Mobile Networks and Applications (2023) 28:19–30
https://doi.org/10.1007/s11036-022-01992-w

Task Scheduling and Resource Balancing of Fog Computing in Smart Factory

Ming-Tuo Zhou[1,2,3] · Tian-Feng Ren[1,2] · Zhi-Ming Dai[1,2] · Xin-Yu Feng[1,2]

Accepted: 27 September 2021 / Published online: 3 June 2022

Abstract
With the development of new generation information technology, many traditional factories begin to transform to smart factories. How to process the huge volume data in the smart factories so as to improve the production efficiency is still a serious problem. Based on the characteristics of smart factory, a fog computing framework suitable for smart factory is proposed, and Kubernetes is used to automatically deploy containerized smart factory applications. First, in the scene of fog computing, an improved interval division genetic scheduling algorithm IDGSA (Interval Division Genetic Scheduling Algorithm) based on genetic algorithm is proposed to schedule and allocate tasks in smart factory. We consider the optimization of task execution time and resource balance at same time and combined with IDGSA, the optimized scheduling decision is given. Second, we further design an architecture of cloud and fog collaborative computing. In this scenario, we propose the IDGSA-P (Interval Division Genetic Scheduling Algorithm with Penalty factor) for optimization based on IDGSA. Finally, we carry out simulation experiments to verify the performance of the proposed algorithms. The simulation results show that compared with Kubernetes default scheduling algorithm, IDGSA can reduce data processing time by **50%** and improve the utilization of fog computing resources by **60%.** Compared with traditional genetic algorithm, with fewer iterations, IDGSA can reduce data processing time by **7%** and improve the utilization of fog computing resources by **9%**. And compared with the conventional Joines&Houck method, the proposed IDGSA-P algorithm can converge much faster and archived better optimization results. Further, the simulation shows that IDGSA-P in cloud and fog collaborative computing can reduce the total task delay by **18%** and **7%**, respectively, when compare to only-cloud and only-fog computing.

Keywords Fog computing · Cloud computing · Smart factories · Resource allocation · Genetic algorithm.

1 Introduction

New generation information technologies, such as Internet of things, cloud computing, fog computing, artificial intelligence, big data, etc., have brought valuable development opportunities to many industries [2, 21]. Traditional industry is experiencing the development of information technology. Smart factory was born under such a background [6, 12]. Compared with traditional factories, smart factories need to process a large amount of data. One way is to use remote cloud computing, but there are many disadvantages [5], such as large delay, high bandwidth requirements, and unable to guarantee security and privacy. The emergence of fog computing can alleviate these problems [1]. It transfers computing, storage, control and network functions from cloud to edge devices, so as to reduce data transmission delay and required bandwidth. It allows a group of adjacent end users, network edge devices and access devices to cooperate to complete the tasks requiring resources. Therefore, many computing tasks that originally need to be completed by cloud computing can be effectively completed at the edge of the network through the decentralized computing resources around the data generation equipment.

Smart factory can realize resource virtualization and automatic service deployment through container

This work was supported by the STCSM Science and Technology Innovation Program on Hightech under Grant No. 18511106500.

✉ Ming-Tuo Zhou
mingtuo.zhou@mail.sim.ac.cn

1 Shanghai Institute of Microsystem and Information Technology, Chinese Academy of Sciences, Shanghai, China

2 University of Chinese Academy of Sciences, Beijing, China

3 Fujian Key Laboratory of Intelligent & Connected Commercial Vehicle (Information Science), Fuzhou, China

technology and container automatic orchestration tools [8]. Container is a virtualization technology. Compared with virtual machine, it is lighter and can be quickly deployed on different operating platforms. Currently, docker containers are common. Related orchestration tool is Kubernetes (K8S), a platform tool that can span multiple computing nodes and manage containers on multiple computing nodes. We can containerize the application in the smart factory [9] into a docker container, and then use Kubernetes to automatically deploy the docker container to the appropriate fog computing node [13, 16].

How to reasonably deploy the above containers to the fog computing node of the smart factory and make full use of the fog computing resources is essentially a resource allocation and scheduling problem. At present, there are some related studies on this problem. Shahab Tayeb [15] proposed a new three-tier fog computing architecture, but they did not study the scheduling of fog computing resources. Skarlat [14] proposed a two-layer fog computing architecture and optimized the delay, but they did not consider resource balance. Sagar Verma [17] proposed a cloud architecture and considered load balancing, but this work did not consider the delay optimization problem. Hamid Reza faragardi [4] proposed the application of fog calculation in intelligent factory. Ruilong Deng [3] mainly studied the energy consumption and time delay of cloud computing system. However, they do not consider the delay caused by network communication. Hend Gedawy [7] used a group of heterogeneous fog computing devices to form an edge micro cloud. Wan Jiafu [18] proposed a load balancing scheduling method based on fog computing to solve the complex energy consumption problem of manufacturing clusters in intelligent factories. Both of them consider the delay and resource balance problems at the same time, but the proposed algorithm needs to be further improved. Wenzhu Wang [19] proposed the allocation method of dominant resources under the condition of heterogeneous resources, but the final result does not accord with Pareto optimality. Zening Liu [10, 11] proposed a scalable and stable distributed task scheduling algorithm, but there is a lack of consideration of resource balance and application scenario of smart factories.

As summary, there are some deficiencies in the existing research: First, they basically optimize the processing time of tasks without considering the limited computing resources in the smart factory; Second, it is basically improved in one aspect, and it is not comprehensively optimized in combination with the characteristics of smart factory as a whole. Third, these optimizations are often based on pure fog computing. There are few articles on cloud and fog collaborative computing.

