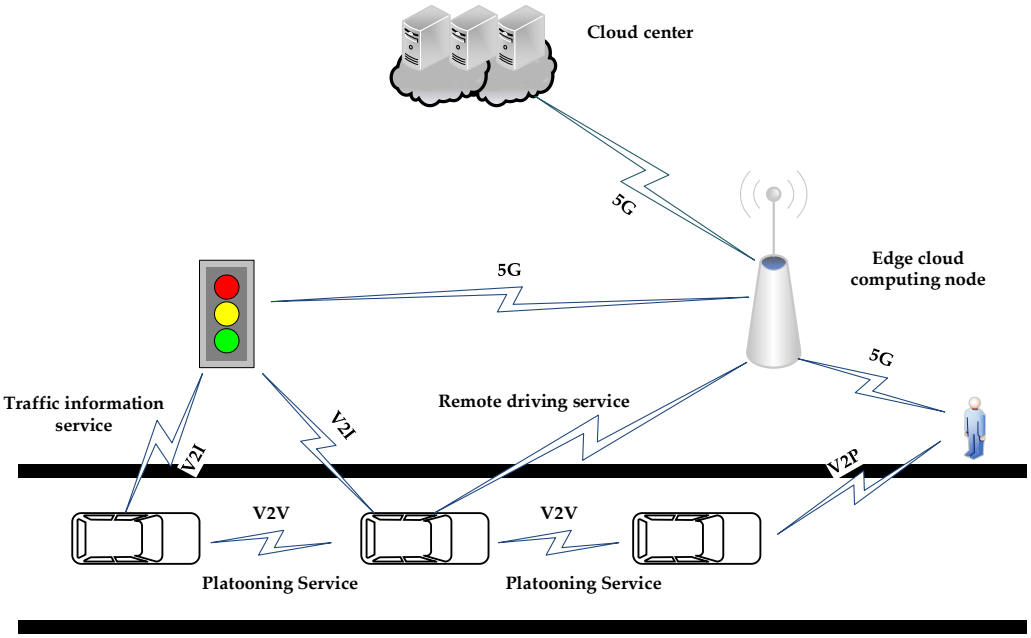

Figure 1. V2X applications deployed in edge computing.

Microservice consists of a series of software components with interdependence and specific functions, which is significant for the full use of resources [6]. Based on microservice, each application can be independently deployed, extended, and updated, which will reduce the development cycle and increase flexibility. However, microservices were deployed directly on bare metal before the advent of container technology, which was challenging to operate and maintain [7]. Moreover, the container is a lightweight virtualization technology [8], whose engine layer provides resource scheduling and component isolation for microservices, as shown in Figure 2. Containers directly use hardware resources in the physical host, which can meet different platforms' requirements regarding development, test, and production environments [9]. However, the granularity of containers is smaller, and the number of physical machines that can be operated is higher, which means that resource management for containers is more complicated. In this study, we will develop a resource scheduling strategy to improve the performance of the container-based microservice.

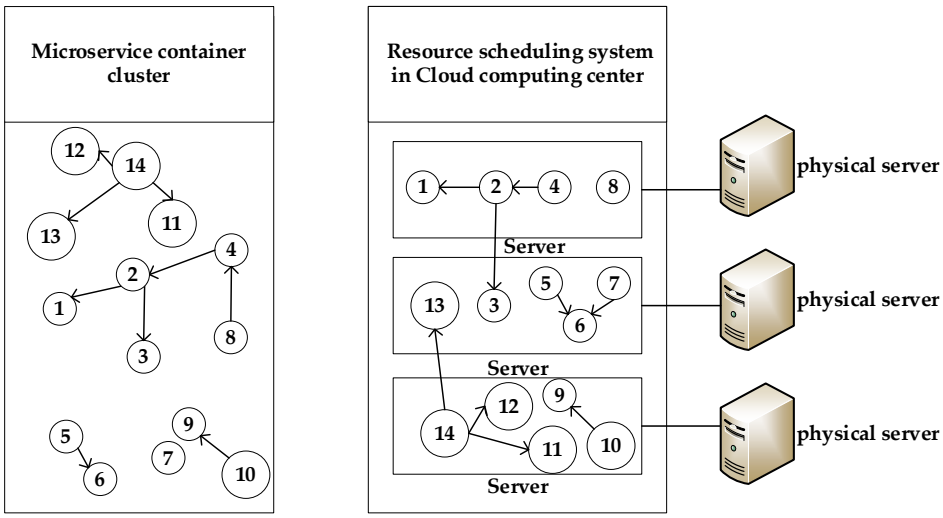

Figure 2. Schematic diagram of cloud microservice resource scheduling.

At present, standard container orchestration tools such as Kubernetes and Docker SwarmKit can only provide simple resource scheduling strategies [10] that cannot utilize physical machine resources. Thus, some studies proposed strategies for better resource utilization. For example, Liu [11] proposed an edge computing architecture that utilized container-based microservice to ensure the flexible deployment of applications in 2016. Xu et al. [12] proposed a cloud center infrastructure which was based on the container virtualization technology. They also put forward a resource scheduling method of container virtual cloud environments to reduce customer operation time and improve supplier resource utilization. Filip et al. [13] proposed a new microservice scheduling model in a heterogeneous cloud edge environment, whose novelty was the microservices-oriented method. Kaewkasi et al. [14] proposed a container scheduling strategy based on the ant colony algorithm to solve the shortcomings of the Docker SwarmKit orchestration engine's scheduling strategy. The above strategy can improve the utilization to a certain extent, but it lacks the consideration of the multi-objective in the scheduling process.

