Resource Allocation Method of Edge-Side Server Based on Two Types of Virtual Machines in Cloud and Edge Collaborative Computing Architecture

Junfeng Man, Longqian Zhao, Cheng Peng(✉), and Qianqian Li

School of Computer Science, Hunnan University of Technology, Zhuzhou 412007, China
chengpeng@csu.edu.cn

Abstract. The process of large-scale manufacturing workshops is complex, and the traditional fixed resource allocation method will cause unbalanced load. Aiming at this problem, an edge-side server resource allocation algorithm based on cloud collaborative architecture has been designed and implemented. By defining the three-dimensional information of each IO-intensive virtual machine in the compute node, the priority of the IO-intensive virtual machine is calculated. Through analyzing the relationship between the CPU-intensive virtual machine and the host physical machine, the number of CPU cores for different tasks of the CPU-intensive virtual machine is obtained, and the hardware resources are uniformly allocated in real time according to the maximum priority list. The experimental results show that the proposed algorithm can significantly satisfy the requirements of high throughput and low latency in large manufacturing workshops, and optimize the resource allocation for actual production.

Keywords: Cloud and edge collaboration · IO-intensive · CPU-intensive · Three dimensional information · Resource allocation

1 Introduction

"Made in China 2025" is China's overall industrial development plan for the next 10 years, marking China's transition from labor-intensive production to technology-intensive, and will make a major breakthrough in cutting-edge advanced technology. In the industrial field, the Internet of Things (IoT) can actively sense and remotely control all physical devices in cloud manufacturing scenario in the existing network infrastructure [1]. By mapping the contents obtained from the physical world (real space) to the data of the information world (cyber space), it can reflect the full life cycle process of corresponding physical equipment and effectively achieve "digital twin" [2, 3].

In industrial-level application scenarios, problems such as single point failure are easy to occur. Therefore, in addition to the unified control of the cloud, certain computing power should be given to the edge terminal nodes to independently judgment and solve

H. Jiang et al. (Eds.): ICECI 2020, LNICST 368, pp. 65–85, 2021.
https://doi.org/10.1007/978-3-030-73429-9_5

problems, so as to improve the factory's capacity and prevent equipment failures. With the continuous research and exploration in the industry, as an important feature of edge computing, cloud and edge collaboration with complementary operation mode has been widely used in many scenarios such as medical, industry and finance [4, 5]. The cloud and edge collaborative computing architecture balances the computing load, reduces the hardware requirements of the edge-side servers, and makes the edge-side servers smaller, lighter, and cheaper while ensuring capacity.

The purpose of this paper is to design corresponding edge-side server resource allocation method, effectively reduce enterprise investment, and fundamentally avoid waste of resources [6]. Due to the difference of the mechanical equipment and business requirements deployed in different factories, the size and meaning of terminal data amount also very from each other. Therefore, the weight of the upload type of the terminal device is important for rational allocation of resources, optimization of operations, and optimization of cluster parameters. The characteristics of time series data generated by mechanical equipment include features such as massive equipment and measuring points, high data acquisition frequency, and large data throughput. In the process of resource allocation, we need to use the available space as a measure to ensure that the edge server can continuously store data [7]. In this background, this paper proposes a resource scheduling scheme and algorithm for cloud and edge collaborative computing architecture for edge-side server clusters in industrial scenarios.

The rest of this paper is arranged as the following sections. In Sect. 2, related works are reviewed. In Sect. 3, a cloud and edge collaborative computing architecture for industrial big data is proposed and the workflow under the architecture is also described. In Sect. 4 and Sect. 5, the system model and problem definitions are introduced, and according resource allocation algorithm is designed. In Sect. 6, the comparative experimental evaluation results are illustrated, and Sect. 7 concludes the paper, and puts forward future work.

2 Related Work

In the edge computing environment, due to the insufficiency of the infrastructure, few studies have focused on the resource allocation of edge-side server. China's 5G technology development in the recent years, focusing on the industrial and manufacturing market, has provoked the interest of the scholars in terms of resource allocation.

A traditional cloud data center is mainly composed of heterogeneous servers that carry multiple virtual machines (VM). The use of these virtual machine resources has potential irregularities and instability, which may result in unbalanced resource usage within the server, resulting in performance degradation. In order to shorten the response time of the system, Rugwiro et al. [8] proposed a task scheduling and resource allocation model based on hybrid ant colony optimization and deep reinforcement learning, based on ant colony algorithm random decision to reduce the chance of falling into local optimum, approximating optimal solution. Devarasetty et al. [9] proposed the improved optimization algorithm for resource allocation by considering the target of minimizing the deployment cost and improving the QoS performance. However, the meta-heuristic algorithm is more slower, and the obtained solution is not always the optimal solution. For

this reason, Jangiti et al. [10] proposed a set of hybrid heuristics and an ensemble heuristic to improve the solution quality. This method can effectively allocate and manage virtual resources in the cloud data center. Since cloud computing and fog calculation cannot meet the practical requirements of high response and low latency in large manufacturing workshops, Liu et al. [11] proposed to apply extreme value theory to impose probability and statistical constraints on task queue lengths in order to use a higher rate or visit a nearby server to offload the task. Liao et al. [12] proposed to use machine learning to maximize long-term throughput under the long-term constraints of energy budget and service reliability. However, Liu et al. consider that the data uploaded by an individual unit is much smaller than that collected by mechanical equipment in the actual production workshop, and the uploading protocol is just one type of protocol. This paper studies the impact of the large-scale mechanical equipment data collection on IO-intensive virtual machines [13], and the impact of large-scale manufacturing workshop access on CPU-intensive virtual machines [14].