Compared with other studies, according to the task characteristics of smart factory, this paper uses Kubernetes to realize the task automatic deployment in smart factory. The contributions of this paper are summarized as following:

1) Framework: A framework with fog and cloud computing is proposed, in which the tasks are categorized into different types and may be matched to different type of fog computing resources. Compared with the traditional computing framework, the framework is more flexible and can adaptively adjust the use of cloud and fog computing resources.
2) Scheduling algorithm for only-fog resource scheduling: In this paper, we proposed IDGSA (Interval Division Genetic Scheduling Algorithm) to optimize tasks scheduling for use of only-fog resources scenarios. Different to other works, we consider to jointly optimize task processing time delay and resource usage balance. Basically, the scheduling optimization problem is nondeterministic polynomial (NP), and may not be solved by traditional genetic algorithm because of double optimization objects and easy to get local optimization solutions. As such, we propose to construct a dual-objective fitness function and to allocate individuals into three intervals based on their fitness values, and to use different operators for individuals of different intervals in the process of evolution. This algorithm may keep more diversities and can avoid local optimization.
3) Scheduling algorithm for joint cloud and fog resource scheduling: Further, in case of joint cloud and fog resource scheduling, we propose IDGSA-P (Interval Division Genetic Scheduling Algorithm with Penalty factor)algorithms to optimize resource allocation, in which the scheduling work is modeled as a NP problem with constraints so a penalty factor is introduced to solve the problem.
4) Simulations: We carried out simulation experiments to verify our proposed algorithms. Compared with the traditional genetic algorithm and the default algorithm of K8S, our proposed algorithm has faster iteration speed and better optimization effect. IDGSA can reduce data processing time by 50% and improve the utilization of fog computing resources by 60% when compare with K8S default scheduling method. Compared with traditional genetic algorithm, with fewer iterations, IDGSA can reduce data processing time by 7% and improve the utilization of fog computing resources by 9%. With the simulation settings, IDGSA-P algorithm converges faster than Joines&Houck method and may get better results. It may be 18% and 7% faster when compare to only-cloud and only-fog computing, respectively.

2 Scheduling framework in smart factories

In the smart factory with cloud computing resources, task allocation and management are very important. At present, the mainstream virtualization technology, such as docker technology, is commonly used for resource management and allocation. This virtualization technology can bring many benefits. Kubernetes is a commonly used docker management framework, which can realize the flexible distribution and deployment of tasks. In this section, we will introduce a framework of computing with Kubernetes in smart factories.

2.1 Task classification and fog resource classification

Firstly, in order to process tasks efficiently, we divide factory tasks into three categories based on their properties:

1) Real-Time Tasks (RTT): delay sensitive tasks, such as: judgment of the operating status of key smart devices and faults. This type of tasks must be completed within specified time delay.
2) General Tasks (GT): it is a type of task that can withstand a certain amount of delay, such as: monitoring the quality of products, processing video information in the factory.
3) Storage Tasks (ST): tasks that has no requirement on latency but needs storage spaces, such as: data analysis for each production line, analysis of the energy consumption.

Secondly, we categorize fog nodes with different characteristics, too:

1) Exclusive Fog Computing Nodes (EFCN): These nodes are close to the equipment, have outstanding performance, and can give result feedback in the fastest time.
2) Balanced Fog Node (BFN): The processing performance and storage performance of this type of fog node are relatively good, and it can process most tasks in the smart factory.
3) Fog Storage Node (FSN): Such nodes have average processing performance, but good storage performance, closer to the cloud, and can upload data to the cloud data center for processing at an appropriate time.

2.2 Task assignment

Secondly, we study how to assign different type of tasks to appropriate fog computing nodes after task classification. In previous works, tasks were usually deployed in the way of fog and cloud classification, but it did not involve automatic deployment and container application monitoring. Therefore, combined with Kubernetes, this paper further improves the fog computing framework in smart factory.

The components in Kubernetes mainly include:

Etcd: used to save the configuration and object status information of all networks in the cluster.
API server: Provides API interface and is the hub of data interaction and communication between other modules.
Scheduler: The execution module of Kubernetes scheduler dispatched the task to the appropriate node by algorithm.
RC (replication controller)/deployment: Monitors the number of tasks in the Kubernetes cluster to stabilize the number of tasks.

The scheduling framework of the smart factory proposed in this paper is shown in Fig. 1. Firstly, the tasks in the smart factory are containerized, and then the containerized tasks are labeled as different type. This information will be stored in Etcd. Then, the Scheduler will interact with API server to obtain the tasks that have not been deployed in Kubernetes cluster, and automatically deploy the tasks to the corresponding nodes according to their labels. For the container application whose label is RTT, it is assigned to EFCN for processing. For a container application with a label of ST, it is deployed on FSN. And for the container application with a label as GT, it is deployed on the BFN. In the whole process, the deployment module in Kubernetes will monitor the container applications in Kubernetes and recreate them when a container application has problems.

Because label is the most widely used container in handling tasks, how to reduce the delay and improve the resource utilization is the most important question. Therefore, we propose the corresponding system model and algorithms to reasonably allocate such tasks.

3 Fog resource scheduling and balancing in smart factories

In this section, we will discuss the problem of resource scheduling and balancing in intelligent factory in fog computing scenario. We establish delay optimization model and resource balance model to describe the whole fog computing system, and design TSB factor to balance delay and resource usage. Finally, we designed IDGSA algorithm to solve the optimization problem.

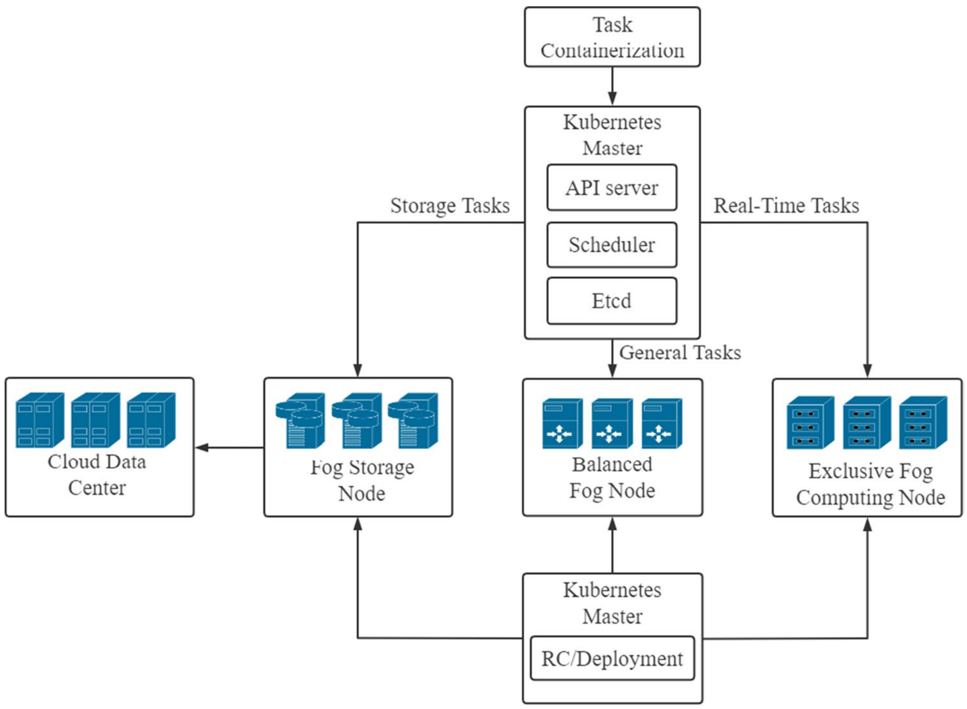

Fig. 1 Scheduling frameworks in smart factories incorporating Kubernetes

3.1 System model

In an intelligent manufacturing factory, a production line is a service. The services in the smart factory can be defined by app_j, $app_j \in A$, where j represents the $j-th$ service in the smart factory and A represents the collection of all services in the whole smart factory. All container applications used during the execution of a production line app_j are defined as set S, and the $i-th$ container application is defined as ms_i. $ms_{i,\,cpu}$, $ms_{i,\,mem}$ respectively represent the minimum requirements of container application ms_i for CPU and memory of fog computing node. When all container applications monopolize one CPU for task processing, the time required is a unit time, expressed in ut(unit time), because the number of different container applications may have different requirements when processing tasks, Therefore, ms_i^{req} is used to indicate how many such container applications are needed on this production line.