There were some studies considering the multi-objective problem. For example, Kaur et al. [15] designed a multi-objective function using a game theory model to reduce energy consumption and maximum task completion time by considering different constraints such as memory, CPU, and user budget. Meanwhile, they used lightweight containers instead of traditional virtual machines to reduce the fog device's overhead, response time and overall energy consumption. Liu et al. [16] proposed a fuzzy service offload decision mechanism (FSODM) algorithm based on the characteristics of limited resources under edge computing and based on the micro-scale deployment architecture to achieve rapid deployment and dynamic expansion of microservice containers. Hani Sami and Azzam Mourad [17] proposed an evolutionary memetic algorithm to solve the multi-objective container optimization problem in a dynamic fog environment, whose goals were Quality of Service (QoS) maximization, survivability factor maximization, and active host minimization. Lin et al. [18] established a multi-objective optimization model for container-based microservice scheduling, which took computing, the storage resources of the physical nodes, the number of microservice requests, and the failure rate of the physical nodes into consideration. The current proposed multi-objective studies lacked the consideration of microservices' independence, which made the improvement of resource utilization and other goals limited.

Some studies considering mobile agents have been proposed. For example, Shi et al. [19] used a tail matching subsequence (TMSS) based mobile access prediction method which exploited the inherent user mobility patterns to solve the intermittent connectivity. Meanwhile, the authors developed an integer encoding-based adaptive genetic algorithm (GA) to make offloading decisions, which considered the cloudlet reliability, computation requirements, and mobile device energy consumption. Avgeris et al. [20] established the vertical and horizontal scaling mechanism to satisfy the QoS requirements. The vertical scaling mechanism used the linear switching system and state feedback controller to develop a dynamic resource allocation and admission control mechanism. The horizontal scaling mechanism was in charge of each edge server's activation/deactivation, the placement of the applications' instances, and the distribution of the incoming offloaded requests.

In this study, we try to address several challenges in resource scheduling for V2X microservice deployment. Previous studies did not consider the microservice dependencies, which led to wasted resources. In our study, we try to use the calling distance to quantify the dependence. Then, we model a multi-objective model considering resource utilization, resource balancing, and calling distance. We propose a multiple fitness genetic algorithm to balance the above factors, where a container dynamic migration strategy is introduced in the crossover and mutation operation.

The rest of this study is organized as follows. The resource scheduling model in containerized microservice is discussed in Section 2. The resource scheduling algorithm is proposed in Section 3. The simulation experiments will be discussed in Section 4. Finally, we conclude this study in Section 5.

2. The Resource Scheduling Model in Containerized Microservice

We establish a multi-objective resource scheduling model based on the edge container cloud architecture for V2X microservices in this article. This model aims to achieve a balance between cross-server calls, server resource utilization, and load balancing. Simultaneously, our model considers the dependencies and hardware constraints between containers, thereby ensuring the response speed, which improves the cloud center's overall performance.

The existing M microservice containers need to be mapped to N physical hosts. Because small cloud computing centers such as the edge cloud have specific service scenarios, the physical server types they deploy are generally uniform, which means that their resource types and capabilities are set to the same. Therefore, the resource scheduling model can be further described as follows: M containers are divided into H groups, and each group of containers corresponds to a physical host. If H is not equal to N, activate or deactivate the physical host. The goal is to minimize the number of calls between containers across physical hosts, to reduce the number of physical hosts occupied by microservice containers, and to ensure load balancing of activated physical clusters as much as possible. Ultimately, we achieve the best overall performance of microservice deployment.

2.1. Microservice Resource Constraints

In the beginning, we use P as the set of physical hosts, where $P = \{P_1, P_2, \ldots, P_n\}$. n is the number of physical hosts. $P_i = \{0, 1\}$ is a binary variable. When it is 1, it indicates that the physical host i is activated. Otherwise, it shows that the physical host i is in the sleep state. When deploying microservices, we firstly consider that each container can only be deployed on one physical host, as shown in Equations (1) and (2). The binary parameter, k_{ij}, indicates that the container j will be deployed on the host i, where the amount of containers is m.

$$\sum_{i=1}^{n} k_{i,j} = 1 \tag{1}$$

$$k_{ij} = \begin{cases} 1, \text{ the container } j \text{ is placed on the physical host } i \\ 0, \text{ else} \end{cases} \tag{2}$$

At the same time, there is also a comprehensive resource constraint, which means that the sum of the resources required by all microservice containers on the same host i cannot exceed the physical host's resources, as shown in Equation (3).

$$\sum_{j=i}^{m} k_{ij} \times r_{j,l} \leq c_{i,l} \quad 1 \leq l \leq \mathrm{s} \tag{3}$$

In Equation (3), the set of all resources requested by the container is $R_j = \left\{r_{j1}, r_{j2}, \ldots, r_{js}\right\}$, where the represents the type l resource requested by the container j. In this article, we use C_{is} to indicate the set of the type s resources that the physical host i can provide. In this article, we only consider four types of resources, namely CPU, memory, disk, and bandwidth, so the maximum of s is 4.

2.2. Microservice Container Dependencies

The microservice architecture's characteristic is to differentiate an extensive service project into individual modules for a single service. There exists a calling relationship between these modules, which creates a dependency relationship between the deployed containers. Therefore, the dependency relationship between containers is a factor that must be considered in the resource scheduling process for containers. This means that containers with call dependencies are allocated to the same physical host as much as possible, which can reduce the time for containers to call across physical hosts, minimize the waste of network resources, and improve the QoS.

In this article, the calling distance, $v\left(k_i, k_j\right)$, is used to quantify the dependence between the container i and j, as shown in Equation (4).

$$v\left(k_i, k_j\right) = \begin{cases} 0, \text{ no direct calling relationship between container } i \text{ and } j \\ 1, else \end{cases} \tag{4}$$

Through $v\left(k_i, k_j\right)$, we can calculate the sum of the dependency relationship between the container i in the host t and other used containers, as shown in Equation (5). When the i container in the physical host t has a stronger dependence on different containers, this may indicate that the i container is more suitable for deployment on the physical host t.

$$v_{ti} = \sum_{j=1}^{w} v\left(k_i, k_j\right) \tag{5}$$

Finally, we can calculate the dependency correlation between all containers in the physical machine t, as shown in Equation (6). When the value of v_t increases, the total dependency of all containers in the same physical host t will be higher. At the same time, this also means that the number of network calls across physical hosts generated in the system is less, and the response time of client requests is shorter.

$$V_t = \sum_{i=1}^{w} v_{ti} \tag{6}$$

2.3. Critical Factors in Resource Scheduling

We propose three goals for this model: the shortest calling distance, the highest physical cluster resource utilization rate, and the highest load balancing after analyzing the resource constraints and microservices' dependencies. In the following chapters, we will quantify the three goals and give the quantification equation.