In view of the above problems, this paper proposes an edge-side server resource allocation method based on the cloud and edge collaborative computing architecture by studying the research results of predecessors. The algorithm is designed on base of the StarlingX virtualization platform. The main contributions are as follows:

Solve the situation that the response delay of the terminal device is high due to the large amount of uploaded data. Solve the case of terminal equipment due to differences in application scenarios lead to the actual deployment difficult. Promote the intelligent and portable products of industrial enterprises, and effectively help traditional industrial enterprises to get rid of backward production capacity.

3 Architecture and Workflow

3.1 Computing Architecture

The classic solution of cloud computing architecture is to upload various types of sensor data on the facilities (vibreation, pressure, temperature, etc.) to the remote cloud server through data acquisition modual (AGV, PLC, RTU, etc.). With the help of big data analysis technologies, mathematical model will be established and the production quality, work efficiency and competitiveness of these facilities can be improved. Taking the coal mining industry as an example, the mine is generally in a remote located in remote areas and network communication is difficult. However, due to the large scale, variety, low value density, and fast update and processing requirements of coal mines, traditional cloud computing architectures cannot be adopted because traditional cloud computing architectures are prone to single-point problems and slow closed-loop response. To this end, this paper proposes a cloud and edge collaborative computing architecture [15] for industrial big data to cope with the problems of fast real-time control response and fast data processing in large manufacturing workshops.

Figure 1 illustrates the proposed architecture, it consists of a cloud component consisting of a remote server cluster and an edge end component consisting of an edge-side server cluster. To gain insight, the cloud component is mainly responsible for model training of the data collected by the terminal data collection device, and the edge component is mainly responsible for providing real-time services for the factory equipment

by acquiring the data in the model dictionary. The closed-loop response time and the production quality are improved because of the reduction of the training time. OpenStack and StarlingX are the most widely used open source cloud computing platforms and the latest distributed edge computing platforms, respectively. we use OpenStack [16] and StarlingX [17] to manage and maintain remote server clusters and edge-side server clusters. The cloud and edge collaborative computing architecture shown in Fig. 1 mainly includes a big data analysis application layer, an industrial big data platform service layer, a data resource layer, and a device sensing layer.

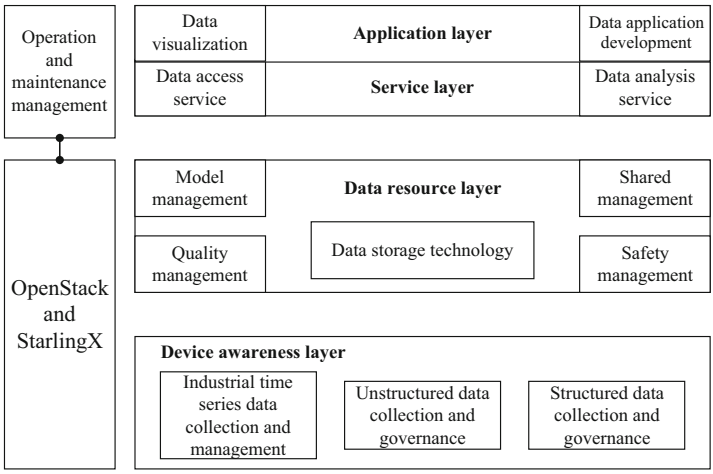

Fig. 1. Cloud and edge collaborative computing architecture.

The industrial big data analysis application layer is mainly responsible for providing corresponding services for data under different application scenarios, such as factory equipment expert knowledge base management system, factory equipment security problem reasoning and interpretation system.

The service platform of the industrial big data platform includes a task scheduling control module, a data dictionary matching module, and a data processing module. The data dictionary matching module monitors and manages the business logic, and returns the data parameters of the corresponding model information to the edge-side server to achieve fast edge-side server data processing. The data processing module specifically processes the data collected by the data acquisition module for different data scales including machine learning, deep learning and traditional data analysis methods.

The data resource layer mainly includes a core database, a business auxiliary database, a file system, etc. The core database is responsible for the cloud data collected by the storage device sensing layer device; the service auxiliary database can use the relational database such as Oracle to assist the rapid processing of the system; the file system adopts the HDFS distributed file system [18].

The device sensing layer mainly includes various sensors installed on industrial equipment, such as temperature, pressure, vibration sensors or smart factory equipment such as controllers and range finder.

3.2 Workflow

Figure 2 shows the workflow of the proposed architecture. The data acquisition device and the request action of the user are collectively referred to as a collector. The intelligent terminal simply pre-processes the information of the collector and sends it to the computing node in the cluster of the edge-side server. The IO-intensive virtual machine in the computing node is responsible for receiving and storing it in the database of the storage node.

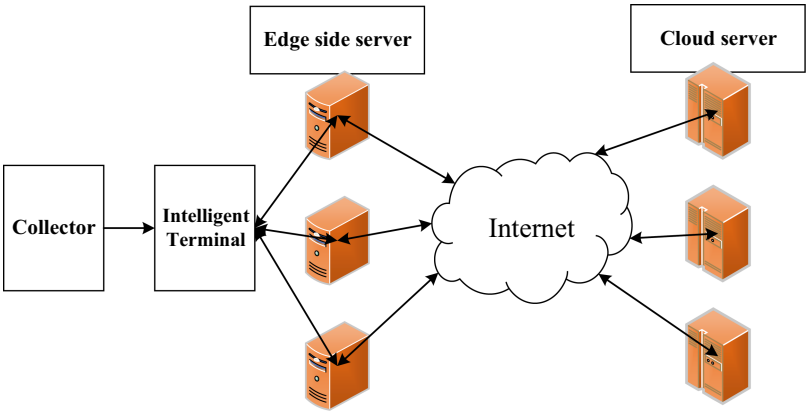

Fig. 2. Schematic diagram of the working process of the cloud and edge collaborative computing architecture.