In the process of task processing, container applications are executed in sequence, so container applications may use each other's data or processing results. Therefore, two container applications with consumption relationship can be expressed as (ms_{prov}, ms_{cons}) indicating that ms_{cons} needs the processing results of ms_{prov}. The fog computing node resource pool can be defined as set P. the fog computing nodes in the node resource pool are represented by pm_l. If a container application ms_i is deployed on the node pm_l, it can be represented as $alloc\ (ms_i) = pm_l$. $pm_{l,\,cpu}$ and $pm_{l,\,mem}$ respectively represent the CPU resources and memory resources of the fog computing node.

3.2 Balance factor

In this paper, there are three optimization objectives: (1) Task execution time; (2) Cluster resource balance; (3) Cluster equalization degree and delay equalization factor.

3.2.1 Task execution time

The execution time of the task can be expressed by the maximum of the completion time of all container tasks, as shown below

$$T_{service} = \max\left\{S\left(ms_i\right)\right\}\ i = 1, 2, 3\ldots \tag{1}$$

Where $S(ms_i)$ represents the data processing time of the container application ms_i.

The calculation time of a single container application can be expressed as

$$S\left(ms_i\right) = \max\left\{T_{run}\left(ms_i\right), T_{wait}\left(ms_i\right)\right\} \tag{2}$$

Where $T_{run}\left(ms_i\right) = \frac{ms_{i,cpu} \times R}{pm_{l,cpu}}$ represents the processing time of the container application ms_i(unit:ut), and R in the molecule represents the number of all container applications on pm_l.

If the container application has a consumption relationship with other container applications, then we have:

$$T_{run} = \max\left\{S(ms_j) + T_{trans}(ms_j)\right\}, \forall ms_j \mid (ms_j, ms_i)_{prov/cons} \quad (3)$$

Where, $T_{trans}(ms_j)$ represents the time when the provider container application transmits the processing results to the consumer container application. For ease of calculation, if two container applications are deployed on the same fog computing node, $T_{trans}(ms_j) = 0$. If they are deployed on different fog computing nodes, $T_{trans}(ms_j) = 0.1 \times S(ms_j)$.

3.2.2 Cluster resource balance

In order to make full use of the fog computing resources in the cluster and keep the cluster balance, we should try our best to ensure that the usage of various resources in the node are balanced. At the same time, the resource usage of each node in the cluster should also be consistent. Therefore, the cluster resource balance can be divided into two parts: (1) the balanced use of various resources on a single fog computing node; (2) The balanced use of resources among different nodes in the whole cluster.

$$B_{all} = B_{cluster} + B_{single} \quad (4)$$

And we define:

$$B_{cluster} = \sigma\left(PM_{usage}^{pm_l}, if\, \exists ms_i | alloc(ms_i) = pm_l\right) \quad (5)$$

$$PM_{usage}^{pm_l} = \frac{1}{2}\left(\frac{\Sigma_{ms_i} ms_{i,cpu}}{pm_{l,cpu}} + \frac{\Sigma_{ms_i} ms_{i,mem}}{pm_{l,mem}}\right) \quad (6)$$

$B_{cluster}$ is equal to the balance degree of different fog computing nodes in the whole cluster. The smaller the value of $B_{cluster}$, the more uniform the use of resources on different fog computing nodes in the whole cluster.

Similarly, we have:

$$B_{single} = \left| \frac{pm_{l,cpuusage}}{pm_{l,cpu}} - \frac{pm_{l,memusage}}{pm_{l,mem}} \right| \quad (7)$$

B_{single} represents the resource usage balance of a fog computing node, where $pm_{l,\, cpuusage}$, $pm_{l,\, memusage}$ represent the CPU and memory used on the node respectively. In this way, it can ensure that in a single fog computing node, there will be no excessive use of one resource and almost no use of another resource, so that all resources on the node can be fully utilized.

3.2.3 Tradeoff between service-time and balance (TSB)

This paper defines an equilibrium factor as another optimization objective of the model. This optimization objective integrates the two objectives of task calculation time and cluster equilibrium, which can enable the factory to rely on $T_{service}$ or B_{all} by adjusting the weight of cluster equilibrium in TSB. It can be expressed by the following formula:

$$TSB(i) = \beta \times B_{all}^{norm}(i) + (1-\beta) \times T_{service}^{norm}(i) \quad (8)$$

$B_{all}^{norm}(i)$, $T_{service}^{norm}(i)$ represents the normalized result of $B_{all}(i)$, $T_{service}(i)$.

In summary, container application scheduling problems in smart manufacturing plants can be summarized as follows:

$$\begin{aligned} &\min T_{service}, B_{all} \\ &s.t.\ \ alloc\ (ms_i) = pm_l, \forall ms_i \in\ \ app_j \end{aligned} \quad (9)$$

The above problems are NP (non-deterministic polynomial) problems, so heuristic genetic algorithm can be used to solve them. For the assignment of tasks in a factory, genetic algorithm is the most common one. This paper proposes the IDGSA algorithm and introduces the concept of interval partition to improve the performance of the traditional genetic algorithm.

3.3 IDGSA algorithm

When be applied to smart factories, traditional genetic algorithms cannot handle the dual-objective problem, and some invalid results may possibly be not properly handled. Furthermore, the iteration speed is relatively slow and has problem of local optimum results.

Therefore, this paper presents IDGSA algorithm to deal with the situation where the initialization results in an individual that is not a feasible solution (to assign container applications on overutilized nodes to other nodes), and to improve the idea that the crossover mutator and roulette selection operator in the traditional genetic algorithm use interval partitioning. Compared with the traditional genetic algorithm, it improves the iteration speed and achieves better results, while guaranteeing the diversity of the population and avoiding falling into the local optimal situation.