2.3.1. Microservice Calling Distance

When a user sends a service request, a call chain will be formed on the server. All microservices in the call chain will work together to meet the user needs. Therefore, it is necessary to call the containers deployed in each host. The time taken by container calls in the same physical host will be significantly shorter than those across physical hosts. Therefore, in the process of container deployment and resource scheduling, the cross-physical host calls of the container should be as few as possible. In this way, it is possible to reduce the time for containers to transfer network calls between physical hosts and reduce network resource waste. We define Equation (7) to calculate the calling distance between containers to quantify the calls between containers.

$$\begin{cases} d\left(k_i, k_j\right) = \begin{cases} 1, \text{ calls across physical hosts between containers } i \text{ and } j \\ 0, else \end{cases} \\ D_{ij} = \sum_{j=1}^{m-1} d\left(k_i, k_j\right) \\ D_i = \mathrm{D}_{ij} + D_{ji} \\ \mathrm{D} = \sum_{i=1}^{m} D_i \end{cases} \tag{7}$$

$d\left(k_i, k_j\right)$ can quantify the calling relationship between containers i and j. In Equation (7), we can use D_{ij} to calculate the sum of the calling distances of all containers starting from a particular container. Then, we use D_i to calculate the sum of the calling distances between all containers that have a dependency on container i. Finally, we can obtain the sum of the calling distance of all containers

through D. The smaller the value of D, the faster the microservice response month, and the higher the network resource utilization.

2.3.2. Resource Utilization

Because the computing resources in the edge cloud are relatively small compared to traditional data centers, the energy consumption of the physical host when it is not loaded accounts for around 70% of the energy consumption when it is fully charged. Therefore, it is of considerable significance to the edge cloud data center to effectively use the computing resources and reduce the energy consumption in the data center. When resource scheduling is performed, one of the optimization goals is to occupy the lowest number of physical hosts under the premise of meeting their needs, so that the resources of the physical hosts are effectively used.

We define the parameter Z to represent the total number of physical hosts occupied by deploying containers with microservices, as shown in Equation (8). In the entire edge cloud, the overall resource utilization rate of the physical host group activated for the deployment of microservices is represented by the parameter U, as shown in Equation (9).

$$Z = \sum_{i=1}^{n} P_i \tag{8}$$

$$U = \left(\frac{\sum_{i=1}^{n} \sum_{j}^{m} \sum_{l}^{s} p_i \times k_{ij} \times r_{jl}}{\sum_{i}^{n} \sum_{l}^{s} p_i \times c_{il}} \right) \tag{9}$$

In the edge cloud computing center, microservices have different requirements for different types of resources, so it is easy to cause a load of various types of resources in the physical host to be unbalanced, thereby causing a waste of server resources. To effectively use various resources, an optimization goal is to make the physical server achieve a better resource balance as much as possible. This study defines the quantitative calculation method of resource utilization, as shown in Equation (10), where u_{il} represents the utilization of type l resources on host i. The parameter u_i represents the average utilization rate of all resources on the physical host i.

$$\begin{cases} u_{il} = \frac{\sum_{i=1}^{m} k_{ij} \times r_{jl}}{c_{il}} \\ u_i = \frac{1}{s} \sum_{l=1}^{s} u_{il} \end{cases} \tag{10}$$

2.3.3. Resource Utilization Imbalance of Server

At the same time, we can also calculate the resource utilization imbalance of server i, according to Ni, as shown in Equation (11). Finally, we can use N to calculate the resource utilization imbalance of the entire edge cloud. The smaller the values of Ni and N, the more balanced the corresponding load.

$$\begin{cases} N_i = \sqrt{\frac{1}{s} \sum_{i=1}^{s} (u_{i,l} - u_i)^2} \\ N = \sqrt{\frac{1}{z} \sum_{i=1}^{n} p_i \times \sum_{l=1}^{s} \left(u_{i,l} - u_i \right)} \end{cases} \tag{11}$$

2.4. Target in Resource Scheduling

After the above quantitative calculations and analysis of the constraints, we can obtain the scheduling model of the edge cloud microservice container. Based on this model, we propose the research goals of this study: the minimum calling distance of microservice containers across physical

machines, high resource utilization of edge cloud physical machine clusters, and load balancing. The multi-objective function used in this article is shown in Equation (12).

$$\max Aim(aim_1, aim_2, aim_3) = \max\left\{\frac{1}{D}, \frac{U}{Z}, \frac{1}{N}\right\} \tag{12}$$

In this formula, D, U, Z, N have the same meaning as described above. In this article, we have defined the physical machine load evaluation function $Evaluate_i$ as shown in Equation (13) to calculate the load performance of each physical host. We quantify the physical host's load performance with three parameters: the physical host's resource utilization, load balancing, and the degree of dependence of the microservice container deployed on it.

$$Evaluate_i e_1, e_2, e_3 = \left\{V_i, u_i, \frac{1}{N_i}\right\} \tag{13}$$

3. The Resource Scheduling Algorithm

To solve the above model, we design a gene evaluation function based on the classic genetic algorithm, introducing the container dynamic migration strategy into the crossover and mutation processes. Finally, we propose a new multi-fitness genetic algorithm (MFGA) that combines resource utilization, load balancing, and microservice dependencies to solve the resource scheduling problem.

3.1. Chromosome Coding

Similarly to evolution in nature, chromosome coding is a vital part of ensuring individual performance. According to the problem analysis, the real number coding method is more suitable to solving this problem [21]. Therefore, according to this method, we code and assign the initial solution according to the process shown in Figure 3.