The following is the detailed communication process:

1) On edge-side server processing
   The intelligent terminal device sends the collected data to the edge data storage module; The data processing module obtains corresponding data from the edge side data storage module according to the requirements of the user; The data processing module performs lightweight big data analysis according to the model parameters provided by the data dictionary module, and synchronizes to the edge side data dictionary module; The decision module feeds back to the intelligent terminal to perform corresponding control according to the result processed by the data processing module.
2) On remote centralized server processing: The edge-side server synchronizes the incremental data to the remote centralized data storage module; The edge-side server synchronizes the incremental data to the remote centralized data storage module; The data processing module performs heavyweight big data analysis according to the model parameters provided by the data dictionary module, and synchronizes to the remote data dictionary module; The remote data dictionary module will synchronize the data and edge data dictionary modules according to specific needs.

The edge-side server and the remote centralized server periodically analyze and mine the stored data to update the data dictionary to ensure the accuracy of the decision message.

4 Resource Allocation for IO-intensive Virtual Machines

4.1 Computing Architecture

The edge computing model of this paper consists of an edge-side server cluster and terminal devices that continuously send data. As shown in Fig. 3, an edge-side server cluster consists of two control nodes and two or more compute nodes in order to implement a highly available service architecture [19]. Among them, different data storage methods are adopted for different data sizes. As shown in Fig. 3, small-scale data is stored in the control node; large-scale data is stored in the storage node.

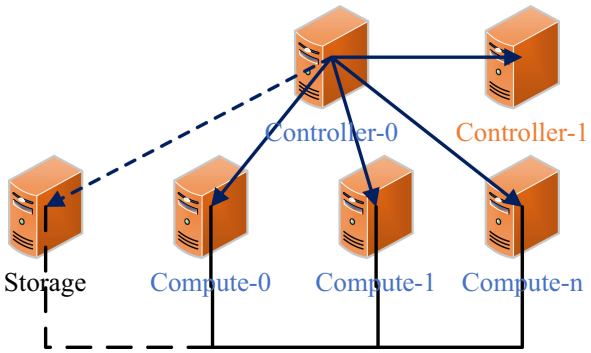

Fig. 3. Data storage mode.

In the industrial scenario, the amount of data sent by the terminal equipment deployed in each factory is different for different factory equipment and actual business needs. Therefore, it is necessary to design a design scheme for the edge-side server equipped with different factories, thereby Scientifically reduce the procurement funds of enterprises and avoid the waste of limited resources. Collaborative computing enables the processing of large-scale manufacturing workshops. Since the important parameters of most mechanical equipment do not change frequently, the mechanical characteristics of the mechanical equipment determine that most of the parameters will remain stable. Taking the piston condition of the fluid equipment as an example, the state and the outlet pressure are strongly correlated in most cases, and the outlet pressure gradually climbs from the safety initial value to the stable value, and changes stably according

to the business demand of the mechanical equipment [20]. Therefore, we obtain the dimensionality information from three aspects of IO-intensive virtual machine, namely, the weight W of the data type uploaded by the terminal device, the pre-allocated space S_{pre} corresponding to each IO-intensive virtual machine in the computing node and the actual occupied space S_{post}, and generate the maximum priority list to uniformly allocate hardware resources in real time.

4.2 Resource Allocation Model

In this section, we will obtain the priority P_i of each virtual machine through the dimension information of the three aspects of the IO-intensive virtual machine. Assume that the number of existing terminal devices is N, the data type uploaded by one terminal device is $v_i = \{v_{i1}, v_{i2}, \ldots, v_{in}\}$, and n represents the number of upload types of terminal devices. The data size corresponding to the uploaded data types is $\alpha_i = \{\alpha_{i1}, \alpha_{i2}, \ldots, \alpha_{in}\}$, the corresponding sampling time is $s_i = \{s_{i1}, s_{i2}, \ldots, s_{in}\}$, and the data collection interval is $\sigma_i = \{\sigma_{i1}, \sigma_{i2}, \ldots, \sigma_{in}\}$. The terminal device sends data to the edge-side server cluster, and the data is received by the IO-intensive virtual machine in the computing node. The number of existing virtual machines is Q, the virtual machine in the computing node is $V = \{V_1, V_2, \ldots, V_Q\}$, and the resource of a virtual machine of V_i is configured as $R_i = [x, y, z]$, x is the CPU, y is the memory, and z is the hard disk space. In order to meet the service requirements of the terminal equipment and the edge-side server cluster at the same time, we propose a dynamic resource allocation scheme based on the second-order differential heuristic algorithm. Assuming that the storage rate of the edge-side server cluster is v_i, Then the resources of the host and its internal virtual machine are satisfied:

A V_i virtual machine receives data sent from the terminal device. At the time of sampling t_i, the received data τ_i is as shown in Eq. 1:

$$\tau_i = \sum_{j=1}^{N} \sum_{i=1}^{n} \alpha_{ij} \times 1s + \varepsilon \tag{1}$$

ε represents the error generated by sampling and calculation. The average speed of a V_i virtual machine processing data is v_i, then it takes $t_i^{'}$ time to process all the data sent at time t_i, which satisfies the Eq. 2:

$$t_i^{'} = \frac{\tau_i}{v_i} \tag{2}$$

A V_i virtual machine needs $t_i^{'}$,$t_{i+1}^{'}$, and $t_{i+2}^{'}$ time to store data in the database. The second-order difference of the data received by the virtual machine is used to obtain the data upload rate increment Δv as shown in Eq. 3:

$$\Delta v = \tau_{i+2} - 2\tau_{i+1} + \tau_i \tag{3}$$

It can be known from Eq. 3 that a virtual machine of V_i has six states in the time $t_i \sim t_i + 2\sigma_i$ period, and is represented by a set $status = \{1, 2, 3, 4, 5, 6\}$, where 1 is a no data state; 2 is a reduced state; 3 is a steady state; 4 is an increased state; 5 is an early warning State; 6 is the excess state. When $\Delta v \approx -\tau_i$, it indicates that the virtual machine of the $t_i \sim t_i + 2\sigma_i$ period has not received the data sent by the terminal device. When $-\tau_i < \Delta v < 0$, it indicates that the data uploaded by the terminal device is decreasing during this time period. When $\Delta v \approx 0$, it indicates that the data uploaded by the terminal device remains basically unchanged during this time period. When $\Delta v < M_0$, it indicates that the data uploaded by the terminal device is increasing during the time period. When $M_0 \leq \Delta v < M_1$, it indicates that the data uploaded by the terminal device has exceeded the warning value M_0 during the time period. When $\Delta v \geq M_1$, it indicates that the data uploaded by the terminal device has exceeded the actual physical storage capacity of the storage node during the time period, and the rate of data uploading of the terminal device needs to be solved by increasing the hard disk space or optimizing the data storage solution. M_1 is the actual storage size of the edge-side server storage space.

It is assumed that by collecting a large number of state sets $num_i \subseteq status$ of a V_i virtual machine in a working time period in advance, ξ_i indicates that the previous state of states 4, 5, and 6 is 4, 5, and 6. Then the probability π_i of the previous state of 4, 5, and 6 is 4, 5, and 6 as shown in Eq. 4:

$$\pi_i = \frac{\xi_i}{size(num_i) - 1} \tag{4}$$

The ith virtual machine in the compute node of the edge-side server soft allocates a pre-storage space of S^i_{pre}, and the data size of a V_i virtual machine that has been stored to the storage node at time t_i is S^i_{post}. When the amount of data received by a V_i virtual machine is τ_i at a time, the priority p_i of the data stored by the virtual machine of the V_i is as shown in Eq. 5:

$$p_i = \frac{S^i_{pre} - S^i_{post}}{\tau_i} \tag{5}$$

In the industrial application scenario, due to the particularity of the industrial equipment itself, taking the aircraft as an example, there are a large number of key mechanical components such as gears, shafts, bearings, blades, etc. in the power transmission system of the aircraft, and the analysis of the collected data through deep learning is obtained. The weight W [21] of parameter, combined with Eq. 4 and Eq. 5, gives the priority P_i of a V_i virtual machine in response to stored data as shown in Eq. 6:

$$P_i = \frac{\sum w}{||W||} \times \frac{1}{p_i} \times \pi_i \tag{6}$$

w represents the weight of each parameter. N represents the collection of w. It can be known from Eq. 3 that assuming that there are terminal devices transmitting data to the edge-side server, the data type uploaded by one terminal device is $v_i = \{v_{i1}, v_{i2}, \ldots, v_{in}\}$. n indicates the number of terminal device upload types. When the P_i value is larger, the more the virtual machine needs to be allocated to the more memory space in time, thereby increasing the speed of data storage.

4.3 Priority List and Resource Allocation

The edge-side IO-intensive virtual machine in the cloud and edge collaborative computing architecture proposed in this paper is mainly responsible for receiving terminal data. The mathematical model of virtual machine IO performance and hardware resources is established by second-order differential heuristic algorithm, and finally the IO-intensive virtual machine adaptive configuration is realized. Suppose a server at the edge end of V_i receives k tasks $List = \{\tau_1, \tau_2, \ldots, \tau_k\}$ continuously, set the current task waiting queue as $T = \{\tau_i\}$, the weight value of characteristic parameters as W, and the storage space S_{pre} and S_{post} corresponding to IO-intensive virtual machines in the server at the edge end, and the priority is P_i. The priority generation algorithm is shown in Algorithm 1:

Algorithm 1 Priority generation algorithm

Require: T, W, S_{pre}, S_{post}

Ensure: P_i

1: function PriorityGenerate(T, W, S_{pre}, S_{post})

2: while $\tau_i \in T$ and $i \in \{1,2,\ldots,k\}$ do

3: if Data allocation table can view τ_i then

4: $p_i = \frac{s^i_{pre} - s^i_{post}}{\tau_i}$

5: $P_i = \frac{\sum w}{||W||} \times \frac{1}{p_i} \times \pi_i$

6: end if

7: end while

8: end function

Each IO-intensive virtual machine in the edge-side server will have a priority P_i. The control node Controller will obtain this priority P_i and establish the maximum priority queue P. The resource configuration of the IO-intensive virtual machine is updated correspondingly by the change of the priority queue P. The virtual machine resource allocation algorithm is as shown in Algorithm 2:

Algorithm 2 IO-intensive virtual machine resource allocation algorithm

```
Require：P
Ensure：Max_heap_PList
1: function IOResourceAllocation(P)
2:   InitalGenerate(PList)
3:   InitalGenerate(Max_heap_PList)
4:   while P_i ∈ P and i ∈ {1,2, ... , k} do
5:     if Data allocation table can view P_i then
6:       size[PList] = size[PList]+1
7:       PList[size[PList]] = 0
8:       if P_i<PList[i] then
9:         Erro
10:      end if
11:      PList[i] = P_i
12:      while i>1 && PList[PARENT(i)]< PList[i] do
13:        exchange PList[i]<-> PList[PARENT(i)]
14:        i= PARENT(i)
15:      end while
16:    end if
17:  end while
18:  while P_i ∈ P and i ∈ {1,2, ... , k} do
19:    if size[PList] < 1 then
20:      Error
21:    end if
22:    max_P_i = PList[1]
23:    PList[1] = PList[size[PList]]
24:    size[PList] = size[PList]-1
25:    adjust_max_size(PList,1)
26:    Max_heap_PList[i] = max_P_i
27:  end while
28: end function
```

The Algorithm 2–3 lines initialize the biggest priority queue. Lines 4–7 insert priority P_i into the set in turn. Lines 8–17 new P_i insertion at the end of the priority pair. Then adjust the queue from the rear parent node. Lines 18–27 copy the last element in the *PList* queue to the first location and delete the last node. Put the first element of the *PList* queue into the *Max_heap_PList* queue and delete an element of the *PList* queue. And then adjust the queue.