The traditional genetic algorithm selects good individuals, crosses over, mutates them, and generates the next generation using a method called roulette selection operator. The idea of the roulette selector is to select individuals according to their fitness values, cross and mutate them, and produce the next generation of individuals.

Traditional roulette operator thought: the probability of an individual being selected is proportional to its fitness function value, set the population size to N, the fitness of an individual x_i to $f(x_i)$, then the selection probability of an individual x_i is

$$P(x_i) = \frac{f(x_i)}{\Sigma_j f(x_j)} \tag{10}$$

Although this selection operator is simple to construct and widely used, it has drawbacks because it can preserve excellent genes, but it has always been saved by individuals with higher fitness values. As a result, the individual diversity of the population is poor and the results tend to be locally optimal, resulting in no better results. To avoid premature convergence of IDGSA algorithm and traditional genetic algorithm and discard some search subspaces, this paper proposes interval division method to treat individuals with different fitness values. Figure 2 shows the diagram of IDGSA algorithm that incorporating the interval division method. First, the population is initialized and then to evaluate the rationality of each individual. If an individual is not rational then it will be modified until it becomes rational. For rational individuals, in each evolution we calculate its fitness value. When its calculated fitness value is high, then we keep the individual. When the fitness is low, we mutate it and try to let it change to an individual with high fitness value in next round evolution. For individuals with medium fitness value, we propose to use an interval division selection operator to generate new individual. We call the proposed interval division selection operator as interval partitioning roulette selection operator.

The operation steps of the proposed interval partitioning roulette selection operator is described as following:

1) According to the dual-objective fitness function in the algorithm (which is TSB, Eq. 8), the fitness values of all individuals in the population are calculated.
2) Select the individuals with the best and worst fitness values in the whole population, then divide the individuals with the best and worst fitness values into M grades and assign the individuals of the population to the corresponding grade areas according to the fitness values.

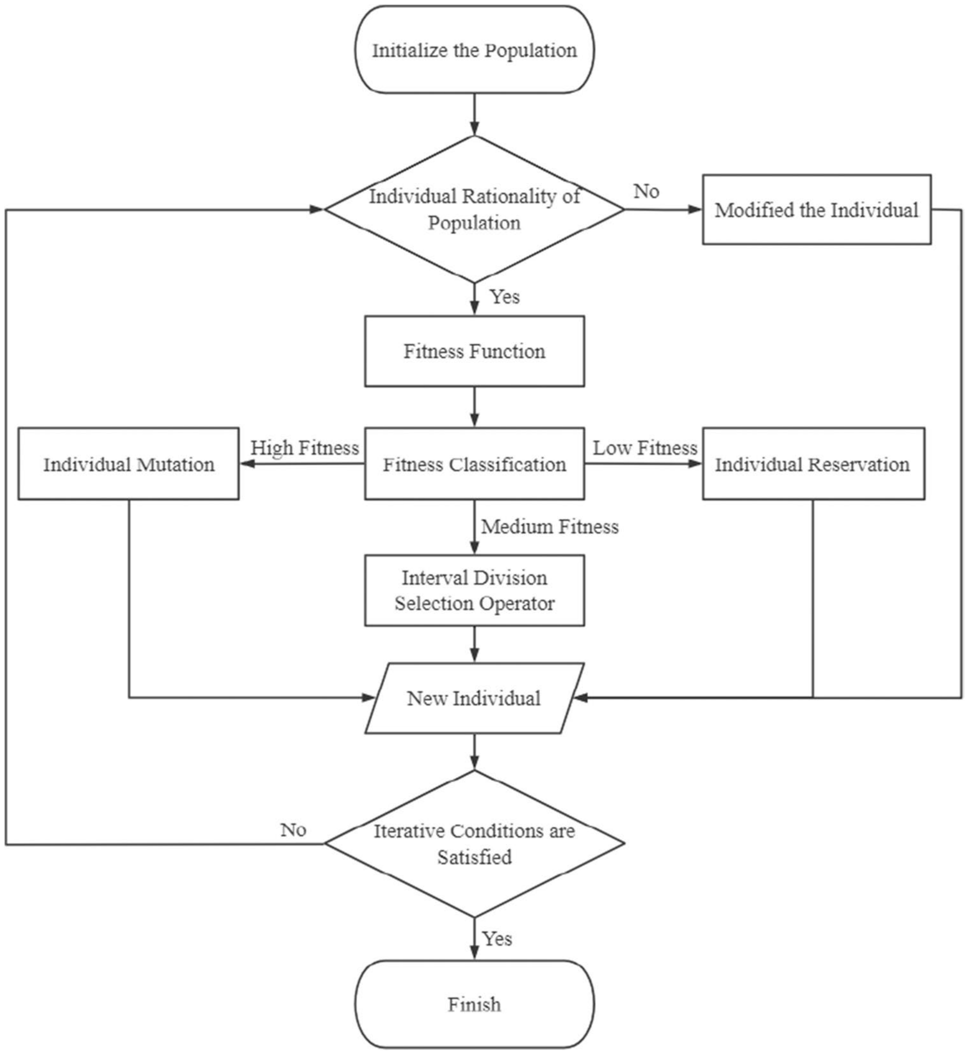

Fig. 2 The proposed IDGSA algorithm incorporating interval division selection based on individual's fitness values

3) Calculate the average fitness values for each of the M regions (dividing all individual values in this region by the number of individuals in this region);
4) In M regions, assume that the probability of each hierarchical region being selected is P_m, where P_m is the average fitness value of the current hierarchical region divided by the sum of the average fitness values of all hierarchical regions (M), and then calculate P_m;
5) Assume that the probability of an individual x_i being selected in each of the M-level zones is $P_m^{x_i}$, which is the sum of the fitness values of the individual divided by the fitness of all the individuals in the hierarchical zones in which they are located;
6) Calculate the probability that each individual in the entire population will be selected $P(x_i) = P_m \times P_m^{x_i}$

