Hindawi
Mobile Information Systems
Volume 2022, Article ID 5830760, 10 pages
https://doi.org/10.1155/2022/5830760

Research Article

A Heuristic Task Scheduling Strategy for Intelligent Manufacturing in the Big Data-Driven Fog Computing Environment

Rong Zhou

The University of Hong Kong, Hong Kong 999077, China

Correspondence should be addressed to Rong Zhou; zhourong2020@gsm.pku.edu.cn

Received 14 June 2022; Revised 23 July 2022; Accepted 29 July 2022; Published 26 August 2022

Academic Editor: Shadi Aljawarneh

A heuristic task scheduling strategy for intelligent manufacturing in the big data-driven fog computing environment is proposed to address the problem that current resource scheduling and allocation methods in the fog computing environment cannot comprehensively consider the dynamics and uncertainty of resources, resulting in the prolonged delay and high energy consumption. First, a system model with three computing modes for intelligent manufacturing is constructed based on intelligent terminal devices, fog nodes, fog servers, fog gateways, and the cloud. Then, the objective function is optimized by jointly considering the delay matrix, the energy consumption matrix, and the reliability matrix, and corresponding constraints are given based on the selection of computing modes, the decision variables of fog nodes as well as the constraints about the delay. Finally, the intervals of crossover-mutation operators are divided according to the fitness value, and individuals of the population are updated by taking different operations based on the operators in different intervals, so as to achieve an improvement on the traditional genetic algorithms. Meanwhile, a fog resource scheduling algorithm is proposed based on the improved adaptive genetic algorithm. Simulation experiments are conducted to compare and analyze the delay, energy consumption, and reliability of the proposed method with three other methods under the same conditions. The results show that the proposed method has the lowest delay and energy consumption and the highest reliability, with values of 361.3 s, 352.4 J, and 94.6%, respectively, when the number of task requests is 500. The performance is better than the other three comparison methods.

1. Introduction

With advances in information technology such as the Internet of Things (IoT) and fog computing, the "smart factory" controlled by cyber-physical systems (CPS) is also developing rapidly [1, 2]. The Intellectualization of information and production methods has led to a higher level of interconnectivity, smarter devices, and more powerful data processing in intelligent manufacturing. Through the connected information, statistical data, and dynamic analysis, intelligent manufacturing makes the production smarter, leaner, more effective, and more energy-efficient [3–5]. The introduction of the IoT technology in intelligent manufacturing places new and higher demands on data sensing, collection, consolidation, transmission, and reverse control in smart factories. At the same time, the spread of intelligent devices and terminals and the use of various kinds of sensors will bring about ubiquitous sensing and connectivity, generating a constant flow of industrial data [6–8].

The efficient processing of large volumes of data in traditional manufacturing can be achieved using cloud computing. However, due to the complexity of networking in the IoT and the limited computing capacity of devices in the bottom layer, traditional data processing methods cannot be used for delay-sensitive applications of intelligent manufacturing services [9, 10]. Fog computing, as a highly virtualized platform that locates on the edge of the local network, can provide computing and storage services near the underlying network and the Internet. Therefore, it is a good solution to the problem of rapid response and bandwidth consumption for delay-sensitive applications [11, 12]. When terminal devices request services from the cloud data center, fog computing can first perform data filtering, data pre-processing, and analysis before delivering it to the cloud computing system, thus reducing the burden on the cloud data center. Fog computing resources are therefore ideal for terminal users today [13]. Due to the dynamic and uncertainty of resources and the high variability and unpredictability of the fog computing environment, rational resource scheduling and allocation are particularly important, which have become a research hotspot in the industry [14, 15].

A heuristic task scheduling strategy for intelligent manufacturing in a big data-driven fog computing environment is proposed to address the problems of prolonged delay, high energy consumption, and low reliability of current resource scheduling and allocation methods in fog computing environment. The basic ideas are: (1) First, a system model for intelligent manufacturing that can be used for computing mode selection is constructed. (2) The corresponding objective function and constraints are given in the consideration of delay, energy consumption, and reliability. (3) The applicability of the genetic algorithm to the problem under study is improved by making appropriate refinements to it. Compared with traditional task scheduling strategies, the main contributions of the proposed method can be summarized as follows:

(1) Intelligent devices, fog nodes, fog servers, fog gateways, and the cloud are taken into account when constructing the system model, greatly improving the utilization of fog computing resources.
(2) The objective function is optimized in terms of three aspects of task scheduling: delay, energy consumption, and the success rate of task execution.
(3) The traditional genetic algorithm is improved by dividing the crossover-mutation operators into intervals to further enhance the reliability of task scheduling.

The remaining chapters of this paper are arranged as follows: the second chapter introduces the efficient resource allocation and task scheduling strategies for intelligent manufacturing in the fog computing environment, and some current research results. The third chapter establishes the system model. The fourth chapter introduces the proposed fog resource scheduling method based on a genetic algorithm. In Chapter 5, experiments are designed to verify the performance of the proposed algorithm. The sixth chapter is the conclusion, which summarizes this study and puts forward the further improvement direction.

2. Related Works

Some scholars have done relevant research and achieved certain results on resource allocation and task scheduling strategies for intelligent manufacturing in the fog computing environment. Li et al. standardized and normalized the attributes of resources, and combined the fuzzy clustering method with particle swarm optimization method to reduce the scale of resource search and divide the resources [16]. It proposed a new resource scheduling algorithm based on optimized fuzzy clustering for fog computing. However, the method only optimizes the processing time and cost of the tasks without considering the limited computational resources in the smart factory. In Yang et al., aiming at the multi-objective task scheduling problem in fog computing, a multi-objective task scheduling model was designed based on the Pareto optimal solution [17]. However, the method cannot reduce the overall reliance on large data centers and the Internet data are distant from the users. Ren et al. addressed the problem of limited resources in mobile devices and used fog computing to improve the WAN delay for delay-sensitive and resource-intensive applications [18]. It proposed an improved three-layer fog-to-cloud architecture and the schedule fit algorithm, which can provide computational resources and transmission delay according to the delay sensitivity of applications. The method fails to minimize the processing delay of the task and needs further improvement. Based on Lyapunov drift and the penalty function on the queue length, a scheduling strategy for tasks in the queues was proposed in Reference [19] to decide the number of tasks to be offloaded to the underloaded fog nodes to fully utilize the computational resources offered by all fog nodes in the network. However, this approach does not consider the service delay. By extending the architecture of fog computing, Tang et al. proposed a computational resource allocation scheme based on stable matching for open fog computing environment [20]. Based on the idea of stable matching, the allocation problem between tasks and computing service devices is solved by combining the priority list of subtasks and the preference lists of subtasks and computing service devices. However, the fog computing network is heavily used in this method. Alqahtani et al. classified requests to real-time, important, and time-tolerant, and proposed a scheduling method for allocating customers' requests to the resources of the cloud-fog environment by considering load balancing among resources while allocating requests to them [21]. But the algorithm does not take the cost and energy consumption of resource scheduling into account. In Reference [22], in order to solve the resource scheduling and load balancing problems in fog computing, an optimized fuzzy clustering-based resource scheduling and dynamic load balancing algorithm was proposed. On this basis, an enhanced fuzzy C-means and the crow search optimization algorithm in fog computing were used to solve the problem. However, the method does not combine the characteristics of smart factories as a whole to fully optimize them.