The time complexity of the algorithm is mainly the process of initializing the *PList* queue and the process of re-establishing the *PList* queue after each selection of the maximum number. The time complexity of initializing the *PList* queue is *O(n)*. The time complexity of changing the queue element to reconstruct the *PList* queue is *O(nlogn)*, and the space complexity of the algorithm is *O(1)*. Based on the original algorithm, the algorithm is applicable to the problem of IO-intensive virtual machine resource allocation caused by large-scale terminal equipment sending data in industrial scenarios, and maximally responds to resource requests of IO-intensive VM.

By using the algorithm of this paper, it is assumed that there are N terminal devices transmitting data to Q virtual machines. The resource allocator unit time average processing capacity is μ, and the average number of requests per unit time is λ [22]. If we need to allocate i virtual machines and can process m requests at the same time, you can get the resource allocator utilization ρ according to multi-queue system, as shown in Eq. 7:

$$\rho = \frac{i}{m\mu}, m \leq i \leq \lambda \leq Q \tag{7}$$

The relationship between the resource allocator utilization ratio ρ and the response time R is as shown in Eq. 8:

$$R \approx \frac{1}{\mu} \times \frac{1}{1 - \rho^m} \tag{8}$$

5 Resource Allocation for CPU-intensive Virtual Machines

The virtual machines in the edge-side server are mainly divided into IO-intensive and CPU-intensive virtual machines. IO-intensive virtual machines are mainly responsible for receiving and storing data. CPU-intensive virtual machines are mainly responsible for lightweight edge calculation of data. At the same time, the IO-intensive virtual machine uses the peer-to-peer mode to receive the data sent by the terminal device, as shown in Fig. 4. Decoupling storage from computing reduces interference between services and helps services become more efficient.

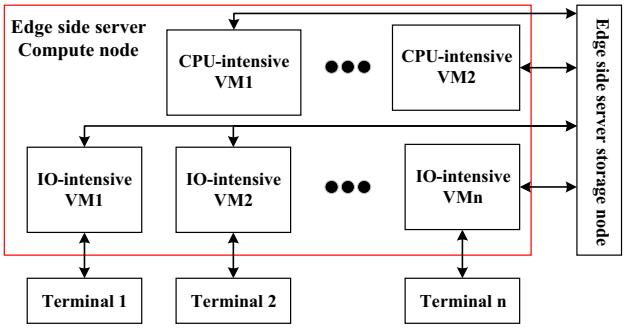

Fig. 4. Storage and computation decoupling, peer-to-peer mode receiving data.

5.1 Resource Allocation Model

In this section we will handle lightweight edge computing tasks with CPU-intensive virtual machines. Suppose the number of existing tasks is b, and the task type $e = \{e_1, e_2, \ldots, e_f\}$ on an edge-side server cluster, f represents the number of task

types on an edge-side server cluster. The terminal device sends data to the edge-side server cluster, and the CPU-intensive virtual machine in the compute node processes the data. There are o CPU-intensive virtual machines. CPU-intensive virtual machine $V' = \{V'_1, V'_2, \ldots, V'_o\}$ in the compute node. The resource configuration of a V'_i virtual machine is $R_i = [x, y, z]$, x is CPU, y is memory, and z is hard disk space. In order to meet the service requirements of terminal equipment and edge side clusters at the same time, we propose a optimal virtual machine performance (OVMP) resource allocation method for CPU-intensive virtual machines. Assuming that T tasks are run in the CPU-intensive virtual machine V'_i in the edge-side server cluster, the execution time of the CPU-intensive application [23] is as shown in Eq. 9:

$$Time = \begin{cases} time, & T \leq C \\ time + \frac{time}{C}(T - C), & T > C \end{cases} \tag{9}$$

Where *time* is the time when a single task is executed, C is the number of CPU cores, T is the number of simultaneous executions of the application, and *Time* is the time required for all applications to execute. In industrial scenarios, artificial intelligence algorithms such as machine learning and deep learning are commonly used to implement edge-side data processing, while Hadoop and Spark are useful tools to allow storage and process big data in a distributed environment across computer clusters using a simple programming model [24, 25]. Assuming that the minimum execution time of the CPU-intensive virtual machine in the pseudo-distributed computing environment is R_p, and the minimum execution time in the distributed parallel computing environment is R_n, the acceleration ratio *sp* obtained by the distributed computing system is as shown in Eq. 10:

$$sp = \frac{R_p}{R_n} \tag{10}$$

Due to the wide variety of tasks in industrial big data, such as filtering for noise data, cleaning and denoising, modeling integration and multi-scale classification, and tasks such as correlation analysis of manufacturing parameters such as process parameters and equipment status parameters [26]. This article uses *time_com* to indicate the completion time of task. Considering the communication delay of different tasks, such as cloud computing communication delay is about 100 ms, small data center communication delay is about 10 to 40 ms, router communication delay is about 5 ms, communication delay between terminal devices It is about 1 to 2 ms [27]. The industrial big data task requires completion time as shown in Eq. 11:

$$Req\_Time^{'} \geq time\_com + time\_cor \tag{11}$$

Where $Req\_Time^{'}$ indicates the required completion time of the task, *time_cor* indicates the communication delay of the task. The virtual machine creates an impact on the performance of the physical machine. When the number of virtual machines is less than the number of CPU cores, the CPU resources occupied by system users increase as the number of virtual machines increases. The execution time of the application is not affected by the number of virtual machines. When the number of virtual machines

