



# Real-Time Task Scheduling in Smart Factories Employing Fog Computing

Ming-Tuo Zhou<sup>1</sup>(✉), Tian-Feng Ren<sup>1,2</sup>, Zhi-Ming Dai<sup>1,2</sup>, and Xin-Yu Feng<sup>1,2</sup>

<sup>1</sup> Shanghai Institute of Microsystem and Information Technology,  
Chinese Academy of Sciences, Shanghai, China  
mingtuo.zhou@mail.sim.ac.cn

<sup>2</sup> University of Chinese Academy of Sciences, Beijing, China

**Abstract.** With the development of the new generation of information technology, traditional factories are gradually transforming into smart factories. How to meet the low-latency requirements of task processing in smart factories so as to improve factory production efficiency is still a problem to be studied. For real-time tasks in smart factories, this paper proposes a resource scheduling architecture combined with cloud and fog computing, and establishes a real-time task delay optimization model in smart factories based on the ARQ (Automatic Repeat-request) protocol. For the solution of the optimization model, this paper proposes the GSA-P (Genetic Scheduling Arithmetic With Penalty Function) algorithm to solve the model based on the GSA (Genetic Scheduling Arithmetic) algorithm. Simulation experiments show that when the penalty factor of the GSA-P algorithm is set to 6, the total task processing delay of the GSA-P algorithm is about 80% lower than that of the GSA-R (Genetic Scheduling Arithmetic Reasonable) algorithm, and 66% lower than that of the Joines & Houck method algorithm; In addition, the simulation results show that the combined cloud and fog computing method used in this paper reduces the total task delay by 18% and 7% compared with the traditional cloud computing and pure fog computing methods, respectively.

**Keywords:** Smart factory · Fog computing · ARQ protocol · Genetic algorithm · Resource scheduling

## 1 Introduction

The new generation of information technology such as the Internet, Internet of Things, cloud computing, fog computing, artificial intelligence, big data, etc. has brought valuable development opportunities for many industries [1, 2]. Traditional industries are undergoing technological changes caused by the development of information technology. Smart factory was born under such a background [3, 4]. Compared with traditional factories, smart factories need to process massive amounts of data. At present, there are two mainstream processing methods. One is to use remote cloud computing for processing and return the results. However, this method has many drawbacks [5], such as:

relatively large delay and high bandwidth requirements. And there is no guarantee of security and privacy.

In order to alleviate this problem, another method, fog computing is proposed [6], which transfers computing, storage, control, and network functions from the end to the fog, thereby reducing data transmission delay and the required bandwidth. It allows a group of adjacent end users, network edge devices, and access devices to collaborate to complete tasks that require resources. Therefore, many computing tasks that originally required cloud computing can be effectively completed at the edge of the network through the distributed computing resources around the data generating device.

The rest of this paper is as follows: Sect. 2 is related works, Sect. 3 analyzes the task scheduling framework, Sect. 4 establishes the mathematical model of the system, and discusses the GSA-P algorithm. Section 5 shows and analyzes the simulation results, and we summarize and draw the conclusion in Sect. 6.

## 2 Related Works

There have been some related studies on this issue. Olena Skarlat et al. optimized the resource allocation problem of the cloud and fog [7], reduced the task delay by 39%, and provided a fog resource allocation scheme for delay-sensitive applications. Luxiu Yin et al. replaced virtual machines with containers to execute tasks in smart factories [8], and proposed a container-based task scheduling algorithm. The task execution is divided into two steps: first consider whether the task is accepted or rejected, and then consider whether to run on the local fog node or upload to the cloud. Experimental verification shows that the task scheduling algorithm reduces the task execution time by 10%. Hend Gedawy et al. used a group of heterogeneous mobile and Internet devices to form an edge micro cloud [9], under the condition of ensuring that the energy consumption is below the threshold, maximize the computing throughput and minimize the delay. In order to solve this non-linear problem, they used heuristic algorithms. The simulation results show that the computational throughput is increased by 30% and the delay is reduced by 10% to 40%.

However, the existing research still has some shortcomings: First, the current task scheduling of smart factories is basically based on fog computing or cloud computing, but when the task volume is large, blindly handing over tasks to fog resource computing will lead to The calculation pressure of the fog node is too high, and once the task is completely delivered to the cloud resource for calculation, it will cause the problem of excessive delay; Second, in the actual production environment, network packet loss will have some impact on communication, but in the current research on resource scheduling analysis, the impact of network packet loss on task processing is not considered.