The whole population is defined as P, and there are N individuals in a population. The fitness value of individual x_i is $f(x_i)$ calculated from the dual-bojective fitness function. At the second iteration, the fitness values of individuals in the entire population P can be expressed as

$$P(T) = \{f(x_1),f(x_2),\dots f(x_N)\} \tag{11}$$

The range of fitness values for subspaces of population P is

$$diff = \frac{f(x_i)_{max} - f(x_j)_{min}}{M} \tag{12}$$

Therefore, the population P of the $T-th$ iteration can be divided into

$$P(T) = \left\{f(x_1)_1^T,f(x_2)_2^T,\dots f(x_i)_m^T\right\} \tag{13}$$

Where $f(x_i)_m^T \in [f(x_i)_{min} + diff \times (m-1), f(x_i)_{min} + diff \times m]$. Combined with the previous analysis, we can get that

$$P_m = \frac{f_m^T(x_i)_{avg}}{\Sigma_m f_m^T(x_i)_{avg}} \tag{14}$$

$$P(x_i) = \frac{f_m^T(x_i)}{n_m \times \Sigma_m f_m^T(x_i)_{avg}} \tag{15}$$

It can be seen that $P(x_i)$ is inversely proportional to n_m, so if the number of individuals in an interval is too large, the probability of being selected will be reduced, and if the number of individuals in an interval is small, the probability of being selected will be increased. So, when the fitness of all individuals in the whole population is too different, the roulette selector with interval partitioning can avoid the early elimination of poor fitness individuals and improve the diversity of selection.

4 Joint scheduling of fog and cloud resource in smart factories

In above, we discussed the problem of task scheduling and resource balance of smart factory with only-fog computing resources, and proposed IDGSA for optimization. In this section, we will further discuss the task scheduling problem under the scenario of cloud and fog collaborative computing. We try to introduce cloud computing into the fog computing resource pool, and ultimately achieve cloud-fog collaborative computing.

Therefore, we set up a fog management node in the fog computing resource pool to assign tasks to specific fog nodes and determine when to collaborate with cloud resources for computing. The specific scheduling architecture is shown in Fig. 3.

4.1 System model of cloud and fog collaborative computing

The task assignment framework can be regarded as a weighted undirected graph $G(V,E)$ as shown in Fig. 4, where $V = \{F_1, F_2, \dots, F_i, F_M, \dots, F_m, C\}$ is a vertex set, F_i is the fog node, F_M is the fog manager,C is cloud server.$E = \{e_{F_M F_1}, \dots, e_{F_M F_i}, \dots, e_{F_M C}\}$ is the edge set,$e_{F_M F_i}$

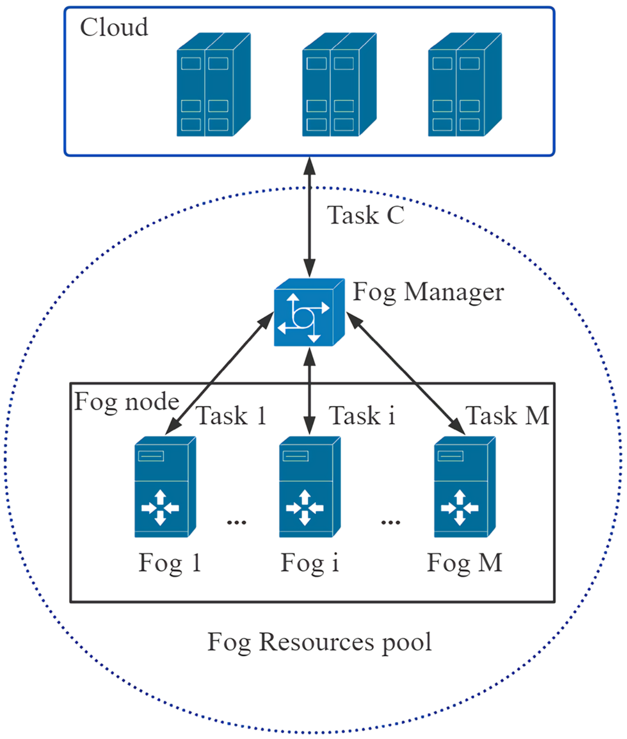

Fig. 3 Task assignment framework in case with joint cloud and fog computing

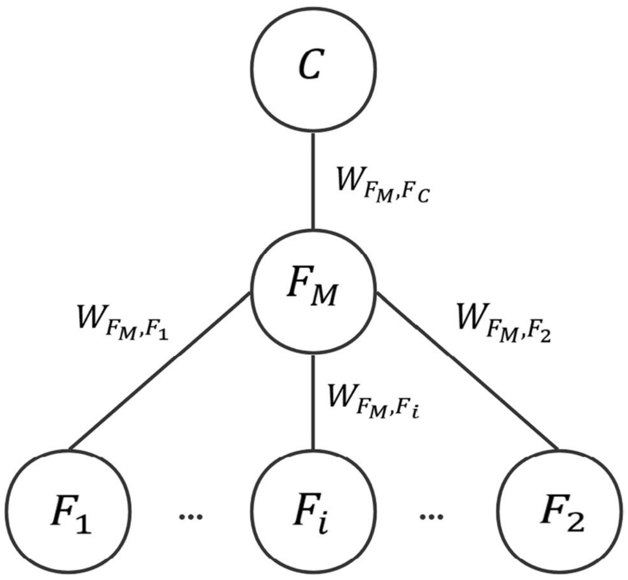

Fig. 4 Undirected graph of fog exclusive node

a communication link between fog node F_i and fog manager, weight on edge is $W_{F_M F_i}$. The fog manager only allocates tasks and does not perform specific tasks. The computing power of F_i is denoted as A_{F_i}, the computing power of C is denoted as A_C.For m fog nodes and 1 cloud server, task D can be divided into m subtasks of different sizes, which is denoted as $D = \{d_1, d_2, \ldots, d_j, \ldots, d_C, \ldots, d_m\}$, where d_C is the subtask that is assigned to the cloud server.

The total time consumed by node F_i to execute task d_j is

$$T_a(F_i, d_j) = \frac{d_j}{A_{F_i}} + T_t(F_i, d_j) \tag{16}$$

where $\frac{d_j}{A_{F_i}}$ is the computing delay, $T_t(F_i, d_j)$ is the communication delay.

Similarly, the total time consumed by C is:

$$T_a(C, d_C) = \frac{d_c}{A_C} + T_t(C, d_c) \tag{17}$$

The total time delay $T(d_j, d_C)$ cost by the task can is:

$$T(d_j, d_C) = \max\{T_a(F_i, d_j), T_a(C, d_C)\}\ j = 1, 2, 3, \ldots \tag{18}$$

Finally, we have a constrained optimization problem:

$$\begin{aligned} \min\ \max\ &\{T(d_j, d_c)\} j = 1, 2, 3, \ldots \\ &s.t. 0 \le d_j, d_c \le D \\ &\sum_{j=1}^{m-1} d_j + d_c = D \end{aligned} \tag{19}$$

4.2 IDGSA-P algorithm

The original problem is an optimization problem with constraints, and the IDGSA cannot solve it directly, so we propose IDGSA-P to convert the original constrained problem into an unconstrained problem.