exceeds the number of host CPU cores C_{phy}, as the number of virtual machines increases, the performance gradually decreases, and the impact rate is θ [28]. Thus, we get the relationship between the number of virtual machines o and the impact rate θ, as shown in Eq. 12:

$$\theta = \frac{o}{C_{phy}} \tag{12}$$

Assuming b tasks are respectively executed in o CPU-intensive virtual machines of the same configuration and the corresponding execution time $Time = \{Time_1, Time_2, ..., Time_b\}$ is obtained. Combined with Eq. 9 and Eq. 10, the processing time $Time' = \left\{Time_1^{'}, Time_2^{'}, ..., Time_b^{'}\right\}$ is obtained in a distributed parallel environment. For different business needs, the execution time $Time'$ must be less than the required completion time Req_Time', as shown in Eq. 13:

$$\Pi_i = \begin{cases} 0, \ Time_i \le Req\_Time_i^{'} \\ 1, \ Time_i > Req\_Time_i^{'} \end{cases} \tag{13}$$

Suppose there are d in the number of 1 in Π, the task set $b' = \left\{b_1^{'}, b_2^{'}, ..., b_d^{'}\right\}$ is not completed on time, its corresponding sequence $seq = \left\{seq\left(b_1^{'}\right), seq\left(b_2^{'}\right), ..., seq\left(b_d^{'}\right)\right\}$, the corresponding execution time $Time_{unfinished}^{'} = \left\{Time_{seq\left(b_1^{'}\right)}^{'}, Time_{seq\left(b_2^{'}\right)}^{'}, ..., Time_{seq\left(b_d^{'}\right)}^{'}\right\}$, and the corresponding required completion time $Req\_Time_{unfinished}^{'} = \left\{Req\_Time_{seq\left(b_1^{'}\right)}^{'}, Req\_Time_{seq\left(b_2^{'}\right)}^{'}, ..., Req\_Time_{seq\left(b_d^{'}\right)}^{'}\right\}$. Combine Eq. 9 and Eq. 13 to obtain a CPU-intensive virtual machine resource allocation method, as shown in Eq. 14:

$$C_i^{add} = \begin{cases} 1, & \frac{Time_{seq\left(b_i^{'}\right)}^{'} \times T}{Req\_Time_{seq\left(b_i^{'}\right)}^{'}} \le 1 \\ \frac{Time_{seq\left(b_i^{'}\right)}^{'} \times T}{Req\_Time_{seq\left(b_i^{'}\right)}^{'}} - \frac{1}{2}, & \frac{Time_{seq\left(b_i^{'}\right)}^{'} \times T}{Req\_Time_{seq\left(b_i^{'}\right)}^{'}} > 1 \end{cases}, \quad i = 1, 2, ..., d \tag{14}$$

Through extensive research, CPU-intensive applications mainly consume CPU resources. Combined with Eqs. 12 and 14, it is necessary to increase the number of corresponding C^{add} cores by a maximum priority list of no less than C and no more than d CPU-intensive virtual machines.

5.2 Resource Allocation

The CPU-intensive virtual machine in the edge-side server first allocates the same virtual machine resource. Hadoop sends the task to the corresponding CPU-intensive virtual

machine according to the default scheduler. The impact on the resource configuration of the CPU-intensive virtual machine is updated accordingly. The virtual machine resource allocation algorithm is as shown in Algorithm 3:

Algorithm3 CPU-intensive virtual machine resource allocation algorithm

Require: *T, C, Req_Time*
Ensure: C^{add}
1: function CPUResourceAllocation(*T, C, Req_Time*)
2: InitialGenerate(C^{add})
3: temp = 0
4: while $Time_i \in Time\ and$ i $\in \{1,2,\ldots,b\}$ do
5: $Time'[i] = \frac{Time[i]}{sp}$
6: if $Time'_i \le Req\_Time'_i$ then
7: $\Pi_i = 0$
8: else
9: $\Pi_i = 1$
10: temp = temp + 1
11: end if
12: end while
13: while $\Pi_i \in \Pi\ and$ i $\in \{1,2,\ldots,b\}$ do
14: if $\Pi_i = 1$ then
15: if $\frac{Time'_{seq(b'_i)} \times T}{Req\_Time'_{seq(b'_i)}} \le 1$
16: $C^{add}[i] = 1$
17: else
18: $C^{add}[i] = \left\lfloor \frac{Time'_{seq(b'_i)} \times T}{Req\_Time'_{seq(b'_i)}} - \frac{1}{2} \right\rceil$
19: end if
20: end while
21: while $C_i^{add} \in C^{add}$ and i $\in \{1,2,\ldots,C\}$ do
22: $C[i] = C[i] + C^{add}[i]$
23: end while
24: end function

Line 2 adds the list of CPU cores for preliminary testing. Lines 4–12 calculate the execution time of each task under a distributed parallel computing system. Lines 13–20 calculate the number of allocated CPU cores for d of the b tasks that do not meet the required completion time. Lines 21–23 assign the corresponding number of C^{add} virtual machine cores to C virtual machines.

6 Experiment and the Results

6.1 Experimental Configuration

In this section, we will get the performance of the modified algorithm through the StarlingX virtualization platform [29].

We created a virtual environment as an edge-side server through the StarlingX virtualization platform, including one Controller node, four IO-intensive virtual machines, and three CPU-intensive virtual machines. For the specific configuration of the above examples, see Table 1, Table 2 and Table 3.

Table 1. Physical host related information.

Parameter	BOGOMIPS	Memory	Disk	Bandwidth	CA	OS
host	3791.22	128 GB	1TB	1000 Mb/s	X86	Linux

Table 2. Information about IO-intensive virtual machines.