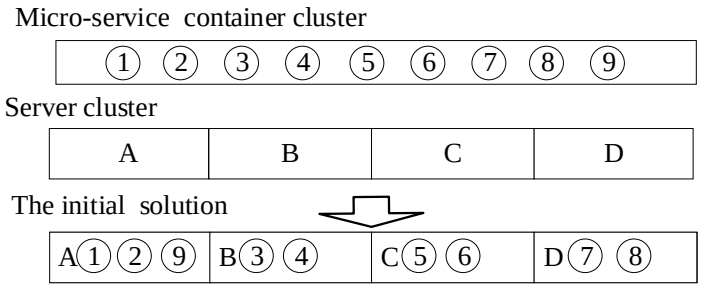

Figure 3. Solving the initial feasible solution.

As shown in Figure 3, the numbers represent the microservice container cluster to be allocated. The letters represent the cloud computing center's physical hosts cluster. The correspondence between the numbers and the letters represents the deployment of different microservice containers to the designated physical host. First, the system randomly generates an array of integers from 1 to M that does not repeat. According to the resource constraints defined by Equations (1) and (3), the first adaptation algorithm is used to divide the containers into H groups, each group corresponding to a physical host. After the above steps, an individual chromosome code can be formed, which means an initial feasible solution to the problem model.

3.2. Fitness Function

In the genetic algorithm, the fitness function is used to evaluate the quality of the individuals in the population. A small fitness value indicates that the solution to the problem is weak, and vice

versa. According to the previous chapter's model, we propose fitness function combined with the goals, as Equation (14) shows.

$$\begin{cases} Fitness f_{resptime}, f_{resutlization}, f_{lodabalance} = 1000\left(\alpha\frac{1}{D} + \beta\left(\frac{U}{Z}\right) + \lambda\frac{1}{1000N}\right) \\ \alpha + \beta + \lambda = 1 \end{cases} \tag{14}$$

In Equation (14), α, β, and λ are weight parameters whose value ranges are (0, 1), representing the proportion of container called distance, resource utilization, and load balancing in the function. The larger the parameter value, the higher the impact of the corresponding goal on the algorithm objective function value. The administrator determines the initial cost according to the actual situation.

3.3. Gene Evaluation Function

By introducing the gene evaluation function g_i, the fitness value of each gene in an individual can be evaluated. According to the problem model established in the previous chapter, each gene represents the load status of a physical host. Therefore, based on the main parameters that affect the load performance of the physical machine, we establish a gene evaluation function, as shown in Equation (15). This equation can improve each machine's performance and accelerate the algorithm's convergence when solving the multi-objective optimization model of microservice resource scheduling.

$$g_i = \alpha V_i + \beta(10u_i) + \lambda\frac{1}{N_i} \tag{15}$$

The parameters in Equation (15), α, β, and λ, are the same as those in Equation (14). The higher the resource utilization on the physical host, the more balanced the load and the more influential the dependency of the microservice containers deployed. Equation (15) can be used to evaluate the fitness value of genes and accelerate the convergence of the algorithm.

3.4. Crossover Operation

In this section, the crossover operator is redesigned. The gene evaluation function is used to achieve the crossover effectiveness of each task set in the offspring while ensuring a feasible solution, thereby speeding up the convergence of the algorithm. When performing crossover, first select the initial solutions T1 and T2 of the initial population and randomly generate a random number between 0 and 1. If the random number is less than the crossover probability pc, the crossover operation is performed. If it is higher than the crossover probability pc, the two parents enter into the next iteration. Then, it completes the procedure according to the selection of three processes: swap, delete, and re-add. The selection exchange process mainly completes the exchange of the most adaptable genes, as shown in Figure 4a. The function of the deletion process is to delete the same microservice container in the obtained new chromosome, as shown in Figure 4b. Finally, in the addition process, containers that are missing due to gene exchange are sorted according to the order in which they were initially placed in the physical host to obtain the crossed offspring, as shown in Figure 4c.

3.5. Mutation Operation

The mutation process is for local fine-tuning so that individuals cover the optimal global solution [22]. Therefore, the mutation operator used in the algorithm calculates each gene's fitness value according to Formula (14) and finds the gene with the smallest fitness value as the mutation object. The physical host corresponding to the gene with the lowest fitness value is canceled. The containers on it are redistributed according to the first adaptation algorithm, as shown in Figure 5. Finally, the container migration is completed.