Compared with other research, this article has made overall improvements in system models and algorithms based on the characteristics of today's smart factories. First, we improve the fog computing framework, and then model the problem based on the improved framework, and finally use the improved genetic algorithm to solve the fog computing resource scheduling model. The work and main contributions of this paper are as follows:

- 1) **Framework improvement:** In order to further reduce the delay of real-time tasks in smart factories, this article adds cloud computing resources to the traditional fog computing architecture, so that the current fog computing architecture can disassemble a part of the task when the computing pressure is high. And deliver it to the cloud for processing. At the same time, considering the impact of network packet loss on task delay during actual task transmission, and based on this impact, a time delay optimization model for smart factory real-time tasks under constraint conditions is established.
- 2) **Algorithm improvement:** the real-time task delay model based on the improved framework is a NP (Nondeterministic Polynomially) problem with constraints. For this problem, we usually use heuristic algorithm to solve it [10], such as genetic algorithm [11], but the traditional genetic algorithm has some shortcomings that are difficult to solve the constraint problem. Therefore, this paper proposes GSA-P algorithm, which transforms the original constraint problem into unconstrained problem by constructing penalty function.
- 3) **Simulation verification:** The real-time data simulated in this paper adopts the industrial Internet of Things data of real-time path planning of industrial robots [12], and 4 fog nodes and a cloud server are set up. In the simulation, the difference of approximate optimal solution under different penalty factors is compared. Meanwhile, the differences of optimization results of GSA-P, GSA-R, Joines&Houck method are compared. Finally, the performance improvement of cloud collaborative processing compared with cloud computing and pure fog computing is demonstrated.

### 3 Scheduling Frameworks in Smart Factories

#### 3.1 Task Classification

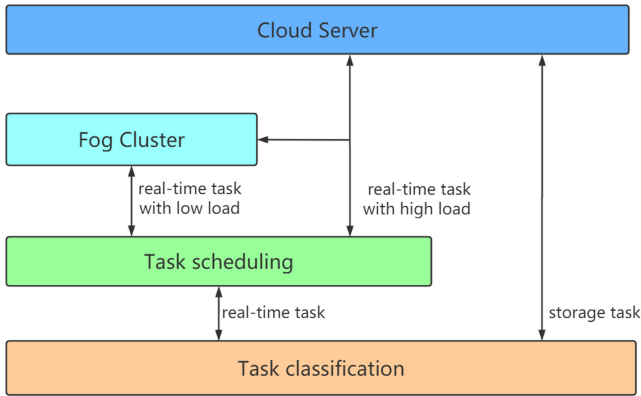
In a smart factory, there are multiple tasks with different requirements for delay and storage. In order to improve the production efficiency of the smart factory, it is necessary to effectively classify the different tasks in the smart manufacturing factory. According to the task's tolerance to delay and the size of task data, tasks can be divided into the following categories.

- 1) **Real-time tasks with low load:** tasks with small data volume and low latency requirements, such as: judgment of the operating status of key smart devices and faults.
- 2) **Real-time tasks with high load:** tasks with large data volume and low latency requirements, such as: monitoring the quality of products in the entire manufacturing process, processing video information in the factory, and production materials in smart factories registration.
- 3) **Storage tasks:** tasks that can accept low latency, such as: data analysis for each production line and analysis of the energy consumption of the entire factory, and other intelligent calculations and processing that can improve the efficiency of the factory.

In the case of 1), due to the small size of data and less requirements for computing resources, the task can be allocated to fog resources for processing.

In the case of 2), due to the large size of data for this type of task, if the data is only deployed on the fog node for calculation, it will cause excessive pressure on the fog node calculation, therefore, cloud computing is needed to reduce the computational pressure of fog nodes.

In the case of 3), regardless of the size of the task's data, it can be directly transferred to the cloud server for processing, and the result will be returned at an appropriate time. Therefore, for the entire smart factory system, the task scheduling framework can be seen in Fig. 1.



**Fig. 1.** Task scheduling framework

### 3.2 Scheduling Framework for Real-Time Tasks

We refer to 1) and 2) in Sect. 2.1 collectively as real-time tasks. For real-time tasks, we need to consider how to divide tasks. For low-load real-time problems, we need to appropriately allocate tasks to the fog node cluster. For high-load real-time tasks, we have to consider allocating tasks on cloud servers and fog node clusters at the same time.

Therefore, we consider adding a fog management node to assign tasks to fog nodes and cloud servers. When a real-time task is delivered, the fog management node determines the task distribution. The scheduling framework of real-time tasks can be seen in Fig. 2.

This paper mainly studies the resource scheduling of real-time tasks in smart factories under this framework.

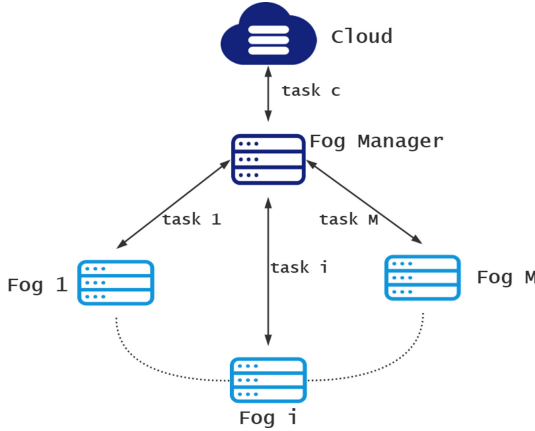


Fig. 2. Real time task scheduling framework

## 4 System Model

### 4.1 Time Delay Optimization Model Based on ARQ Protocol

The real-time task scheduling framework described in Sect. 2.2 can be regarded as a weighted undirected graph  $G(V, E)$  as shown in Fig. 3, 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,  $e_{F_M F_i}$  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  $D = \{d_1, d_2, \dots, d_j, \dots, d_C, \dots, d_m\}$ , where  $d_C$  is the subtask that is assigned to the cloud server.