Parameter	BOGOMIPS	Memory	Disk	Bandwidth	CA	OS
Controller	3791.22	32 GB	70G	1000 Mb/s	X86	Linux
IO-1	3791.22	12 GB	70G	1000 Mb/s	X86	Linux
IO-2	3791.22	12 GB	70G	1000 Mb/s	X86	Linux
IO-3	3791.22	12 GB	70G	1000 Mb/s	X86	Linux
IO-4	3791.22	12 GB	70G	1000 Mb/s	X86	Linux

Table 3. Information about CPU-intensive virtual machines.

Parameter	BOGOMIPS	Memory	Disk	Bandwidth	CA	OS
CPU-1	3791.22	3	70G	1000 Mb/s	X86	Linux
CPU-2	3791.22	3	70G	1000 Mb/s	X86	Linux
CPU-3	3791.22	3	70G	1000 Mb/s	X86	Linux

6.2 Experimental Results

We run the client service of the resource allocator on four IO-intensive virtual machines, which are responsible for monitoring the IO performance of the virtual machine. The priority is obtained by calculation and sent to the Controller node. The server service of the resource allocator running on the Controller node is responsible for collecting the priority of the IO-intensive virtual machine, and responding to the resource request of

Table 4. Average response time for IO-intensive virtual machines.

Number of IO-intensive virtual machines	Maximum average response time
1	0.017 s
2	0.016 s
3	0.017 s
4	0.014 s

the IO-intensive virtual machine according to the maximum priority list. The average response time of the algorithm is 0.01 s by simulation. As shown in Table 4:

It can be seen from Eq. 5 that the theoretical relationship between the utilization ratio ρ and the response time of the IO-intensive virtual machine is as shown in Fig. 5. It can be seen that when the utilization rate is gradually increased to 1, the response time grows slowly. Explain that our resource utilization rate has little effect on response time when it is close to 90%.

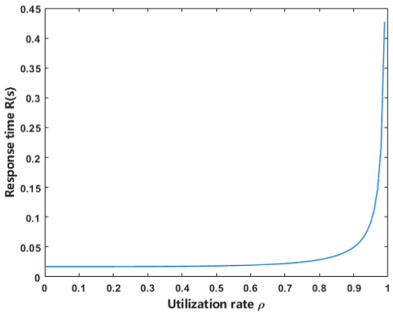

Fig. 5. Relationship between resource utilization and response time under theoretical conditions.

We compare the algorithm with the genetic algorithm and obtain the performance data of the two algorithms in the experimental scene. We can see that the algorithm guarantees the allocation of resources with the optimal response of the IO-intensive virtual machine with a sampling width of 4. However, due to the uncertainty of random number, the genetic algorithm is not reasonable enough to allocate resources. Results are shown in Fig. 6.

We use the dd command to evaluate the hard disk read/write speed of the virtual machine. By setting the memory size of the IO-intensive virtual machine, the bandwidth of the corresponding hard disk data storage in the case of 1 GB data write is obtained, as shown in Table 5. It can be seen from the table that the impact of memory on data storage speed is relatively large. Combined with Fig. 6, we can get the impact of three algorithms on the data written by the hard disk. Using the algorithm IO-intensive virtual machine requires 417.8814 s to store all data to the hard disk, and the genetic algorithm needs 436.9942 s. The specific results are shown in Fig. 7.

Table 5. The effect of different memory on hard disk write data.

Memory (GB)	Bandwidth (MB/s)	Time to write data (s)
16	266	4.0308
20	315	3.4082
24	352	3.05264

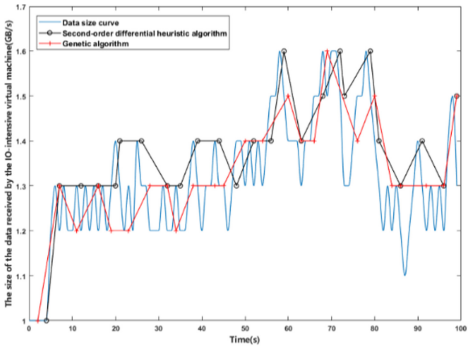

Fig. 6. Resource allocation graph for two algorithms.

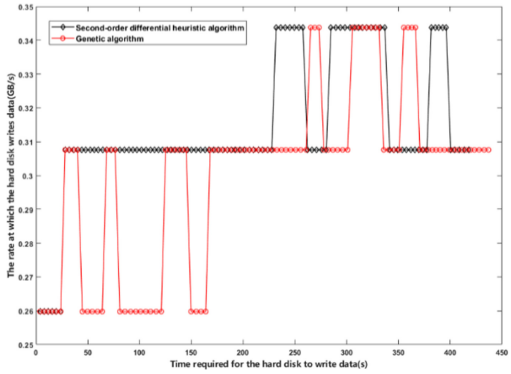

Fig. 7. The effect of two algorithms on the hard disk write data.

We run Hadoop and Spark distributed storage and computing frameworks on three CPU-intensive virtual machines. Common CPU-intensive applications are: WordCount, Sort, TeraSort, RandomWriter [30]. For the above-mentioned types of CPU-intensive applications, we generate different scale test data sets through HiBench and store them in the HDFS file system and submit them to Spark to perform the corresponding tasks [31]. The specific parameters of the tasks are shown in Table 6:

Table 6. CPU-intensive application test case parameters.

Tasks	Task type	Task data set size	Request completion time
Task-1	WordCount	108 MB	20 s
Task-2	WordCount	108 MB	20 s
Task-3	Sort	216 MB	23 s
Task-4	Sort	216 MB	23 s
Task-5	TeraSort	432 MB	25 s
Task-6	TeraSort	432 MB	25 s

According to the initial resource configuration in Table 3, we can get the maximum execution time of the application in the three cases of pseudo-distributed conditions, fully distributed conditions and multi-task co-competition, as shown in Table 7:

Table 7. Application execution schedule in different execution environments.