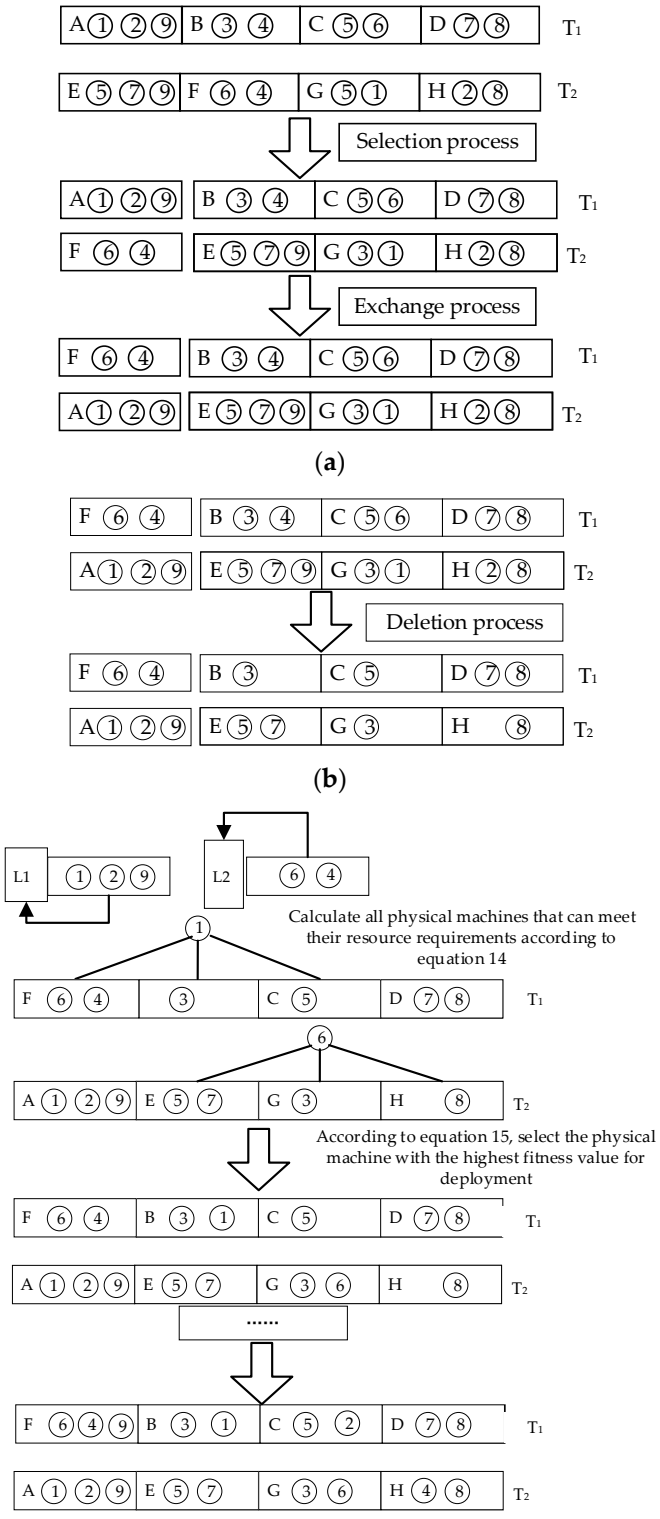

Figure 4. The process of crossover operator: (**a**) the selection process in crossover operation; (**b**) the deletion process in crossover operation; (**c**) the re-add process in crossover operation.

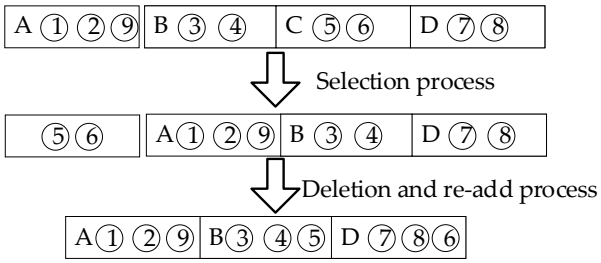

Figure 5. The process of the mutation operation.

3.6. Algorithm Flow

Steps of microservice resource scheduling based on the MFGA are shown in Figure 6.

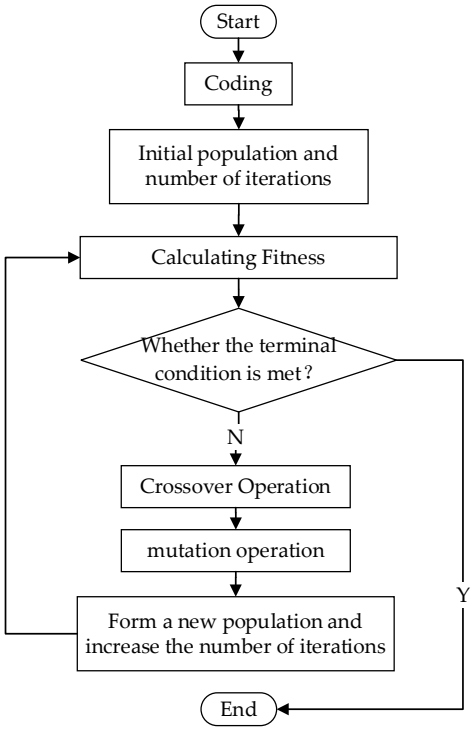

Figure 6. The steps of microservice resource scheduling based on the MFGA.

(1) Coding: Coding the problem based on the above model.

(2) Population initialization: Initialize the population using the first adaptation algorithm to obtain the initial feasible solution.

(3) Finding the fitness value: Calculate all individuals' fitness values according to the resource scheduling model and fitness function.

(4) Crossover operation: Based on the crossover probability of the algorithm, select crossover individuals and perform crossover operations based on the gene evaluation function to complete the update of the individual structure.

(5) Mutation operation: Select mutation individuals based on the mutation probability of the algorithm and perform mutation operations based on the gene evaluation function to obtain the mutated population.

(6) Selection process: Through the selection operator designed by the algorithm in this study, individuals who enter the next generation are selected to obtain a new population. The fitness value of individuals in the new community is calculated.

(7) If the number of iterations reaches the set value (350), the program will be terminated; otherwise, return to Step (4) and continue the cycle.

4. Simulation Experiments and Analyses

We use Container-CloudSim, whose simulation flows are shown in Figure 7, which is the container cloud evaluation environment in CloudSim [13], to simulate the designed microservice resource scheduling optimization model and algorithm. In order to compare the effectiveness of the algorithm, we select the round-robin algorithm (RR) [23], the most-utilization first algorithm (MF) [24], and the first come first serve (FCFS) algorithm in ContainerCloudSim-4.0 for comparison.

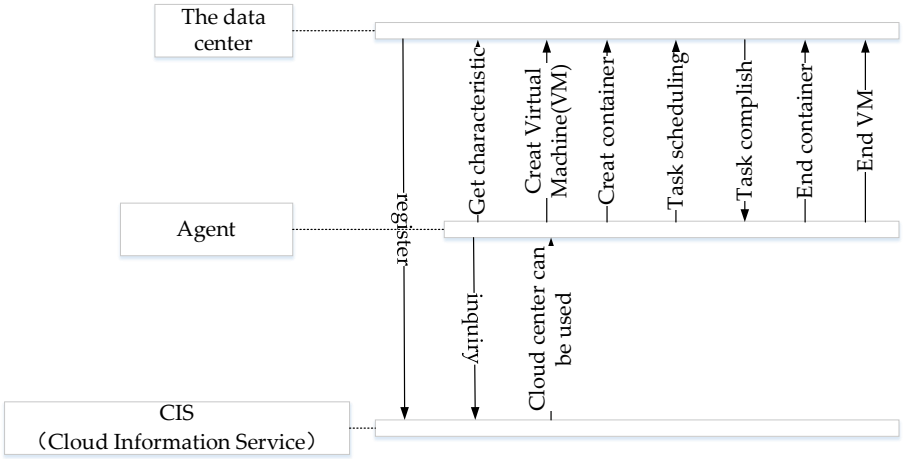

Figure 7. The flow of Container-CloudSim simulation.