3. System Model

3.1. System Model Description. Different from the traditional computing mode selection strategies, the proposed computing mode selection strategy is applicable to fog computing platforms with the combination of heterogeneous computing modes. In the fog computing platform, if the traditional strategies are still used, not only will the resource utilization of fog computing be reduced, but also the tasks which are related to intelligent manufacturing will not be guaranteed to be real-time, energy-efficient, or reliable. In order to improve the utilization of resources in fog computing, a system model of computing mode selection for intelligent manufacturing is established, which is shown in Figure 1.

As can be seen from Figure 1, the system model of computing mode selection for intelligent manufacturing consists mainly of intelligent terminal devices, fog nodes, fog servers, fog gateways, and the cloud. The intelligent terminal devices can either execute tasks locally or transmit the tasks that they cannot execute to other computing nodes. The fog node is the middleware that connects the intelligent terminal device to the cloud, and using the fog node as well as the cloud can enhance the computing capacity of the intelligent terminal devices. The fog node provides a real-time and distributed fog computing model for intelligent terminal devices in task execution. The cloud provides a remote and centralized cloud computing model for intelligent terminal devices in task execution. Different from the traditional strategies where all of the tasks are transmitted to the cloud to be executed, each task in the fog computing environment has a choice of three computing modes.

Intelligent terminal devices forward the tasks through the fog nodes. Task requests from intelligent terminal devices are transmitted to the fog nodes, and all information of task requests is gathered at the fog gateway and subsequently forwarded to the fog server. The fog server can sense the global information of the fog nodes, which provides a reference for the dynamic selection of the computing modes of the task. The fog server manages the tasks, calculates the priority of each task, and adjusts the task queue according to the priority of the task, and then it formulates the optimal selection scheme of computing modes for the task. The fog server sends the selection scheme of computing modes to the fog gateway, and the fog gateway then sends the set of allocated tasks to the corresponding intelligent terminal devices, thus implementing the function of computing mode selection for the intelligent terminal devices when executing tasks.

All devices in the system model are described below. The system model contains z intelligent terminal devices, n fog nodes and a cloud server. The set of intelligent terminal devices is represented by Z, where $Z = \{E_1, E_2, E_3, \ldots, E_z\}$ and the attributes of the intelligent device E_i can be modeled as a triple (C_i, P_{0i}, P_{1i}). C_i represents the computing capacity of the intelligent terminal device E_i. P_{0i} represents the transmission power of the intelligent terminal device E_i. P_{1i} denotes the computation power of the intelligent terminal device E_i. N denotes the set of fog nodes, where $N = \{F_1, F_2, F_3, \ldots, F_n\}$. And the computation power of the fog nodes is denoted as J_N. The computing capacity of the cloud server is denoted as J_C. T is a compound task and can be partitioned into n subtasks according to certain rules, which can be represented as $T = \{T_1, T_2, T_3, \ldots, T_n\}$. The attributes of the task T_i can be modeled as a triple $(D_i, \rho_i, t_{\max i})$. D_i represents the amount of data of the task T_i. ρ_i represents the computational density of the task T_i. $t_{\max i}$ denotes the maximum tolerance time of the task T_i. The system model of computing mode selection shows that each

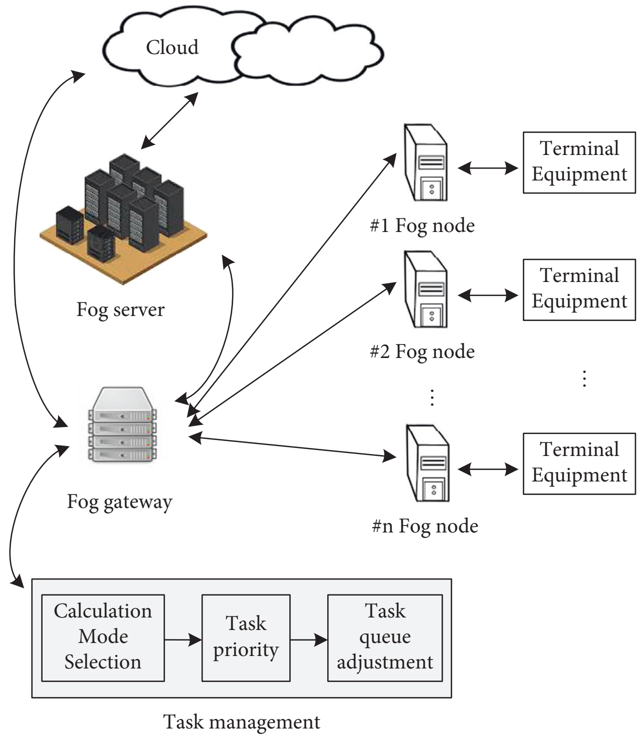

FIGURE 1: The system model of computing mode selection for intelligent manufacturing.

task can be executed through one of the three computing modes. The task can be either executed locally in the intelligent terminal device or transmitted to a fog node or a cloud server for execution. For the task T_i, the delay and energy consumption in the local execution mode are recorded as t_{i1} and e_{i1}, respectively. The delay and energy consumption in fog computing mode are recorded as t_{i2} and e_{i2}, respectively. And the delay and energy consumption in cloud computing mode are recorded as t_{i3} and e_{i3}, respectively.