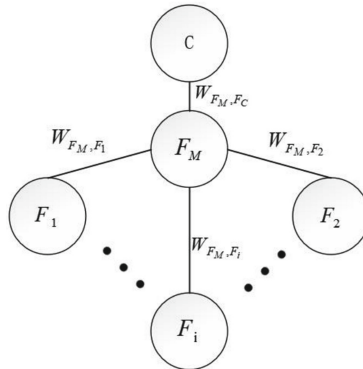


Fig. 3. Undirected graph of fog exclusive node

The total delay of  $F_i$  processing task  $d_j$  can be expressed as:

$$T_a(F_i, d_j) = \frac{d_j}{A_{F_i}} + T_t(F_i, d_j) \quad (1)$$

where  $\frac{d_j}{A_{F_i}}$  is the computing delay of the  $F_i$  processing task  $d_j$ ,  $T_t(F_i, d_j)$  is the communication delay between  $F_i$  and  $F_M$  by transfer  $d_j$ .

Similarly, the total delay of cloud server  $C$  processing task  $d_j$  is:

$$T_a(C, d_C) = \frac{d_C}{A_C} + T_t(C, d_C) \quad (2)$$

where  $\frac{d_C}{A_C}$  is the computing delay of  $C$  processing task  $d_C$ ,  $T_t(C, d_C)$  is the communication delay between  $C$  and  $F_M$  by transfer  $d_C$ .

When the network transmits task  $d_j$ ,  $d_j$  is divided into several packets. Suppose a packet length is  $L_p$  and the data transmission rate between  $F_i$  and  $F_M$  is  $v_i$ , when there is no network packet loss between communication links, the delay of a data packet's successful transmission is calculated as follows:

$$T_{S_i} = \frac{L_p}{v_i} \quad (3)$$

In the actual transmission process, there must be a packet loss rate between links. When packet loss occurs on the network, the stop-and-wait ARQ protocol is often used to retransmit data packets. The principle of the stop-and-equal ARQ protocol [13] is: after the data message is sent, the sender waits for the status report of the receiver, if the status report message is sent successfully, the subsequent data message is sent, otherwise the message is retransmitted.

Let  $T_L$  represent the time required for data packet transmission between  $F_i$  and  $F_M$  when there is a packet loss rate, ignoring the queue delay. Suppose  $n$  is the number of times of transmissions required to successfully send a data packet, and  $E_i$  is the packet loss rate between  $F_i$  and  $F_M$ . When the  $m$ -th data packet is transmitted incorrectly, the probability of its  $n$ -th transmission success is:

$$P(m = n) = (1 - E_i)E_i^{(n-1)} \forall n = 1, 2, 3 \dots \quad (4)$$

According to the TCP protocol, the waiting time before retransmission is generally:  $T_{out} = 2 \times T_{S_i}$ , the size of a single packet is  $L_p = 1448B$ , Therefore, the packet transmission time between  $F_i$  and  $F_M$  can be expressed as:

$$T_L = T_{S_i} + T_{out} \sum_{n=1}^{\infty} (n-1)P(m = n) \quad (5)$$

Combining Eq. (4), we can finally get the average transmission time of a single data packet when the packet loss rate is  $E_i$ :

$$T_L = T_{S_i} \frac{1 + E_i}{1 - E_i} \quad (6)$$

Therefore, the communication delay  $T_t(F_i, d_j)$  caused by the transmission of  $d_j$  between  $F_i$  and  $F_M$  is:

$$T_t(F_i, d_j) = \frac{d_j}{L_p} \times T_L = \frac{d_j}{v_i} \times \frac{1 + E_i}{1 - E_i} \quad (7)$$

Similarly, we can get the delay between  $F_i$  and  $C$ :

$$T_t(C, d_C) = \frac{d_C}{v_C} \times \frac{1 + E_C}{1 - E_C} \quad (8)$$

In summary, when the fog node and cloud server are not faulty, the total time delay  $T(d_j, d_C)$  spent by the task can be expressed as:

$$T(d_j, d_C) = \max\{T_a(F_i, d_j), T_a(C, d_C)\} j = 1, 2, 3, \dots \quad (9)$$

When processing real-time tasks, in order to reduce the processing delay of tasks, it is necessary to integrate the computing power of all fog nodes and the computing power of cloud server to find a task D allocation method  $D = \{d_1, d_2, \dots, d_j, \dots, d_C, \dots, d_m\}$  to minimize formula (10). Therefore, we finally get the constrained optimization problem:

$$\begin{aligned} \min T(d_j, d_C) j = 1, 2, 3, \dots \\ \text{s.t. } 0 \leq d_j, d_C \leq D \\ \sum_{j=1}^{m-1} d_j + d_C = D \end{aligned} \quad (10)$$

## 4.2 GSA-P Algorithm

In Sect. 3.1, the real-time task delay optimization problem belongs to NP problem with constraints. GSA algorithm can not effectively deal with the optimization problem with constraints, so it needs some additional skills to deal with constraints. When using genetic algorithm to do constrained optimization, the following two ways are generally used:

- 1) A penalty function is used to transform a constrained problem into an unconstrained one.
- 2) Reasonable crossover and mutation operators are designed so that only chromosomes satisfying constraint conditions can be generated in each iteration.