We design an adaptive penalty function based on the concept of offset. First, construct the chromosome of genetic alg orithm:$x_i = \{\delta_1, \delta_2, \ldots, \delta_j, \ldots, \delta_m, \delta_C\}$, where δ_j is scale factor:

$$\delta_j = \frac{d_j}{D}\ j = 1, 2, 3, \ldots \tag{20}$$

Then, the original constraint can be converted to:

$$\begin{aligned} s.t. 0 \le g_j(x_i) \le 1 j = 1, 2, 3 \ldots \\ h(x_i) - 1 = 0 \end{aligned} \tag{21}$$

where $g_j(x_i)$ and $h(x_i)$ is:

$$g_j(x_i) = \delta_j\ j = 1, 2, 3 \ldots \tag{22}$$

$$h(x_i) = \sum_{j=1,}^{m-1} \delta_j + \delta_C \tag{23}$$

We define $g_z(x_j)$, $g_f(x_j)$ as the positive and negative offset value of $g_j(x_i)$. Then we have:

$$g_z(x_i) = \max\{0, g_j(x_i) - 1\}\ j = 1, 2, 3 \ldots \tag{24}$$

$$g_f(x_i) = \min\{0, g_j(x_i)\}\ j = 1, 2, 3 \ldots \tag{25}$$

Similarly, we have the positive and negative offset value of $h(x_j)$:

$$h_z(x_i) = \max\{0, h(x_i) - 1\} \tag{26}$$

$$h_f(x_i) = \min\{0, h(x_i) - 1\} \tag{27}$$

In the chromosome set $\{x_i\}$, we can find the infeasible solution set according to (24)~(27), it is denote as$\left\{x_j^{reject}\right\}$, thus, the positive and negative offset degrees constrained in eq. (19) are introduced:

$$D_z(f(x_i)) = \frac{f_z(x_i)}{\max\left\{f_z\left(x_j^{reject}\right)\right\} - f_z(x_i)} \tag{28}$$

$$D_f(f(x_i)) = \frac{|\, f_f(x_i)\, |}{|\, \min\left\{D_f\left(f_j^{reject}\right)\right\} | - |\, f_f(x_i)\, |} \tag{29}$$

Based on (28) ~ (29) we can define $\phi(x_i)$, $\xi(x_i)$ as the offset degree of the current solution x_i to the inequality constraint function, its expression is:

$$\phi_j(x_i) = \begin{cases} 1 \; 0 \le g_j(x_i) \le 1 \\ e^{-D_z(g_z(x_i))} \; g_j(x_i) > 1 \\ e^{-D_f(g_f(x_i))} \; g_j(x_i) < 1 \end{cases} \tag{30}$$

$$\xi(x_i) = \begin{cases} 1 \; h(x_i) = 1 \\ e^{-D_z(h_z(x_i))} \; h(x_i) > 1 \\ e^{-D_f(h_f(x_i))} \; h(x_i) < 1 \end{cases} \tag{31}$$

Finally, according to (30) ~ (31), we set the penalty function in the genetic algorithm as:

$$\psi(x_i, \beta) = \left(2 - \frac{1}{2}\left(\xi(x_i) + \frac{1}{m}\sum_{j=1}^{m}\phi_j(x_i)\right)\right)^{\beta} \tag{32}$$

where β is penalty factor.

By introducing a penalty function, the original constrained optimization problem is finally transformed into an unconstrained optimization problem [19]:

$$\min\max\{TSB(j,\gamma) \times \psi(x,\beta)\} \;\; j = 1, 2, 3, \ldots \tag{33}$$

5 Simulation and results

In this part, we design experiments to test our algorithm. Various parameters and equipment in the experimental process are given in the following sections.

5.1 Simulation environment

In the simulation of idgsa algorithm,this paper uses socks shop, a smart manufacturing factory that produces socks to carry out simulation experiments, and the parameter values in the experiment come from the analysis of socks shop [20]. Socks shop is a micro service demo application, which simulates the actual operation of a smart factory producing socks. The resource usage of each container application comes from the load test of the demo, and the number of each container application required by a task comes from the analysis of customer behavior model graph (CBMG) [7]. Table 1 shows the resource consumption.

The IDGSA-P algorithm is simulated in the cloud and fog collaborative computing scenario, so we deal with the cloud node and join the cloud server at the same time. We can see the configuration in Table 2.

Table 1 Container application in socks shop

Name	Id	NUM	CPU	Memory
Worker	Ms1	3	0.47	353.7
Shipping	Ms2	2	0.0028	406.9
Master	Ms3	3	0.0004	696.6
Orders	Ms5	2	0.0048	939.6
Front-end	Ms7	15	0.0528	101.7
Router	Ms8	15	0.0076	11.4
Consul	Ms14	3	0.7508	33.3

Table 2 Parameters related to fog node and cloud server

Parameter type	F_1	F_2	F_3	C
Frequency(GHz)	4	4.5	5	10
Transfer rate(Mbps)	100	350	200	50
Packet loss rate	0.0232	0.140	0.0154	0.0200

5.2 Results and analysis

This part introduces simulation experimental results to verify the proposed scheduling algorithms and presents analysis.

5.2.1 Results of IDGSA with different optimization weights

In this paper, the performance of IDGSA algorithm under different balance optimization weights is simulated experimentally, which changes the weight of balance optimization target in the balance factor TSB.

As can ben seen in Fig. 5(a), with β increasing, the computing time of tasks increases and the balance in the cluster decreases (representing a gradual increase in resource utilization). And as shown in Fig. 5 (b), when β is 0.9, TSB will get the minimum value, that is, it can ensure that the computing time of tasks and the cluster balance are both small (cluster resource utilization is high), and two optimization goals reach a relative optimal value (balance optimization).

5.2.2 Comparison between IDGSA algorithm and traditional genetic algorithm (traditional GA)

To prove the advantages of IDGSA algorithm, IDGSA algorithm is compared with traditional GA by simulation. In the simulation experiment, the population size is 200, the number of iterations is 4000, the horizontal coordinate is the number of iterations, and the vertical coordinate is the balance factor TSB, task completion time, cluster balance, respectively.