3.2. Optimization Objectives.

3.2. Optimization Objectives. In order to facilitate the description of the computing mode selection problem and better formulate the computing mode selection algorithm proposed in this section, relevant definitions and optimization objective functions are given to provide a basis for implementing the selection of computing modes. Based on the above analysis, the delay of the task T_i can be expressed as follows:

$$t_i = \alpha_{i1} t_{i1} + \alpha_{i2} t_{i2} + \alpha_{i3} t_{i3}, \quad (1)$$

where $\{\alpha_{i1}, \alpha_{i2}, \alpha_{i3}\}$ represents the set of decision variables for the selection of the computing mode for the task T_i. When $\alpha_{i1} = 1$, it indicates that the task T_i selects the local execution mode, which means that the task T_i will be executed by the intelligent terminal device E_i. Otherwise, $\alpha_{i1} = 0$. When $\alpha_{i2} = 1$, it indicates that the task T_i selects the fog computing mode, which means that the task T_i will be executed by a fog node. Otherwise, $\alpha_{i2} = 0$. And when $\alpha_{i3} = 1$, it indicates that the task T_i selects the cloud computing mode, which means that each T_i will be executed by a cloud server. Otherwise, $\alpha_{i3} = 0$. The task T_i can only selects one of the computing modes, i.e., $\alpha_{i1}, \alpha_{i2}, \alpha_{i3} \in \{0, 1\}$.

We define the computing mode selection decision matrix as A, where the number of tasks being executed based on (1) is taken as the row and the number of computing modes that can be selected are taken as columns, as follows:

$$A = \begin{Bmatrix} \alpha_{11} & \alpha_{12} & \cdots & \alpha_{1J} \\ \alpha_{21} & \alpha_{22} & \cdots & \alpha_{2J} \\ \cdots & \cdots & \cdots & \cdots \\ \alpha_{I1} & \alpha_{I2} & \cdots & \alpha_{IJ} \end{Bmatrix}, \quad (2)$$

a_{ij} denotes the decision variables for the task T_i in the computing model j, where $i = 1, 2, 3, \ldots, I$ and $j = 1, 2, 3, \ldots, J$.

We define the delay matrix as T_t. According to the computing mode selection decision matrix A, the delay of each task in the task set T under three computing modes can be obtained. Thus, the delay matrix T_t for the task set T can be constructed as

$$T_t = \begin{Bmatrix} t_{11}, t_{12}, \ldots, t_{1J} \\ t_{21}, t_{22}, \ldots, t_{2J} \\ \ldots, \ldots, \ldots, \ldots \\ t_{I1}, t_{I2}, \ldots, t_{IJ}, \end{Bmatrix}, \quad (3)$$

where t_{ij} indicates the delay of the task T_i in the computing mode j, $i = 1, 2, 3, \ldots, I$, $j = 1, 2, 3, \ldots, J$. For tasks that select the fog computing mode, they will be allocated to all of the fog nodes according to the polling algorithm. h_{in} denotes the selection decision of fog nodes for the task T_i, and $h_{\text{in}} = 1$ denotes that the task T_i selects to be executed by the fog node lF. Otherwise, $h_{\text{in}} = 0$. Hence, $h_{\text{in}} \in \{0, 1\}$ and $\sum_{n=1}^{N} h_{\text{in}} = 1$, $(i = 1, 2, \ldots, I)$.

The energy consumption matrix is defined as E. Based on the computing mode selection decision matrix A, the energy consumption of each task in the task set T under three computing modes can be calculated. Hence, the energy consumption matrix E for the task T can be written as

$$E = \begin{Bmatrix} e_{11}, e_{12}, \ldots, e_{1J} \\ e_{21}, e_{22}, \ldots, e_{2J} \\ \ldots, \ldots, \ldots, \ldots \\ e_{I1}, e_{I2}, \ldots, e_{IJ} \end{Bmatrix}, \quad (4)$$

where e_{ij} represents the energy consumption of the task T_i in the computing mode j, $i = 1, 2, 3, \ldots, I$, $j = 1, 2, 3, \ldots, J$.

The reliability matrix is defined as K, and it is used to evaluate the execution performance of each task in the task set T within its maximum tolerance time. The reliability matrix can be expressed as

$$K = \begin{Bmatrix} k_{11}, k_{12}, \ldots, k_{1j} \\ k_{21}, k_{22}, \ldots, k_{2j} \\ \ldots, \ldots, \ldots, \ldots \\ k_{i1}, k_{i2}, \ldots, k_{ij} \end{Bmatrix}, \quad (5)$$

where k_{ij} represents the execution performance of the task T_i in the computing mode j, and it should meet the following constraints:

$$k_{ij} = \begin{cases} 0, & \alpha_{ij} t_{ij} \le t_{\max i}, \\ 1, & \alpha_{ij} t_{ij} > t_{\max i}. \end{cases} \quad (6)$$

If the task T_i can be completed within its maximum tolerance time, the task T_i will be executed successfully. Otherwise, the task T_i fails. The number of successfully executed tasks is counted as z_C and the higher the value of z_C, the better the reliability of the task. The success rate of a task is given by p, which can be calculated as

$$p = \frac{z_C}{z} \times 100\%. \quad (7)$$

The energy consumption of all tasks in the task set T can be described as

$$e_S = \sum_{i}^{I} \left(\alpha_{i1} e_{i1} + \alpha_{i2} e_{i2} + \alpha_{i3} e_{i3} \right). \quad (8)$$

The optimization objective of the computing mode selection strategy is to minimize the task delay and energy consumption, which can be formulated as

$$f_o = \min \begin{Bmatrix} \max\limits_{1<i<I} \left(\alpha_{i1} t_{i1} \right) \\ \max\limits_{1<n<N} \left(\sum\limits_{i=1}^{I} \alpha_{i2} h_{ni} t_{n2,i} \right) \\ \max \left(\sum\limits_{i=1}^{I} \alpha_{i3} t_{i3} \right), e_S \end{Bmatrix}. \quad (9)$$

The constraints are shown as

$$\begin{cases} \alpha_{i1}, \alpha_{i2}, \alpha_{i3} \in \{0, 1\}, & i = 1, 2, 3, \ldots, I, \\ \alpha_{i1} + \alpha_{i2} + \alpha_{i3} = 1, & i = 1, 2, 3, \ldots, I, \\ h_{i1}, h_{i2}, h_{i3}, \ldots, h_{in} \in \{0, 1\}, & i = 1, 2, 3, \ldots, I, \\ \sum\limits_{n=1}^{N} h_{in} = 1, \ (i = 1, 2, \ldots, I), & i = 1, 2, 3, \ldots, I, \\ t_i \le t_{\max i}, & i = 1, 2, 3, \ldots, I, \end{cases} \quad (10)$$

where the first and second constraints are about the decision variables of the computing mode selection, ensuring that only one computing mode can be selected for each task. The third and fourth constraints are about the decision variables of the fog node selection, which guarantee that only one fog node can be selected for each task. The last inequality indicates the constraint about the delay for each task.