The experiments were run on Intel® Core™ i5-4200U CPU@2.30 GHz and 8G RAM computer. To simulate servers' situation in edge computing, we set all servers with the same performance, as shown in Table 1. As stated in the previous chapter, we mainly consider four types of resource utilization: CPU, memory, disk, and bandwidth.

Table 1. Physical server configuration information.

Configuration	Core of CPU	MIPS	RAM (MB)	ROM (MB)	Bandwidth (MBps)
server	4	3000	4096	1,000,000	10,000

We set the microservice container's size to be tenth order, hundredth order, and thousandth order to better simulate the deployment scenario of the V2X application deployment, as shown in Table 2. The characteristics and calling dependencies of containers are created by random functions. These three orders of magnitude can correspond to small-scale, medium-scale, and large-scale deployment scenarios of V2X applications.

Table 2. Container and physical machine scale configuration information.

Scenario	Number of Containers	Number of Hosts
1	70	7
2	280	28
3	1120	112

We found that when $\alpha = 0.3$, $\beta = 0.2$, and $\lambda = 0.5$, the simulation could achieve the best results through multiple sets of experiments. Compared with the other three algorithms, MFGA consumed the fewest physical hosts when the amount of microservice containers reached 280 or more in Figure 8.

Significantly, the multiple fitness genetic algorithm (MAGA) used 20% fewer hosts than the FCFS and RR algorithms, when the size of containers reached the thousandth order.

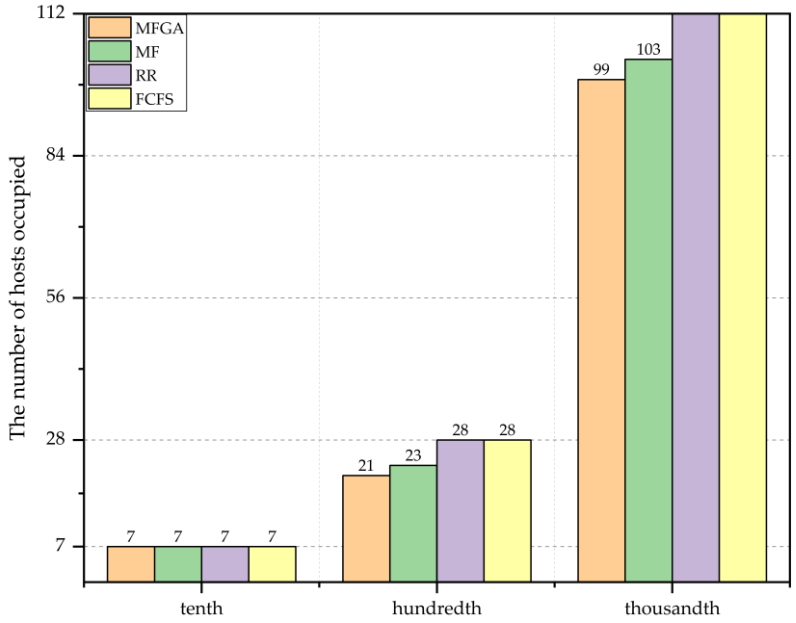

Figure 8. Comparison of the number of hosts occupied compared by four algorithms.

Figure 9 shows the simulation result of the resource utilization rate of the data center. Similarly, the advantages of MFGA become more and more significant in resource utilization with the increase in the microservice containers. The MFGA algorithm keeps the resource utilization of 0.8 in the hundredth or thousandth order scenario, which is superior to all the other algorithms.

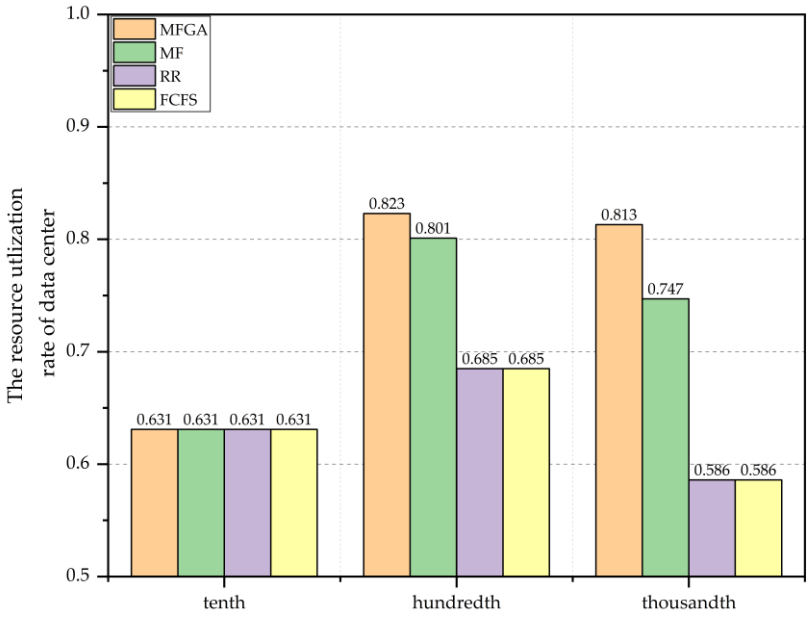

Figure 9. Comparison of the resource utilization rate of data center compared with four algorithms.

Figure 10 shows the simulation results of four algorithms in terms of calling distance. The MFGA has a smaller calling distance, which means that MFGA needs less cross-server scheduling when deploying containers. Although the number of containers changed drastically, the growth curve of the MFGA was not noticeable compared to the other three algorithms. In other words, the MFGA could

perform better in the scenario of a large number of containers due to the consideration of the container's dependencies, which was an important reason that MFGA performed better in the resource utilization.