In this section, we discuss the first method.

### Construct Penalty Function

For the NP problem with constraints, this paper uses the method of constructing penalty function to solve it. The penalty method generally divides the solution space into feasible region and infeasible region, and the solution that does not meet the constraint conditions belongs to the infeasible region. If the current solution is close to the constraint boundary,

the penalty function value is very small, otherwise the penalty function value is very large. When the penalty method is used to solve the problem, an excellent penalty function  $\psi(x)$  is very important because it can guide the iterative results to the feasible region, and at the same time, it transforms the constrained optimization problem into unconstrained optimization problem by punishing the infeasible solution.

When the penalty function is used to punish the infeasible solution, the idea can be taken as follows:

- 1) Death penalty: For an infeasible solution, directly adjust its fitness to a size that is very easy to be eliminated, and let it be eliminated directly.
- 2) Static penalty: Static penalty will reduce the fitness value of the individual who violates the constraint in the fitness function, but the penalty coefficient will not change with the iteration of the algorithm.
- 3) Dynamic penalty: The penalty coefficient will change with the iteration times, in order to ensure convergence, the penalty coefficient will gradually become larger.
- 4) Adaptive penalty: the idea of adaptive penalty is to get feedback from the iterative process of genetic algorithm, and automatically change the penalty function according to the solution.

In order to ensure that the greater the degree of deviation of the solution from the feasible region, the greater the penalty, this paper designs an adaptive penalty function based on the concept of offset. First construct the chromosome of genetic algorithm:  $x_i = \{\delta_1, \delta_2, \dots, \delta_j, \dots, \delta_m, \delta_C\}$ , where  $\delta_j$  is scale factor, it is defined as:

$$\delta_j = \frac{d_j}{D} j = 1, 2, 3, \dots \tag{11}$$

Then, the original constraint can be converted to:

$$s.t. 0 \leq g_j(x_i) \leq 1 j = 1, 2, 3 \dots \tag{12}$$

$$h(x_i) - 1 = 0$$

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

$$g_j(x_i) = \delta_j j = 1, 2, 3 \dots \tag{13}$$

$$h(x_i) = \sum_{j=1}^{m-1} \delta_j + \delta_C \tag{14}$$

Suppose that  $g_z(x_j)$ ,  $g_f(x_j)$  is the positive and negative offset value of the solution  $x_i$  to the inequality constraint function in the problem, respectively. Then we can use (15)–(16) to express it:

$$g_z(x_i) = \max\{0, g_j(x_i) - 1\} j = 1, 2, 3 \dots \tag{15}$$



$$g_f(x_i) = \min\{0, g_j(x_i)\} j = 1, 2, 3 \dots \quad (16)$$

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

$$h_z(x_i) = \max\{0, h(x_i) - 1\} \quad (17)$$

$$h_f(x_i) = \min\{0, h(x_i) - 1\} \quad (18)$$

In the chromosome set  $\{x_i\}$ , we can find the infeasible solution set according to (15)–(18), it is denote as  $\{x_j^{reject}\}$ , thus, the positive and negative offset degrees constrained in Eq. (12) are introduced:

$$G_z(x_i) = \frac{g_z(x_i)}{\max\{g_z(x_j^{reject})\} - g_z(x_i)} \quad (19)$$

$$G_f(x_i) = g_f(x_i) \vee \frac{\quad}{\min\{g_f(x_j^{reject})\} \vee -g_f(x_i) \vee} \quad (20)$$

$$H_z(x_i) = \frac{h_z(x_i)}{\max\{h_z(x_j^{reject})\} - h_z(x_i)} \quad (21)$$

$$H_f(x_i) = h_f(x_i) \vee \frac{\quad}{\min\{h_f(x_j^{reject})\} \vee -h_f(x_i) \vee} \quad (22)$$

Based on (19)–(22) 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:

$$\phi_j(x_i) = \begin{cases} 10 \leq g_j(x_i) \leq 1 \\ e^{-G_z(x_i)} g_j(x_i) > 1 \\ e^{-G_f(x_i)} g_j(x_i) < 1 \end{cases} \quad (23)$$

$$\xi(x_i) = \begin{cases} 1h(x_i) = 1 \\ e^{-H_z(x_i)} h(x_i) > 1 \\ e^{-H_f(x_i)} h(x_i) < 1 \end{cases} \quad (24)$$

Finally, according to (23)–(24), we set the dynamic penalty function in the genetic algorithm as:

$$\psi(x_i) = \left( 2 - \frac{1}{2} \left( \xi(x_i) + \frac{1}{m} \sum_{j=1}^m \phi_j(x_i) \right) \right)^\beta \quad (25)$$

where  $\beta$  is penalty factor. If  $x_i$  is a feasible solution, the value of the penalty function  $\psi(x_i) = 1$ . When  $x_i$  is not a feasible solution,  $1 < \psi(x_i) < 2^\beta$ , so as to ensure that the penalty function  $\psi(x_i)$  can appropriately penalize the infeasible solution according to the constraints, and the greater the  $\beta$  is, the greater the penalty is.