4. Fog Computing Task Scheduling Algorithm

4.1. Overview of the Algorithm. Based on the aforementioned analysis, a fog resource scheduling method is proposed based on an adaptive bi-adaptive genetic algorithm. Population optimization is carried out by drawing on biological phenomena such as heredity, mutation, natural selection, and hybridization in the theory of biological evolution to find the optimal individual. The algorithm takes the individual in the population as a solution and evaluates the performance of the individual by the dual fitness function. The genetic operations can be divided into three main operators, which are selection, crossover, and mutation. The roulette wheel method is used in the selection operation to maintain genetic diversity in the survival of the fittest. However, as the mutation of chromosomal genes is random, the chromosomal solutions generated may not be feasible solutions, which mean that they do not satisfy the constraint between tasks. Therefore, in the selection operation, a repair strategy is also required for the selected individuals, which allows maximizing the maintenance of genes while changing infeasible solutions into feasible ones. Traditional single-point crossover and partial mapping crossover are combined in the crossover operation. The partial mapping crossover is used for scheduling sequences, and the traditional single-point crossover is used for fog node allocation sequences.

4.2. Improved Genetic Algorithm

4.2.1. Encoding. Initialization of the adaptive genetic algorithm: set the termination conditions of the population iteration as the number of iterations reaching the maximum, the appearance of the optimal solution, or the iteration time reaching the constraint time. Set the optimization weights of delay and communication load as $c1$ and $c2$, respectively.

By encoding chromosomes, genetic algorithms make a chromosome correspond to a solution of the optimization problem. The general encoding methods are direct encoding and indirect encoding. In this paper, indirect encoding is used. Thus, the mapping relationship between task requests and fog nodes is represented by a set of sequences (S, A), where S represents the scheduling ordered sequence and A represents the fog node allocation sequence. Each task request is mapped into a fog resource. A single task request corresponds to a single fog resource, and a single fog resource can correspond to multiple task requests. Each bit in the sequence S and the sequence A represents the number of task requests and the fog resource, and all of them are positive integers. The length of the sequence S and the sequence A is equal to the total number of task requests, which is denoted as k. And the gene value represented in the sequence A corresponds to the number of fog resources allocated to the task. The total number of fog nodes in a fog cluster is l, the total number of task requests is k and a set of sequences (S, A) is called a chromosome.

Chromosome decoding strategy: The fog resource allocation strategy can be derived from the chromosome encoding rules. For example, a chromosome $\{(9,5,1,7,3,8,4,6,10,2), (1,2,3,3,2,1,3,2,1,2)\}$ means that 10 tasks will be allocated to 3 fog nodes for execution. First, the task T_9 will be allocated to the fog node F_1 for execution, then the task T_5 will be allocated to the fog node F_2 for execution, and so on. Finally, the allocation strategy can be obtained, where the task T_9, T_8, and T_{10} will be executed in the fog node F_1 in turn. Also, the task T_5, T_3, T_6, and T_2 will be executed in the fog node F_2 in turn, and the task T_1, T_7, and T_4 will be executed in the fog node F_3 in turn.

4.2.2. Populations Initialization. Population initialization is quite important for scheduling terminal user tasks in a fog computing architecture. The initial population is generated randomly and invalid solutions are excluded based on constraints. Let the size of the initial population be P, and the number of iterations is initialized as $i = 0$. The constraints ask that the time-critical requests from the terminal user must be scheduled to the fog that is close to the user for execution, while requests which are not so urgent should be allocated to the cloud for execution. Meanwhile, in order to avoid severe load disparity, all requests from users cannot be placed on the same resource simultaneously.

4.2.3. Fitness Function. The fitness function is the key to assessing the direction of population evolution when scheduling the terminal user tasks in a fog computing architecture. In order to make the fitness function remain meaningful when the total cost of the service provider is zero, it is defined as

$$f(W) = e^{-W}. \tag{11}$$

W represents the total cost of the service provider.

4.2.4. Genetic Manipulation. Once traditional genetic algorithms have selected suitable individuals using the roulette wheel operator, they take crossover and mutation operations to obtain the next generation of individuals. However, traditional genetic algorithms adopt a uniform crossover variation operator for the evolution of populations, which is not conducive to either the retention of good individuals or the generation of even better individuals. Therefore, in this paper, the interval division of crossover-mutation operators is adopted. After the fitness value of individuals in the population is calculated based on the fitness function, all individuals are divided into different intervals according to the fitness value, which are the mutation interval with low fitness value, the retention interval with high fitness value, and the asymptotic interval with moderate fitness value. The population is then updated by applying different crossover-mutation operators to the individuals in the different intervals.

Individuals with high fitness values are retained directly, ensuring that the best individuals are preserved at each iteration. For individuals with low fitness values, mutation operation is taken to change their chromosomes. By doing this, it gives the opportunity to mutate individuals with low fitness values into individuals with high fitness values, allowing the population to jump out of the local optimum and avoid premature convergence during the iteration, so as to improve the performance of global searching. For individuals with moderate fitness values, the custom interval division operator is used to select the parent individuals and then retain the better individuals by crossover operations. The main idea of the interval division crossover-mutation operator is shown in Figure 2.

The crossover operator, which is a search operator in genetic algorithms, is mainly used when generating new individuals by crossing over the chromosomal genes of the current parent individuals. In this paper, a combination of traditional single-point crossover and partial mapping crossover is used. Specifically, the partial mapping crossover operation is used for the scheduling sequence S and the traditional single-point crossover operation is used for the fog node allocation sequence A. In addition, a mechanism of individual self-adaptation is introduced, which means that the crossover rate changes with the fitness value.