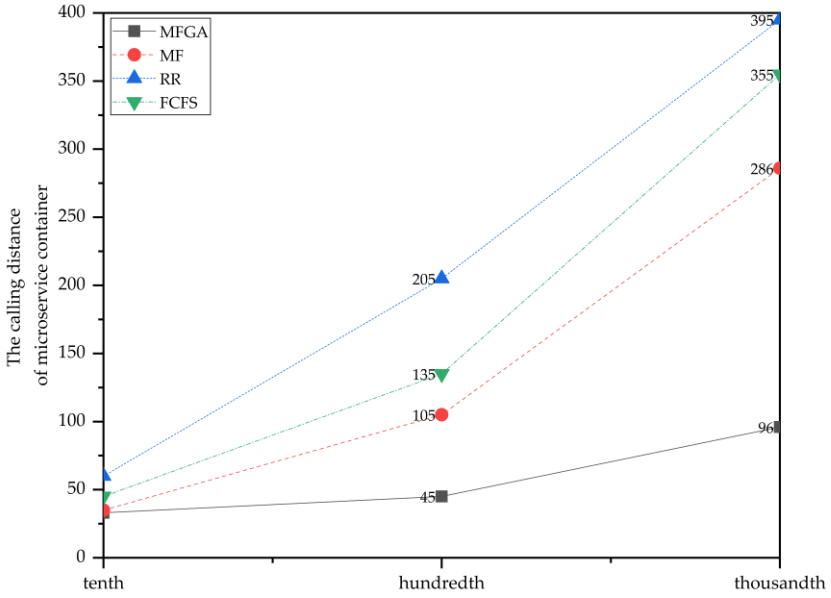

Figure 10. Comparison of microservice calling distance compared with four algorithms.

Figure 11 shows the comparison of four algorithms in terms of the resource utilization imbalance. The resource utilization imbalance increased as the number of containers grew. The MFGA maintained a relatively low level in resource utilization imbalance compared to other algorithms, which allowed it to achieve an ideal resource balance effect.

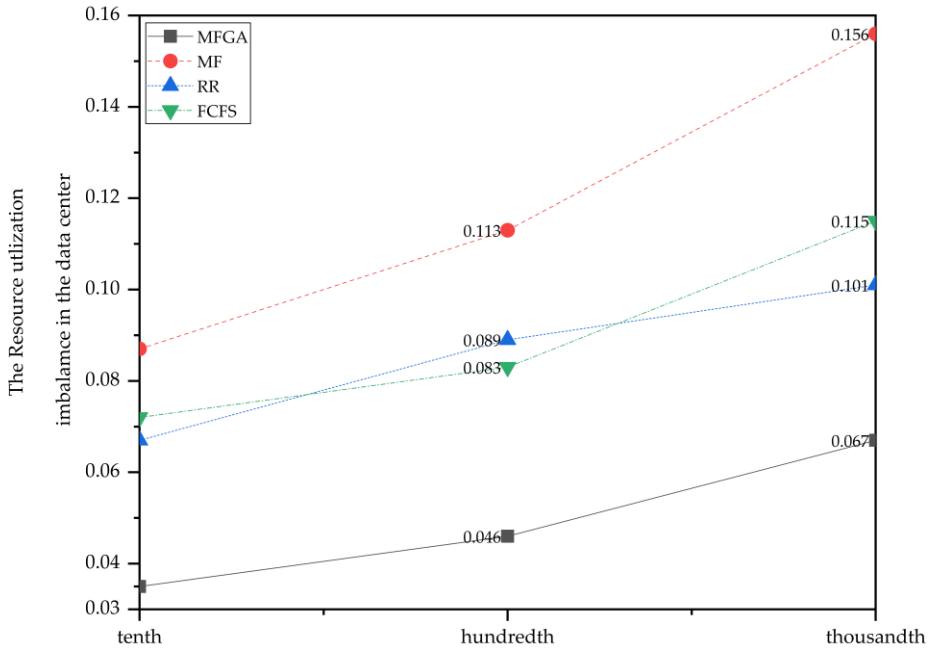

Figure 11. Comparison of resource utilization imbalance compared with four algorithms.

In all, we have achieved our optimization goals: fewest hosts occupied, most resource utilization, lowest calling distance, and resource utilization imbalance. In particular, the MFGA performed well in the thousandth order scenario, which is similar to the real situation of deploying V2X containers. Thus, MGFA is an efficient choice to solve resource scheduling problems.

5. Discussion

In this article, we study the resource scheduling in containerized microservice deployment. To model this problem, we analyze its architecture and quantify the key factors in the scheduling process. After obtaining a multi-objective model, we develop the multiple fitness genetic algorithm, which introduces a container dynamic migration strategy into the process.

To evaluate the efficiency of our model and MGFA, we use Container-CloudSim to simulate the different scenarios in V2X microservice deployment. During the simulation experiments, we find that MGFA performs better than the RR algorithm, the MF algorithm, and the FCFS algorithm, especially when we compare results in terms of the number of hosts occupied, the utilization rate of the data center, resource utilization imbalance, and calling distance.

The method above improves the calculation efficiency of MGFA and makes it more suitable for solving microservice resource scheduling problems. However, we did not consider the situation of deploying the vehicle network application of collaboration with the central cloud. At the same time, the weights in our algorithm are decided by the administrator, which may cause faults in the scheduling process. Last but not least, we only simulate some scenarios in Container-CloudSim, and we do not evaluate our algorithm in an actual situation.

We will develop a weight adaptive adjustment system to change weights based on the different characteristics in edge computing architecture in future work. Moreover, we will design a cloud-side collaborative microservice scheduling strategy to better suit the actual situation. Finally, we will deploy our developed system in a real V2X network to test its performance.

Author Contributions: Conceptualization, Y.S. and L.L.; funding acquisition, Y.S.; investigation, Y.G.; project administration, Y.S.; resources, K.Z.; software, K.Z.; writing—original draft, Y.G.; writing—review and editing, L.L. All authors have read and agreed to the published version of the manuscript.

Funding: This work was funded by China National Key Research and Development Program (NO. 2018YFE0197700).