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

$$\min T(d_j, d_c) \times \psi(x_i) \quad j = 1, 2, 3, \dots \quad (26)$$

### GSA-P algorithm Flow

In summary, the algorithm flow of GSA-P is as follows, and the flowchart is shown in Fig. 4:

- 1) Initialization: Define the scale of chromosomes as  $Y$ , the number of fog nodes as  $k$ , and the number of cloud servers as 1, so the length of each chromosome is equal to the number of fog nodes without fog manager plus the number of cloud servers, that is,  $k - 1 + 1 = k$ . Chromosomes are as defined in Sect. 3.2.1:  $x_i = \{\delta_1, \delta_2, \dots, \delta_k\}$ .
- 2) Use the penalty function as the fitness function in the genetic algorithm to calculate the fitness value of each chromosome  $x_i$ . The fitness function is expressed by:

$$f(x_i) = T(d_j, d_c) \times \psi(x_i) \quad (27)$$

- 3) Calculate the fitness value. Because there is no need to ensure the feasibility of the solution during the cross-mutation process, the chromosome can be cross-mutated by the random method.
- 4) Keep individuals with large fitness function values to ensure that the genes of outstanding individuals can be preserved.

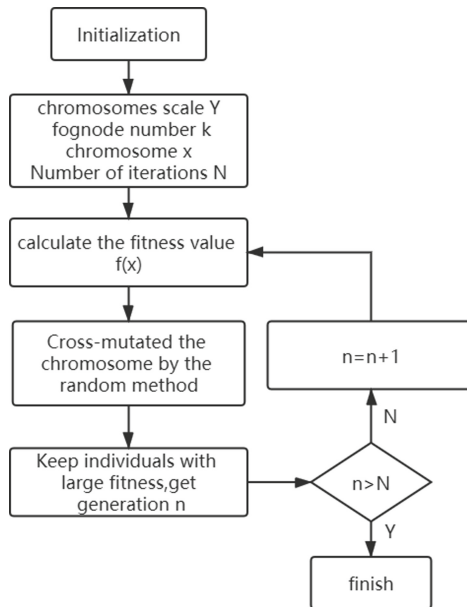


Fig. 4. GSA-P flow chart

### 4.3 GSA-R Algorithm

As mentioned in Sect. 3.2, in addition to the penalty function, a reasonable cross-mutation operator can be designed to ensure that the solution during the iteration is a feasible solution, in order to compare the GSA-P and the GSA-R.

This paper combines the GSA algorithm with the reasonable cross-mutation operator in reference [14], and obtains an algorithm GSA-R that solves the constrained optimization delay problem, which can ensure that the offspring generated after cross-mutation does not violate the constraint condition. The GSA-R algorithm flow is, and the flowchart is shown in Fig. 5:

- 1) Initialization: same as GSA-P
- 2) According to 10, calculate the fitness value of each chromosome  $x_i$ :

$$f(x_i) = T(d_j, d_C) \tag{28}$$

- 3) Select two parent individuals from individuals with appropriate fitness values, and then perform crossover operations. The relationship between offspring and parents in crossover operations is as follows:

$$x_1^s = \lambda \times x_1 + (1 - \lambda) \times x_2 \tag{29}$$

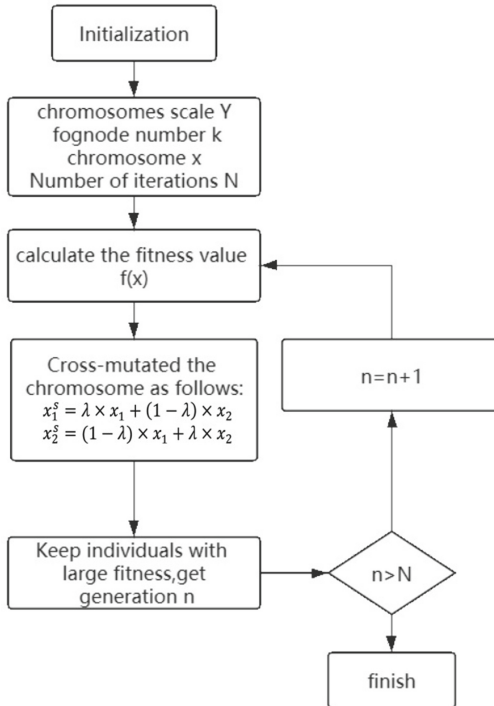


Fig. 5. GSA-R flow chart

$$x_2^s = (1 - \lambda) \times x_1 + \lambda \times x_2 \quad (30)$$

The above formula can ensure that the feasible solutions are still feasible solutions in the process of crossover.