In the partial mapping crossover operation, a crossover selection position is set between any two adjacent genes in each chromosome subsequence S, where the indexes are 1, 2,. . ., $k+1$ from left to right. There are $k+1$ different crossover selection positions in total. A partial mapping crossover is a random selection of two positions from $k+1$ crossover selection positions, and the two parent chromosomes will produce a corresponding pair of genes between these two crossover selection positions. This pair is used to replace the genes of two parent chromosomes, respectively, resulting in two new children chromosomes. For example, if there are two parent sequences {8, 5, 9, 2, 7, 1, 3, 6, 4} and {1, 3, 7, 4, 6, 8, 2, 9, 5}, and the crossover selection positions are randomly generated to be 5 and 8, then the corresponding pair will be {7 : 6, 1 : 8, 3 : 2} and the two children chromosomes resulting from the partial mapping crossover operation will be {1, 5, 9, 3, 6, 8, 2, 7, 4} and {8, 2, 6, 4, 7, 1, 3, 9, 5}. In the single-point crossover operation, a crossover position is set between any two adjacent genes in each chromosome subsequence A, where the indexes are 1, 2,. . ., $k-1$ from left to right. There are $k-1$ different crossover positions in total. In the single-point crossover operation, a crossover position is randomly selected among $k-1$ crossover positions, and all genes in those two chromosomes are exchanged from the positions onwards. For example, if the parent sequences are {3, 1, 1, 2, 3, 2, 1, 3, 2} and {1, 2, 3, 2, 2, 3, 3, 1, 1}, and the randomly generated crossover position is 5, then the two children chromosomes resulting from the single-point crossover operation will be {3, 1, 1, 2, 3, 3, 3, 1, 1} and {1, 2, 3, 2, 2, 2, 1, 3, 2}.

The adaptive crossover rate can be obtained as

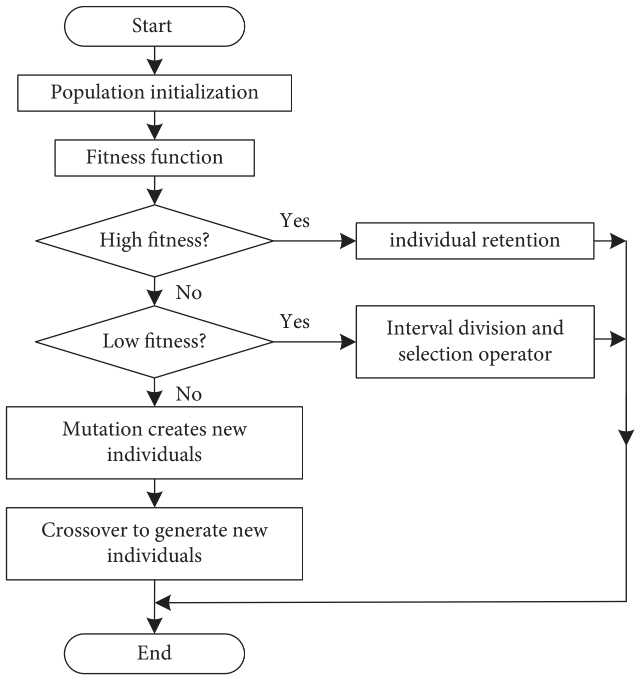
Start
Population initialization
Fitness function
High fitness?
Yes
individual retention
No
Low fitness?
Yes
Interval division and selection operator
No
Mutation creates new individuals
Crossover to generate new individuals
End

Figure 2: The process of interval division of crossover-mutation operators.

$$\begin{cases} f_t = \dfrac{\sum_{m=1}^{M} f_t(m)}{M}, \\ f_c = \dfrac{\sum_{m=1}^{M} f_c(m)}{M}, \end{cases} \qquad (12)$$

$$P_I = \begin{cases} \beta_1 \dfrac{f_{\max} - f_0}{f_{\max} - f}, & f_0 \geq f, \\ \beta_2, & f_0 \geq f, \end{cases}$$

where f_t and f_c denote the average fitness values for the maximum time span and communication overhead, respectively. P_I is the crossover rate of the two individuals. β_1 and β_2 are constant coefficients of the crossover rate and satisfy the constraints of $0 < \beta_1 < 1$ and $0 < \beta_2 < 1$. Two crossed individuals will generate two fitness values each. The fitness function, which can generate the smallest fitness value among the four fitness values, is chosen as the criterion for crossover, i.e., one of $f_t(m)$ and $f_c(m)$ is chosen as the criterion $f(m)$, where $f(m) = [f_t(m) \mid f_c(m)]$ and $f = [f_t \mid f_c]$. f_0 is the larger fitness value of the two individuals $f(m)$ and $f_{\max}$ is the largest fitness value of the current population $f(m)$.

The mutation operator combines the methods of basic bit mutation and inversion mutation. Heuristic mutation is applied for the scheduling sequence S and basic bit mutation is applied for the fog node allocation sequence A. The inversion mutation is applied for the chromosome subsequence S. In the inversion mutation, two gene bits are randomly selected from the k gene bits and the values of genes at these two bits are exchanged. For example, there is the parent sequence {3, 6, 1, 5, 2, 9, 7, 4, 8}, and if the two randomly generated gene bits are 2 and 6, a subsequence {3, 9, 1, 5, 2, 6, 7, 4, 8} will be generated after the inversion mutation. The basic bit mutation is applied for chromosome subsequence A. In the basic bit mutation, a gene bit is randomly selected from the k gene bits and a number from {1, 2, 3,...,n} is randomly selected to replace the gene at the current gene bit, which means that a new fog resource is randomly selected to replace the original fog resource. For example, if the parent sequence is {3, 1 ,1, 2, 3, 2, 1, 3, 2}, and the index of randomly generated gene bit for mutation is 3 and the random gene value selected for replacement is 2, then a child sequence {3, 1, 2, 2, 3, 2, 1, 3, 2} will be generated by the basic bit mutation. Mutation generates new genes and provides the diversity of the population. Like the crossover operator, the mutation operator introduces a mechanism of individual adaption, thus the mutation rate varies with the fitness value.

The adaptive mutation rate can be written as

$$P_V = \begin{cases} \lambda_1 \dfrac{f_{\max} - f_0}{f_{\max} - f}, & f_0 \geq f, \\ \\ \lambda_2, & f_0 \geq f, \end{cases} \tag{13}$$

where P_V is the mutation rate for an individual. λ_1 and λ_2 are constant coefficients of the crossover rate and they satisfy the constraints of $0 < \lambda_1 < 1$ and $0 < \lambda_2 < 1$. The fitness function which can generate the smaller fitness value of the current individual is used as the criterion for mutation, which means that one of $f_t(m)$ and $f_c(m)$ is chosen as the criterion $f(m)$, where $f(m) = [f_t(m) \mid f_c(m)]$ and $f = [f_t \mid f_c]$. $f_{\max}$ is the largest fitness value in the current population $f(m)$.