Figure 6 (a) shows the balance factor TSB obtained by using IDGSA algorithm and traditional GA. It can be seen

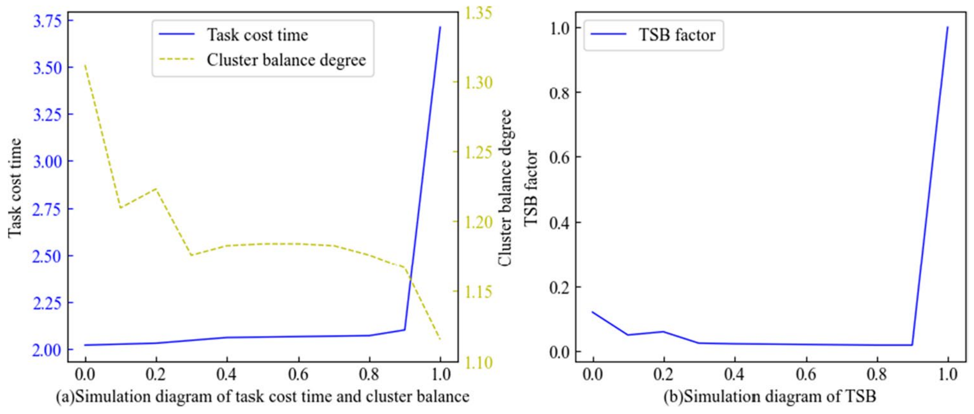

Fig. 5 Simulation results under different equalization optimization weights

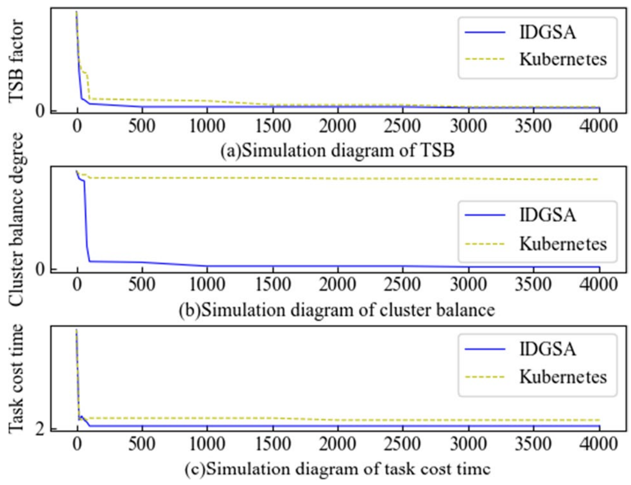

Fig. 6 Performance simulation diagram between IDGSA and traditional GA

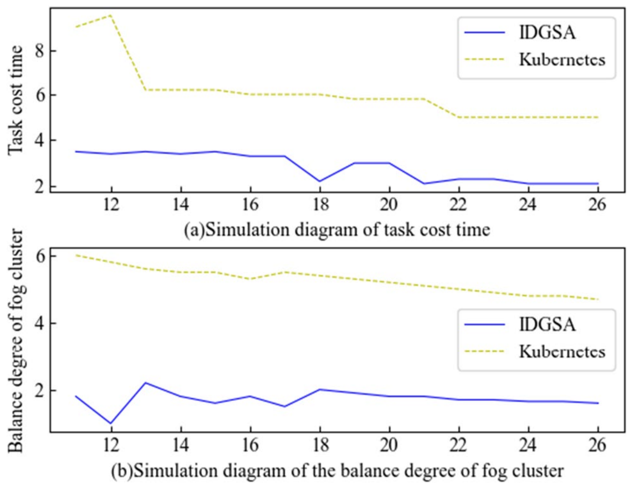

Fig. 7 Performance simulation diagram of IDGSA and Kubernetes algorithm

from that IDGSA algorithm has obtained the optimal solution when iterating to 500 times, while the traditional GA needs to iterate to 1500 times to obtain the optimal solution, and we can also see that the IDGSA can get a better result. Figure 6 (b) and (c) show the cluster balance degree and task execution time obtained by IDGSA algorithm and traditional GA respectively. It can also be seen that IDGSA algorithm has achieved better results than traditional GA when the number of iterations is about 500, while genetic algorithm needs to iterate to 1500 times. Therefore, we can have two conclusions: First, the final results of IDGSA are better than the traditional GA. Second, IDGSA algorithm can achieve the result in fewer iterations.

5.2.3 Comparison of IDGSA algorithm and Kubernetes default algorithm

IDGSA algorithm is needed to replace the default scheduling algorithm of Kubernetes, so the performance of the IDGSA algorithm is compared with that of the Kubernetes default algorithm. By changing the number of fog computing nodes in the fog computing resource pool, the performance of the two algorithms is compared when the resource situation changes. In Fig. 7 (a) and (b), the horizontal coordinates represent the number of fog computing nodes. As the number of fog computing nodes increases, the task execution time and resource balance of both algorithms decrease. The results show that the processing time of IDGSA algorithm is reduced by about 50% and the resource utilization is increased by about 60% compared with the default scheduling algorithm of Kubernetes. It can be also clearly seen that IDGSA algorithm has shorter task processing time and higher resource utilization than Kubernetes default algorithm regardless of whether fog computing resources are sufficient or scarce.

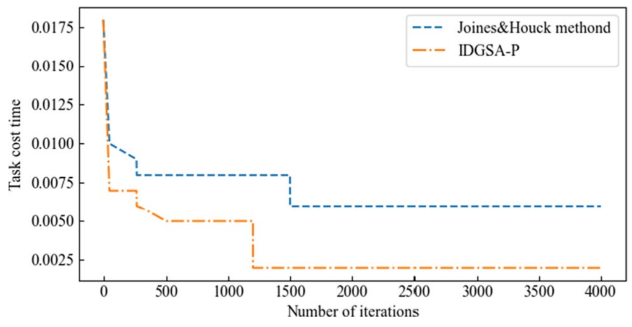

Fig. 8 Performance comparison between IDGSA-P and Joines&Houck method

5.2.4 Comparison between IDGSA-P and Joines & Houck methond

Joines&Houck method is a commonly used method to solve constrained optimization problems. This method also optimizes the traditional genetic algorithm and converts the constrained problem into an unconstrained problem.

In this simulation, IDGSA-P and Joines&Houck method are compared and the results are shown in Fig. 8. We have conducted 4000 iterations, and the iterative results are shown in the figure below. It can be seen that IDGSA-P has faster convergence speed and can better find the optimal solution than the Joines&Houck method.

5.2.5 Performance comparison under different computing scenarios

In order to verify the difference between the computing method of fog exclusive node and cloud server used in this paper and other computing methods, it is simulated and compared with pure cloud computing and pure fog computing. The number of cycles during simulation is 3000, and IDGSA-P is adopted.