- 4) According to the method proposed in reference [14] during mutation operation, boundary mutation and non-uniform mutation are used for feasible and infeasible solutions respectively.

## 5 Simulation Experiment

### 5.1 Simulation Environment

In the simulation, the number of fog nodes is set to 4, and a cloud server is added at the same time. The computing power of each fog node and cloud server is different. The data transmission capabilities of fog nodes, cloud servers, and management nodes are different. The packet loss rates are also independent of each other. The computing power and data transmission rate of fog nodes and cloud servers are set in reference [15]. The relevant parameters of the fog nodes and servers that handle real-time tasks are shown in Table 1.

**Table 1.** Parameters related to fog exclusive node and cloud server

Parameter type	$F_1$	$F_2$	$F_3$	$F_4$	$C$
CPU frequency (GHz)	4	4.5	5	3	10
Bandwidth (Mbps)	100	350	200	250	50
Packet loss rate	0.0232	0.014	0.0154	0.0175	0.02

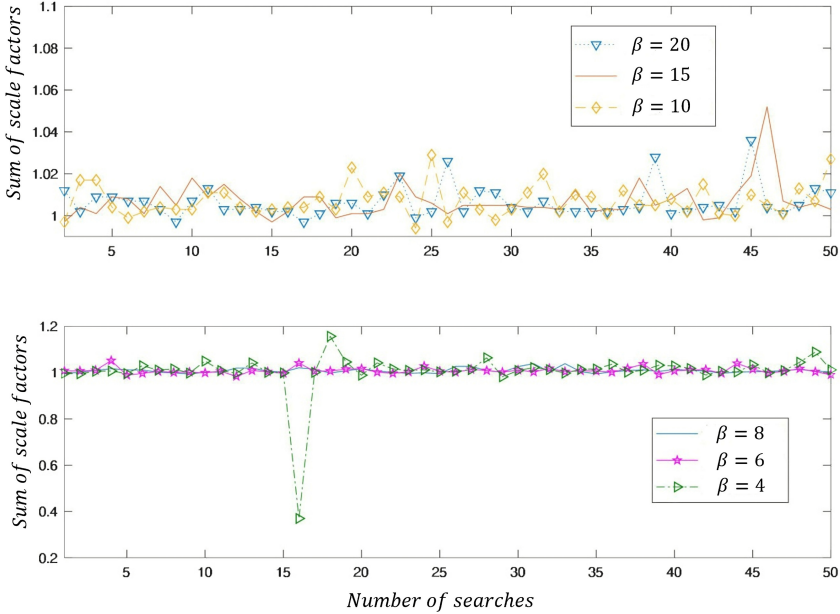
In the simulation, the real-time task data uses the industrial Internet of Things data of the real-time path planning of industrial robots [53]. The amount of data transmitted for each path planning task is 125 KB, and the CPU cycles required to calculate the path planning task is 200M. The basic parameters of the GSA-P and GSA-R algorithms: the population size is 100, and the maximum number of iterations is 3000.

### 5.2 Results and Analysis

#### Comparison of Penalty Factors

When using the penalty function, the main problem is how to choose an appropriate penalty factor  $\beta$ . In order to verify whether the solution with the lowest delay obtained by the GSA-P algorithm is a feasible solution when the penalty factor  $\beta$  is different, this paper changes the  $\beta$  Range to simulate and analyze the GSA-P algorithm.

In the simulation, the value of  $\beta$  is set to  $\{20, 15, 10, 8, 6, 4\}$ , the GSA-P algorithm iterates 3000 times to obtain the lowest delay, and then when different values of  $\beta$  are



**Fig. 6.** Simulation diagram of GSA-P when different penalty factors

calculated, the GSA-P algorithm is calculated whether the solution with the lowest time delay meets the constraint conditions, count 50 results for comparison.

It can be seen from Fig. 6 that the greater the  $\beta$  is, the greater the probability that the solution is a feasible solution.