Genes are constantly evolving in each generation by operations such as selection, crossover, and mutation. Therefore, in order to keep each generation of individuals as feasible solutions, gene repair is required for individuals after the mutation operation.

4.3. Steps for Scheduling Fog Computation Tasks. The flow of the task scheduling algorithm in the fog computing environment for intelligent manufacturing is shown in Figure 3.

The specific process of the algorithm is given as follows:

(1) Population initialization: the users submit the terminal request and the initial population is randomly set based on the request. A counter for the number of iterations is initialized as well.

(2) Fitness value calculation: the fitness values of individuals are calculated according to the fitness function.

(3) Judgment of termination conditions: when the number of evolutionary generations reaches the specified number of iterations, the result is output and a better solution for fog computing task scheduling should be obtained. Otherwise, go to step (4).

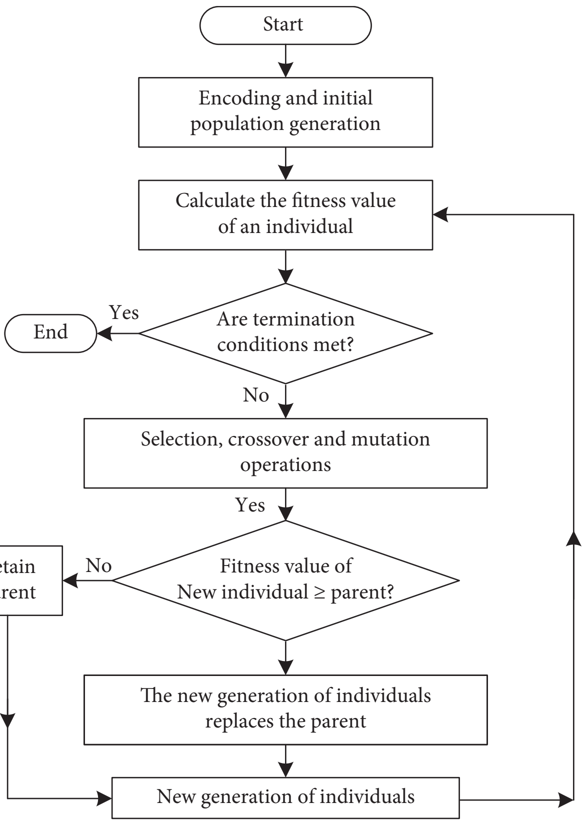
Start
Encoding and initial population generation
Calculate the fitness value of an individual
Yes
End
Are termination conditions met?
No
Selection, crossover and mutation operations
Yes
Retain parent
No
Fitness value of New individual ≥ parent?
The new generation of individuals replaces the parent
New generation of individuals

Figure 3: The flow of task scheduling algorithm in fog computing environment.

(4) Selection operation: a roulette wheel method is used to perform selection operations on the population.

(5) Crossover operation: according to the crossover method in the improved genetic algorithm, two individuals are randomly selected among those produced by the selection operation and then operated according to the crossover rate to generate new individuals.

(6) Mutation operation: according to the mutation method in the improved genetic algorithm, some individuals are selected from the new individuals generated by crossover operation and then operated according to the mutation rate to generate new individuals.

(7) Judgment of fitness value: the fitness value of the mutated individuals is judged, and if the fitness value is smaller than that of the parent, the parent is retained; if it is greater than or equal to that of the parent, the parent is replaced.

(8) Update the iteration counter: add 1 to the iteration counter and go to step (2).

5. Experiments and Analysis

5.1. Simulation Environment and Parameter Settings. The simulation platform is configured with an Intel i5 processor with a CPU of 3.3 GHz and 4 GB of memory. A simulation environment for the selection of computing models for intelligent manufacturing is set up in MATLAB R2020a. A scenario of the intelligent production line is established in the simulation environment, the production object is the personalized candy packaging, and the production link, which is the manufacturing task, is the identification of multiple categories of candy. The size of task data is randomly selected between 1 and 10 Mb, and the number of tasks is taken as 10, 30, 40, 50, 90, 100, 300, 500, 700, and 900, respectively. The parameter set in the simulation is shown in Table 1.

5.2. Simulation Analysis. The delay and energy consumption of the method under the different number of task requests and the reliability of the method for task execution are important evaluation metrics for the performance of the task scheduling method. In the following, the heuristic task scheduling strategy for intelligent manufacturing in big data-driven fog computing environment proposed in this paper is compared and analyzed with the methods proposed in References [16, 17, 20] under the same conditions in terms of three evaluation metrics.

First, the comparative analysis for the delay of the methods is conducted under the different number of task requests. The performance of delay in different methods is simulated in two different cases with a small number of task requests and a large number of task requests. The results are shown in Figures 4 and 5, respectively.

As can be seen from Figures 4 and 5, the delay of all task scheduling strategies tends to rise as the number of tasks increases. However, compared to the other three methods, the heuristic task scheduling strategy proposed in this paper is more advantageous in terms of delay, as it has a smaller delay both in the case of a smaller number of task requests and in the case of a larger number of task requests. The delay is 123.4 s and 361.3 s, respectively when the number of task requests is 100 and 500. This is because the proposed method can better allocate the time-critical task requests to the fog computing resource nodes for execution. Compared with the traditional methods that all of the tasks are executed on the cloud, it can more effectively reduce the delay.

The energy consumption of the methods under the same conditions when there are different numbers of task requests is compared, and the results are shown in Figure 6.

It is shown in Figure 6 that the energy consumption of the system all shows an increasing trend with more task requests. However, the energy consumption of the proposed strategy is always the smallest and the most energy efficient compared to the other three comparison methods. When the number of task requests is 100 and 500, the energy consumption is 205.2 J and 352.4 J, respectively. The energy consumption of the task is determined by the delay of the task and the power of the computing node. In addition, the power of the computing node is a fixed value, and the delay of the task is the main factor that affects the performance of energy consumption of the task. As the delay of the proposed method is small, its energy consumption is also relatively small.

Table 1: Simulation experiment parameters.