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

References

1. Li, G.; Sun, C.; Zhang, J.; Jorswieck, E.; Xiao, B.; Hu, A. Physical Layer Key Generation in 5G and Beyond Wireless Communications: Challenges and Opportunities. *Entropy* **2019**, *21*, 497. [CrossRef]
2. Condoluci, M.; Gallo, L.; Mussot, L.; Kousaridas, A.; Spapis, P.; Mahlouji, M.; Mahmoodi, T. 5G V2X System-Level Architecture of 5GCAR Project. *Future Internet* **2019**, *11*, 217. [CrossRef]
3. Emara, M.; Filippou, M.C.; Sabella, D. MEC-assisted End-to-End Latency Evaluations for C-V2X Communications. In *European Conference on Networks and Communications*; IEEE: Piscataway, NJ, USA, 2018; pp. 157–161.
4. Zhang, H.; Wang, Z.; Liu, K. V2X Offloading and Resource Allocation in SDN-Assisted MEC-Based Vehicular Networks. *China Commun.* **2020**, *17*, 266–283. [CrossRef]
5. Zhou, H.; Xu, W.; Chen, J.; Wang, W. Evolutionary V2X Technologies toward the Internet of Vehicles: Challenges and Opportunities. *Proc. IEEE* **2020**, *108*, 308–323. [CrossRef]
6. Neatu, D.F.; Stochitoiu, R.D.; Postoaca, A.V.; Filip, I.D.; Pop, F. My Cloudy Time Machine: A scalable microservice-based platform for data processing in Cloud-Edge systems A proof of concept for the ROBIN-Cloud project. In Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing, Limassol, Cyprus, 8–12 April 2019; pp. 1451–1458.
7. Shih, Y.; Lin, H.; Pang, A.; Chuang, C.; Chou, C. An NFV-Based Service Framework for IoT Applications in Edge Computing Environments. *IEEE Trans. Netw. Serv. Manag.* **2019**, *16*, 1419–1434. [CrossRef]
8. Buzachis, A.; Galletta, A.; Carnevale, L.; Celesti, A.; Fazio, M.; Villari, M. Towards Osmotic Computing: Analyzing Overlay Network Solutions to Optimize the Deployment of Container-Based Microservices in Fog, Edge and IoT Environments. In Proceedings of the IEEE 2nd International Conference on Fog and Edge Computing (ICFEC), Washington, DC, USA, 1–3 May 2018.
9. Stefanic, P.; Stankovski, V. Multi-Criteria Decision-Making Approach for Container-based Cloud Applications: The SWITCH and ENTICE Workbenches. *Teh. Vjesn.* **2020**, *27*, 1006–1013. [CrossRef]

10. Javed, A.; Robert, J.; Heljanko, K.; Framling, K. IoTEF: A Federated Edge-Cloud Architecture for Fault-Tolerant IoT Applications. *J. Grid Comput.* **2020**, *18*, 57–80. [CrossRef]
11. Liu, P.; Willis, D.; Banerjee, S. *ParaDrop: Enabling Lightweight Multi-Tenancy at the Network's Extreme Edge*; IEEE: Piscataway, NJ, USA, 2016; pp. 1–13.
12. Xu, X.; Yu, H.; Pei, X. *A Novel Resource Scheduling Approach in Container Based Clouds*; IEEE: Piscataway, NJ, USA, 2014; pp. 257–264.
13. Filip, I.; Pop, F.; Serbanescu, C.; Choi, C. Microservices Scheduling Model over Heterogeneous Cloud-Edge Environments As Support for IoT Applications. *IEEE Internet Things* **2018**, *5*, 2672–2681. [CrossRef]
14. Kaewkasi, C.; Chuenmuneewong, K. Improvement of Container Scheduling for Docker using Ant Colony Optimization. In *International Conference on Knowledge and Smart Technology*; IEEE: Piscataway, NJ, USA, 2017; pp. 254–259.
15. Kaur, K.; Dhand, T.; Kumar, N.; Zeadally, S. Container-as-a-Service at the Edge: Trade-off between Energy Efficiency and Service Availability at Fog Nano Data Centers. *IEEE Wirel. Commun.* **2017**, *24*, 48–56. [CrossRef]
16. Liu, C.; Huang, C.; Tseng, C.; Yang, Y.; Chou, L. *Service Resource Management in Edge Computing Based on Microservices*; IEEE: Piscataway, NJ, USA, 2019; pp. 388–392.
17. Sami, H.; Mourad, A. Dynamic On-Demand Fog Formation Offering On-the-Fly IoT Service Deployment. *IEEE Trans. Netw. Serv. Manag.* **2020**, *17*, 1026–1039. [CrossRef]
18. Lin, M.; Xi, J.; Bai, W.; Wu, J. Ant Colony Algorithm for Multi-Objective Optimization of Container-Based Microservice Scheduling in Cloud. *IEEE Access* **2019**, *7*, 83088–83100. [CrossRef]
19. Shi, Y.; Chen, S.; Xu, X. MAGA: A Mobility-Aware Computation Offloading Decision for Distributed Mobile Cloud Computing. *IEEE Internet Things* **2018**, *5*, 164–174. [CrossRef]
20. Avgeris, M.; Spatharakis, D.; Dechouniotis, D.; Kalatzis, N.; Roussaki, I.; Papavassiliou, S. Where There Is Fire There Is SMOKE: A Scalable Edge Computing Framework for Early Fire Detection. *Sensors* **2019**, *19*, 639. [CrossRef] [PubMed]
21. Guerrero, C.; Lera, I.; Juiz, C. Genetic Algorithm for Multi-Objective Optimization of Container Allocation in Cloud Architecture. *J. Grid Comput.* **2018**, *16*, 113–135. [CrossRef]
22. Gao, M.; Chen, M.; Liu, A.; Ip, W.H.; Yung, K.L. Optimization of Microservice Composition Based on Artificial Immune Algorithm Considering Fuzziness and User Preference. *IEEE Access* **2020**, *8*, 26385–26404. [CrossRef]
23. Zhu, H.; Wang, H.B.; Bayley, I. Formal Analysis of Load Balancing in Microservices with Scenario Calculus. In Proceedings of the IEEE 11th International Conference on Cloud Computing (CLOUD), San Francisco, CA, USA, 2–7 July 2018; pp. 908–911.
24. Hu, Y.; de Laat, C.; Zhao, Z. Optimizing Service Placement for Microservice Architecture in Clouds. *Appl. Sci.* **2019**, *9*, 4663. [CrossRef]