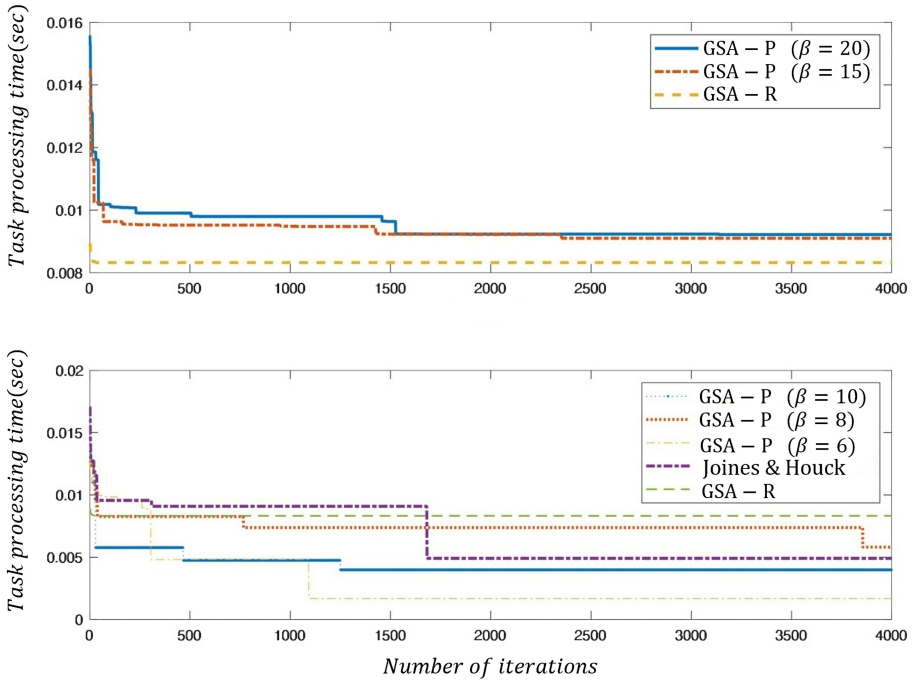
For example, when the penalty factor  $\beta = 20$ , due to the large penalty factor, the penalty for the infeasible solution is large, so the chromosome population will be more concentrated in the feasible region.

When  $\beta$  is a small value, for example,  $\beta = 4$ , the ethnic group will search in the infeasible region more actively, but because the penalty is small, the solution may be an infeasible solution.

We can also see from the line that when the value is 20, 15, 10, 8, the final solution in most cases is a feasible solution or an approximate feasible solution, but when the value is less than 6, due to the penalty is small, the result is very different from the feasible solution. Therefore, in order to ensure that the results are feasible,  $\beta$  should be set as larger than 6 as possible when solving.

**Comparison of GSA-R and GSA-P Algorithms**

According to Sect. 4.2.1, it is better that  $\beta \geq 6$ . But the value of  $\beta$  is not the bigger the better. This simulation compares GSA-P algorithm with GSA-R algorithm, and obtains a value upper limit of  $\beta$ . At the same time, the performance of GSA-P algorithm is compared with that of Joines&Houck method [16]. The simulation result is shown in Fig. 7.



**Fig. 7.** Performance simulation diagram of GSA-R and GSA-P algorithms

It can be seen from Fig. 7 that when  $\beta$  is large, searching for the optimal solution to the infeasible region is discouraged, and it is more likely to converge to a local optimal solution.

As shown in Fig. 7, when the  $\beta$  of the GSA-P is set to 20 and 15, the final optimal delay obtained by GSA-P is larger than that of the GSA-R.

When  $\beta$  is small, the population will actively search for the optimal solution in the infeasible region, so it can find the global approximate optimal solution.

When the  $\beta$  of the GSA-P is set to 10, 8, and 6, the final optimal delay obtained is smaller than that of the GSA-R. When the penalty factor is set to 6, the total task processing delay of GSA-P algorithm is about 80% lower than that of GSA-R algorithm, and about 66% lower than that of Joines&Houck method.

### Comparison of Different Computing Scenarios

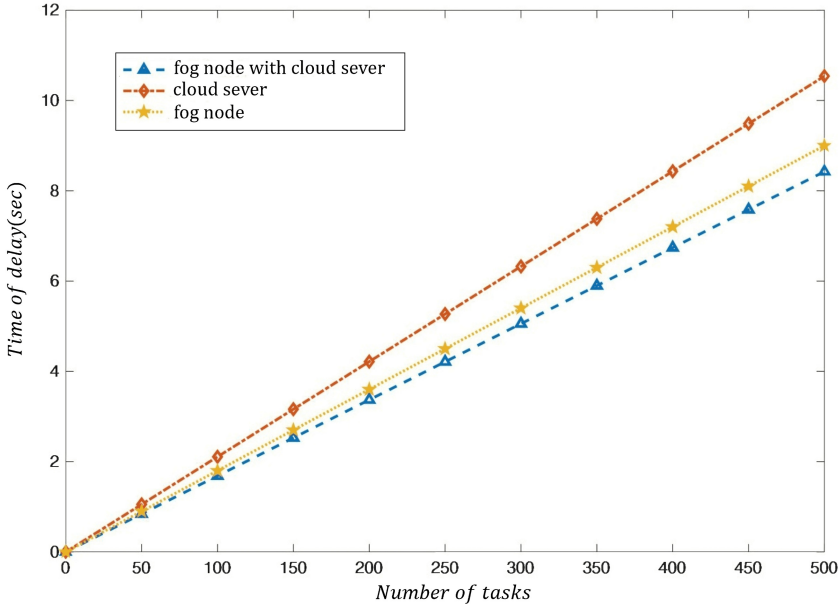
This section compares the difference between cloud and fog calculation methods and other calculation methods. In the simulation, the number of iterations is 3000, the GSA-P algorithm is used, and the penalty factor  $\beta = 6$ .

It can be seen from Fig. 8 that the calculation method of cloud-fog collaboration is better than pure cloud or pure fog computing.