Parameter	Value
Number of intelligent terminal devices	200
Number of fog nodes	15
The size of task data (Mb)	1–10
Maximum tolerance time for task	1–3 s
Computing capacity of cloud server (Gcycles/s)	10
Computing capacity of cloud server (W)	50
Computing capacity of fog nodes (Mcycles/s)	300–500
Computational density of tasks (cycles/bit)	300
Transmission power of smart device (M)	3–6
Computational power of smart device (M)	3–6
Network bandwidth of smart device (M)	150
Fog node network bandwidth (M)	120
Cloud server network bandwidth (M)	5
Real-time intensity weight for tasks	0.8
Complex intensity weight for tasks	0.3
Crossover rate	0.8
Mutation rate	0.1
The maximum number of iterations	150
Number of species	100

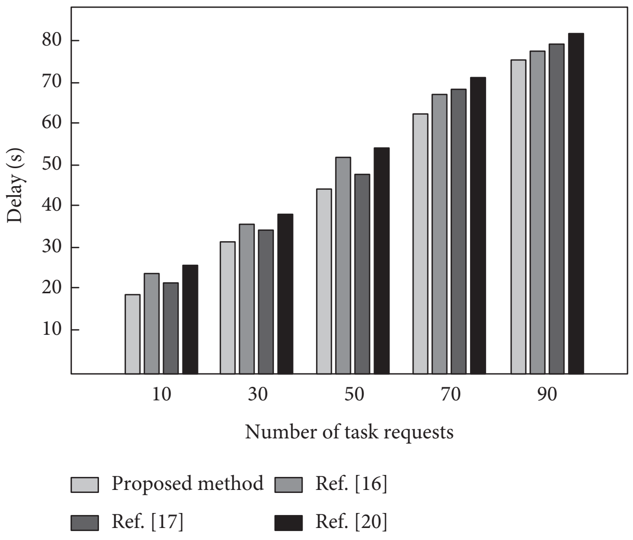

Figure 4: Delay of different methods in the case of a small number of task requests.

Then, the success rate of task execution is calculated for the different number of task requests under the same conditions to verify the reliability of the respective methods. The results of success rates of task execution are shown in Figure 7.

As shown in Figure 7, the reliability of all four methods decreases as the number of tasks increases, but the reliability of the proposed method decreases the least and it remains the highest as the number of task requests changes. The task execution success rates of 95.3% and 94.6% are achieved for 100 and 500 task requests, respectively. This is because the proposed method takes into account the delay matrix, the energy consumption matrix and the reliability matrix in the process of objective optimization. The traditional genetic algorithm is improved in the calculation process. In the proposed method, the fitness function is used to calculate the

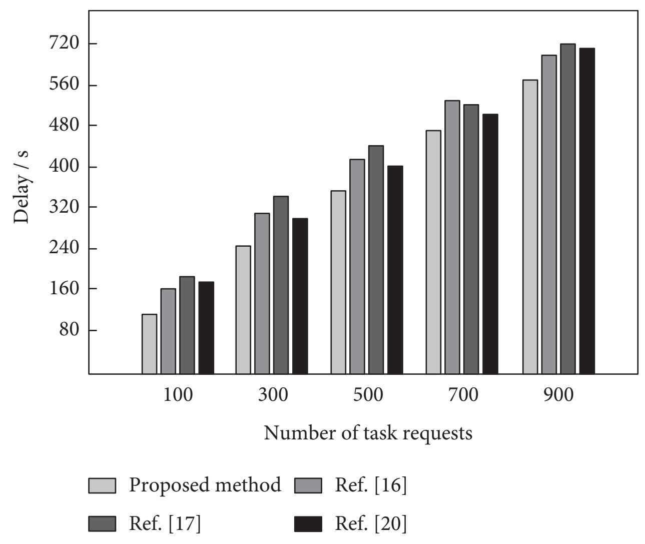

Figure 5: Delay of different methods in the case of a large number of task requests.

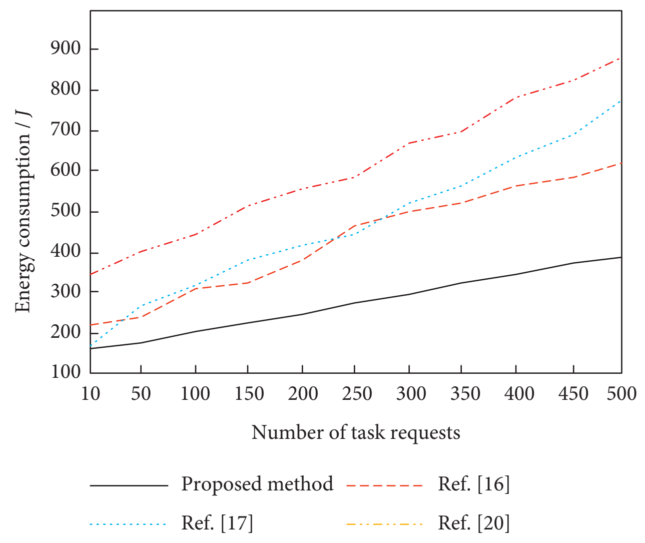

Figure 6: Energy consumption of different methods under different number of task requests.

fitness value of individuals in the population, and all individuals are divided into different intervals according to their fitness values. Also, the crossover-mutation operators are divided by intervals. Therefore, it can generate better individuals in the global scope, which improves the success rate and reliability of the proposed method in task execution.

6. Conclusion

A heuristic task scheduling strategy for intelligent manufacturing in the big data-driven fog computing environment is proposed to address the problems of prolonged delay, high energy consumption, and low reliability of resource scheduling and allocation methods in the fog computing environment. The proposed method and three other comparative methods are compared and analyzed through

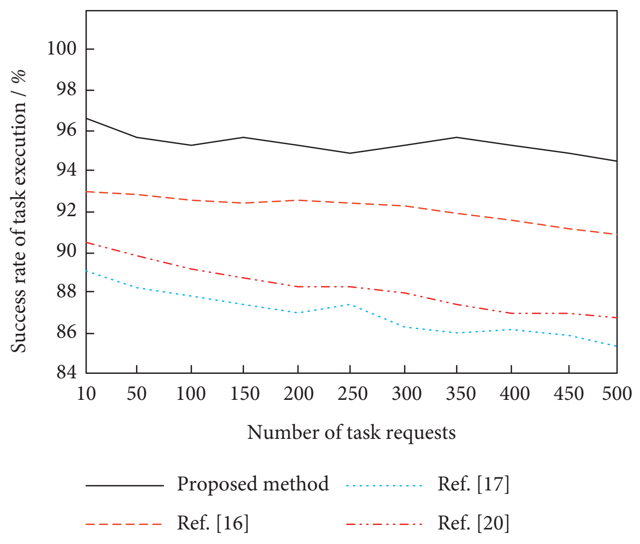

Figure 7: Success rate of task execution of different methods under the different number of task requests.

simulation experiments. The results show that the comprehensive consideration of fog nodes, fog servers, and fog gateways on the basis of intelligent terminal devices and the cloud can significantly improve the utilization of fog computing resources. Optimization of the objective function with the constraints of delay, energy consumption, and success rate can improve the overall performance of the algorithm. By dividing the crossover-mutation operators into intervals, the genetic algorithm can generate better individuals and thus improve the reliability of task scheduling. Future work will delve into the impact of the dynamic characteristics of network and storage resources on fog computing and the improvement method.