As can be seen from Fig. 9, compared with the pure cloud computing method, the cloud computing method adopted in this paper has powerful computing power and can reduce the computing delay of tasks, but the cloud server is generally far away from the smart factory, the network link bandwidth is limited, and there is a large communication delay. Therefore, the total delay of task processing in pure cloud computing is greater, and with the increase of the number of tasks, the performance gap is becoming more and more obvious. Compared with the pure fog computing method, the cloud server with powerful computing power is introduced, and the total task delay can be reduced by reasonably allocating the task proportion. Compared with the traditional cloud computing and fog computing, the method adopted in this paper can reduce the total task delay by 18% and 7% when the number of users is 500, so as to reduce the total task delay and improve the plant efficiency when processing the real-time tasks of the smart factory.

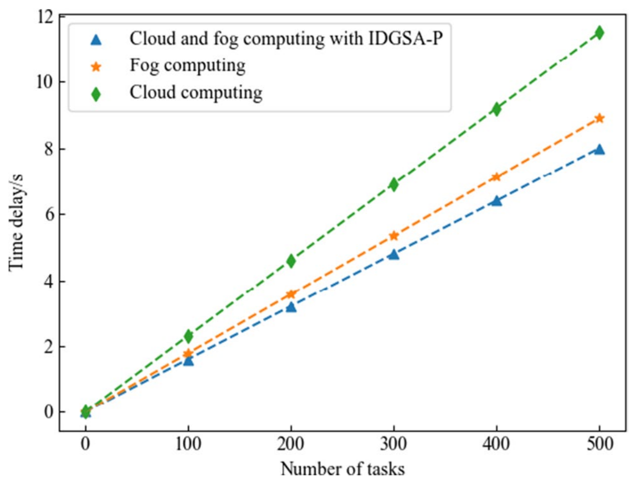

Fig. 9 Performance under different computing scenarios

6 Conclusion

In literatures there is few studies of task scheduling frameworks of fog and cloud computing resources in smart factories with consideration of using task scheduling tools like Kubernetes. Furthermore there is lack of studies with joint optimization of task processing time delay and resource balance. Based on the characteristics and requirements of smart factories, this paper improves the fog computing architecture for smart factories. Firstly, we discuss the scheduling and resource balancing of smart factory tasks in fog computing scenario and design the equalization factor TSB to comprehensively consider the delay and equalization problem. To solve this problem, we proposed IDGSA algorithm. Compared with the traditional genetic algorithm, IDGSA can achieve better optimization and faster convergence speed. Secondly, we further establish a cloud and fog collaborative computing model to flexibly allocate fog computing and cloud computing resources. According to the model, IDGSA-P is designed on the basis of IDGSA to optimize the original problem. Finally, we did simulation experiments to verify the proposed algorithms. The simulation results show that IDGSA has better performance than the traditional genetic algorithm and K8S default algorithm. At the same time, compared with only-cloud computing and only-fog computing, cloud and fog collaborative computing can make more effective use of resources and bring lower computing delay under IDGSA-P.

Future work of this study include considering more optimization objectives like limited computing resource or/ and communication bands, real-time analysis model with consideration of resource faults, and implementation of the algorithm in smart manufacturing testbed using K8S.

References

1. Chen, N., Yang, Y., Zhang, T., Zhou, M. T., Luo, X., & Zao, J. K. (2018) Fog as a service technology. IEEE Commun Mag, pp 1–7
2. Chiang M, Tao Z (2017) Fog and IoT an overview of research opportunities. IEEE Internet of Things J 3(6):854–864
3. Deng R et al (2017) Optimal workload allocation in fog-cloud computing toward balanced delay and power consumption. IEEE Internet of Things J 3.6:1171–1181
4. Faragardi, H. R., et al. (2018) A time-predictable fog-integrated cloud framework: one step forward in the deployment of a smart factory. Rtest
5. Fazio M, Celesti A, Ranjan R, Chang L, Chen L, Villari M (2016) Open issues in scheduling microservices in the cloud. IEEE Cloud Comput 3(5):81–88
6. Gazis, V., Leonardi, A., Mathioudakis, K., Sasloglou, K., & Sudhaakar, R. (2015) Components of fog computing in an industrial internet of things context. 2015 12th Annual IEEE international conference on sensing, communication, and networking – workshops (SECON Workshops). IEEE
7. Gedawy, H., et al. (2018) An energy-aware IoT femtocloud system. 58–65
8. Gribaudo M, Iacono M, Manini D (2018) Performance evaluation of replication policies in microservice based architectures. Electron Notes Theor Comput Sci 337:45–65
9. Ha, J., Kim, J., Park, H., Lee, J., & Jang, J. (2017) A web-based service deployment method to edge devices in smart factory exploiting Docker. International Conference on Information & Communication Technology Convergence. IEEE
10. Joines, J. A., and C. R. Houck (1994) On the use of non-stationary penalty functions to solve nonlinear constrained optimization problems with GA's. IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence IEEE
11. Liu, Z., et al. (2018) DATS: dispersive stable task scheduling in heterogeneous fog networks. IEEE Internet of Things J. 1–1
12. Mourtzis D, Vlachou E, Milas N (2016) Industrial big data as a result of IoT adoption in manufacturing. Procedia CIRP 55:290–295
13. Production-Grade Container Orchestration (2017) https://kubernetes.io/. (2017). Google
14. Skarlat, O., et al. (2016) Resource provisioning for IoT services in the fog. IEEE International Conference on Service-oriented Computing & Applications IEEE
15. Tayeb, S., S. Latifi, and Y. Kim (2017) A survey on IoT communication and computation frameworks: an industrial perspective. Computing & Communication Workshop & Conference IEEE
16. Tihfon GM, Park S, Kim J, Kim YM (2016) An efficient multitask PaaS cloud infrastructure based on docker and access for application deployment. Clust Comput 19(3):1–13
17. Verma, S., et al. (2016) An efficient data replication and load balancing technique for fog computing environment. International Conference on Computing for Sustainable Global Development IEEE
18. Wan, J., et al. (2018) Fog computing for energy-aware load balancing and scheduling in smart factory. Industrial Informatics, IEEE Transactions on 14.10. 4548–4556
19. Wang, W., et al. (2015) Multiple resources scheduling for diverse workloads in heterogeneous datacenter. 2015 4th International Conference on Computer Science and Network Technology (ICCSNT) IEEE
20. Weaveworks, ContainerSolutions: Socks shop – a microservices demo application (2016). https://microservices-demo. github.io/
21. Y. Yang, Luo, X, Chu, X., & Zhou, M. T. (2020). Fog-enabled intelligent IoT systems