Although the cloud computing capability is powerful and can reduce the calculation delay of the task, 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 latency of pure cloud computing task processing is larger, and as the number of tasks increases, the performance gap becomes more and more obvious.

Compared with the pure fog computing method, the cloud-and-fog collaborative computing method introduces a cloud server with powerful computing capabilities, and the total task delay can be reduced through reasonable task allocation.



**Fig. 8.** Performance simulation diagram under different calculation scenarios

Compared with the traditional cloud computing method and fog computing method, the method adopted in this paper can reduce the total task delay by 18% and 7% when the number of users is 500, thereby ensuring the improvement of the efficiency of the smart factory.

## 6 Conclusion

For real-time tasks in a smart factory, this paper first establishes a time delay optimization model for real-time tasks in a smart factory based on the ARQ protocol. But this model is an NP problem with constraints, ordinary genetic algorithms cannot solve it. In this paper, the GSA-P algorithm is proposed on the basis of the GSA algorithm, which transforms the constraint problem into a non-constraint problem by designing a penalty function.

Simulation experiments show that when the penalty factor of the GSA-P is 6, the delay is reduced by 80% compared with the GSA-R, and 66% lower than the Joines& Houck method algorithm. When the number of users is 500, cloud and fog collaboration computing method used in this paper reduces the task delay by 18% compared with the

traditional cloud computing method, and reduces the total task delay by 7% compared with the fog computing method.

In the future work, we will further discuss the resource scheduling problem in the intelligent factory. We will not only consider the processing of real-time tasks, but also study the processing methods of storage tasks.

**Acknowledgement.** This work was supported by the Science and Technology Commission of Shanghai Municipality under Grant 18511106500.

## References

1. Yang, Y., Luo, X., Chu, X., et al.: Fog-Enabled Intelligent IoT Systems, pp. 29–31. Springer, Cham, Switzerland (2019). <https://doi.org/10.1007/978-3-030-23185-9>
2. Chiang, M., Zhang, T.: Fog and IoT: an overview of research opportunities. *IEEE Internet Things J.* **3**(6), 854–864 (2016)
3. Mourtzis, D., Vlachou, E., Milas, N.: Industrial big data as a result of IoT adoption in manufacturing. In: ES, pp. 290–295 (2016)
4. Gazis, V., Leonardi, A., Mathioudakis, K., et al.: Components of fog computing in an industrial internet of things context. In: 2015 12th Annual IEEE International Conference on Sensing, Communication, and Networking - Workshops (SECON Workshops). IEEE (2015)
5. Fazio, M., Celesti, A., Ranjan, R., et al.: Open issues in scheduling microservices in the cloud. *IEEE Cloud Comput.* **3**(5), 81–88 (2016)
6. Chen, N., Yang, Y., Zhang, T., et al.: Fog as a service technology. *IEEE Commun. Mag.* 1–7 (2018)
7. Skarlat, O., Nardelli, M., Schulte, S., et al.: Resource provisioning for IoT services in the fog. *SOCA* **11**(4), 427–443 (2016)
8. Luxiu, Y., Juan, L., Haibo, L.: Tasks scheduling and resource allocation in fog computing based on containers for smart manufacture. *IEEE Trans. Industr. Inform.* 1–1 (2018)
9. Gedawy, H., Habak, K., Harras, K., et al.: [IEEE 2018 IEEE International Conference on Edge Computing (EDGE) - San Francisco, CA (2018.7.2–2018.7.7)] 2018 IEEE International Conference on Edge Computing (EDGE) - An Energy-Aware IoT Femtocloud System, pp. 58–65 (2018)
10. Wei, G., Vasilakos, A.V., Zheng, Y., et al.: A game-theoretic method of fair resource allocation for cloud computing services. *J. Supercomputing* **54**(2), 252–269 (2010)
11. Zhan, Z.H., Liu, X.F., Gong, Y.J., et al.: Cloud computing resource scheduling and a survey of its evolutionary approaches. *ACM Comput. Surv.* **47**(4), 1–33 (2015)
12. Miettinen, A.P., Nurminen, J.K.: Energy efficiency of mobile clients in cloud computing. In: *Usenix Conference on Hot Topics in Cloud Computing* USENIX Association (2010)
13. Li, Y., Li, J., et al.: The research on ARP protocol based authentication mechanism. In: *International Conference on Applied Mathematics, Simulation and Modelling (AMSM)* (2016)
14. Ximing, L., Haoyu, Q., Wen, L.: Genetic algorithm for solving constrained optimization problem. *Comput. Eng.* **36**(014), 147–149 (2010)
15. Xiao, M., Hassan, M.A., Wei, Q., Chen S.: Help your mobile applications with fogcomputing. In: Seattle, WA, USA : 2015 12th Annual IEEE International Conference on Sensing, Communication, and Networking - Workshops (SECON Workshops), pp. 1–6 (2015)
16. Ichalewicz, Z.: A survey of constraint handling techniques in evolutionary computation methods. In: *Proceedings of the 4th Annual Conference on Evolutionary Programming*, pp. 135–155. MIT Press, Cambridge (1995)