Data Availability

The data used to support the findings of this study are included within the article.

Conflicts of Interest

The author declares that there are no conflicts of interest regarding the publication of this paper.

References

[1] S. D. Wang, T. Y. Zhao, and S. C. Pang, “Task scheduling algorithm based on improved firework algorithm in fog computing,” *IEEE Access*, vol. 8, no. 15, Article ID 32385, 2020.
[2] K. Matrouk and K. Alatoun, “Scheduling algorithms in fog computing: a survey,” *International Journal Of Networked And Distributed Computing*, vol. 9, no. 1, pp. 59–74, 2021.
[3] M. Kazemi, S. Ghanbari, and M. Kazemi, “Divisible load framework and close form for scheduling in fog computing systems,” in *Proceedings of the 4th International Conference on Soft Computing and Data Mining (SCDM)*, pp. 323–333, Melaka, Malaysia, January 2020.

[4] Z. N. Liu, X. M. Yang, Y. Yang, K. Wang, and G. Mao, "DATS: dispersive stable task scheduling in heterogeneous fog networks," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 3423–3436, 2019.

[5] H. Rafique, M. A. Shah, S. U. Islam, T. Maqsood, S. Khan, and C. Maple, "A novel bio-inspired hybrid algorithm (NBIHA) for efficient resource management in fog computing," *IEEE Access*, vol. 7, no. 6, Article ID 115760, 2019.

[6] P. Varshney and Y. Simmhan, "Characterizing application scheduling on edge, fog, and cloud computing resources," *Software: Practice and Experience*, vol. 50, no. 5, pp. 558–595, 2020.

[7] H. E. Refaat and M. A. Mead, "DLBS: decentralize load-balance scheduling algorithm for real-time IoT services in mist computing," *International Journal of Advanced Computer Science and Applications*, vol. 10, no. 9, pp. 92–100, 2019.

[8] M. R. Hossain, M. Whaiduzzaman, A. Barros et al., "A scheduling-based dynamic fog computing framework for augmenting resource utilization," *Simulation Modelling Practice and Theory*, vol. 111, no. 37, pp. 201–209, 2021.

[9] B. Jamil, M. Shojafar, I. Ahmed, A. Ullah, and K. Munir, "A job scheduling algorithm for delay and performance optimization in fog computing," *Concurrency and Computation: Practice and Experience*, vol. 32, no. 7, pp. 323–330, 2019.

[10] S. Ghanavati, J. Abawajy, and D. Izadi, "Automata-based dynamic fault tolerant task scheduling approach in fog computing," *Ieee Transactions On Emerging Topics In Computing*, vol. 10, no. 1, pp. 488–499, 2022.

[11] R. M. Ding, X. J. Li, X. Liu, and J. Xu, "A cost-effective time-constrained multi-workflow scheduling strategy in fog computing," in *Proceedings of the 16th International Conference on Service-Oriented Computing (ICSOC)*, pp. 194–207, Hangzhou, China, November 2019.

[12] R. Vijayalakshmi, V. Vasudevan, S. Kadry, and L. K. Ramasamy, "Optimization of makespan and resource utilization in the fog computing environment through task scheduling algorithm," *International Journal of Wavelets, Multiresolution and Information Processing*, vol. 18, no. 1, pp. 37–35, 2020.

[13] J. H. Sun, S. Choudhury, and K. Salomaa, "An online fair resource allocation solution for fog computing," *International Journal of Parallel, Emergent and Distributed Systems*, vol. 3, no. 2, pp. 105–113, 2022.

[14] H. Y. Sun, H. Q. Yu, and G. S. Fan, "Towards energy and time efficient resource allocation in IoT-fog-cloud environment," in *Proceedings of the 16th International Conference on Service-Oriented Computing (ICSOC)*, pp. 387–393, Hangzhou, China, November 2019.

[15] O. H. Ahmed, J. Lu, A. M. Ahmed, A. M. Rahmani, M. Hosseinzadeh, and M. Masdari, "Scheduling of scientific workflows in multi-fog environments using markov models and a hybrid salp swarm algorithm," *IEEE Access*, vol. 8, no. 51, Article ID 189404, 2020.

[16] G. S. Li, Y. C. Liu, J. H. Wu, D. Lin, and S. Zhao, "Methods of resource scheduling based on optimized fuzzy clustering in fog computing," *Sensors*, vol. 19, no. 9, pp. 352–360, 2019.

[17] M. Yang, H. Ma, S. Wei, Y. Zeng, Y. Chen, and Y. Hu, "A multi-objective task scheduling method for fog computing in cyber-physical-social services," *IEEE Access*, vol. 8, no. 12, Article ID 65085, 2020.

[18] Z. B. Ren, T. Lu, X. Wang, W. Guo, G. Liu, and S. Chang, "Resource scheduling for delay-sensitive application in three-layer fog-to-cloud architecture," *Peer-To-Peer Networking And Applications*, vol. 13, no. 5, pp. 1474–1485, 2020.

[19] M. Mukherjee, M. Guo, J. Lloret, R. Iqbal, and Q. Zhang, "Deadline-aware fair scheduling for offloaded tasks in fog computing with inter-fog dependency," *IEEE Communications Letters*, vol. 24, no. 2, pp. 307–311, 2020.

[20] L. Tang, J. Jiang, and K. Gu, "Computing resource allocation scheme based on fog computing," *Computer Engineering and Application*, vol. 55, no. 19, pp. 96–104, 2019.

[21] F. Alqahtani, M. Amoon, and A. A. Nasr, "Reliable scheduling and load balancing for requests in cloud-fog computing," *Peer-To-Peer Networking And Applications*, vol. 14, no. 4, pp. 1905–1916, 2021.

[22] B. Sarma, R. Kumar, and T. Tuithung, "Optimised fuzzy clustering-based resource scheduling and dynamic load balancing algorithm for fog computing environment," *International Journal of Computational Science and Engineering*, vol. 24, no. 4, pp. 343–353, 2021.