Tasks	Single task pseudo distributed	Single task fully distributed	Multi-task fully distributed
Task-1	24 s	15 s	24 s
Task-2	24 s	15 s	24 s
Task-3	24 s	15 s	24 s
Task-4	23 s	15 s	23 s
Task-5	27 s	19 s	45 s
Task-6	27 s	19 s	49 s

The OVMP algorithm of this paper is compared with the genetic algorithm and the performance data of the two algorithms under the experimental scene is obtained. We can see that the algorithm does not require a large amount of actual historical data to obtain similar calculation time with other algorithms. The algorithm used in this paper starts from the impact of virtual machine on physical machine performance and the impact of different configured virtual machines on task execution time. The optimal resource allocation method for CPU-intensive virtual machine is given. The experimental results as shown in Table 8 and Fig. 8:

Table 8. CPU core number allocation scheme corresponding to three algorithms.

Parameter	OVMP algorithm	Genetic algorithm
CPU-1	5	5
CPU-2	5	5
CPU-3	6	6

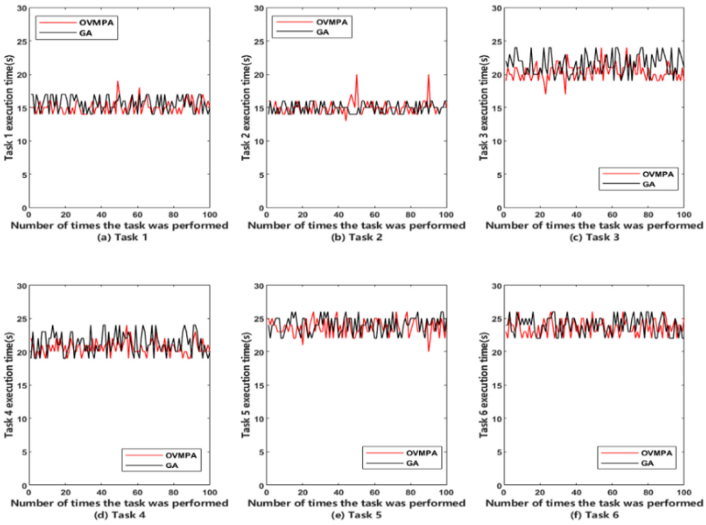

Fig. 8. Algorithm performance comparison chart.

The time complexity for the CPU-intensive resource allocation method used in this paper is $O(n)$, while the time complexity of the genetic algorithm is $O(n^2)$. By adjusting the number of tasks, according to the time complexity, the data of the resource allocation time under the theory can be obtained. By comparison, the algorithm used in this paper is fast and suitable for the actual test lathe inspection environment.

7 Conclusions

In this paper, the cloud and edge collaborative computing architecture and the edge side cluster resource allocation method were proposed. For the IO-intensive virtual machine, the priority of each IO-intensive virtual machine is given in combination with the second-order difference method, and finally the resource allocation algorithm of the dynamic adaptive IO-intensive virtual machine is realized. For the CPU-intensive virtual machine, combined with the application scenario of the test lathe in the actual production environment, the dynamic adaptive CPU-intensive virtual machine resource allocation algorithm is realized. Compared with other heuristic algorithms, the simulation results

show that the edge-side server store data and calculation speed of this algorithm were significantly improved, when the number of factory terminal equipment and test tasks are large, the resource consumption is relatively small.

Acknowledgement. This paper is supported by Natural Science Foundation of China (No. 61871432, No. 61702178), The Natural Science Foundation of Hunan Province (No. 2020JJ4275, 2020JJ6086, 2019JJ60008, 2018JJ4063).

References

1. Li, L.: China's manufacturing locus in 2025: with a comparison of "Made-in-China 2025" and "Industry 4.0". Technol. Forecast. Soc. Change **135**, 66–74 (2018)
2. Zhou, J.: Toward New-generation intelligent manufacturing. Engineering **4**(1), 28–47 (2018)
3. Stark, R.: Development and operation of Digital Twins for technical systems and services. CIRP Ann. **68**(1), 129–132 (2019)
4. Shen, W.: Potential applications of 5G communication technologies in collaborative intelligent manufacturing. IET Collab. Intell. Manuf. **1**(4), 109–116 (2019)
5. Xu, L.D.: Big data for cyber physical systems in industry 4.0: a survey. Enterp. Inf. Syst. **13**(2), 148–169 (2019)
6. Jena, M.C.: Application of Industry 4.0 to enhance sustainable manufacturing. Environ. Prog. Sustain. Energy **39**(1), 13360 (2020)
7. Song, T.: Server consolidation energy-saving algorithm based on resource reservation and resource allocation strategy. IEEE Access **7**, 171452–171460 (2019)
8. Rugwiro, U.: Task scheduling and resource allocation based on ant-colony optimization and deep reinforcement learning. J. Internet Technol. **20**(5), 1463–1475 (2019)
9. Devarasetty, P.: Genetic algorithm for quality of service based resource allocation in cloud computing. Evol. Intel. **16**(4), 1–7 (2019). https://doi.org/10.1007/s12065-019-00233-6
10. Jangiti, S.: Scalable hybrid and ensemble heuristics for economic virtual resource allocation in cloud and fog cyber-physical systems. J. Intell. Fuzzy Syst. **36**(5), 4519–4529 (2019)
11. Liu, C.F.: Dynamic task offloading and resource allocation for ultra-reliable low-latency edge computing. IEEE Trans. Commun. **67**(6), 4132–4150 (2019)
12. Liao, H.: Learning-based context-aware resource allocation for edge-computing-empowered industrial IoT. IEEE Internet Things J. **7**(5), 4260–4277 (2019)
13. Hu, A., Xiang, L., Xu, S., Lin, J.: Frequency loss and recovery in rolling bearing fault detection. Chin. J. Mech. Eng. **32**(1), 1–12 (2019). https://doi.org/10.1186/s10033-019-0349-3
14. Shen, G.: A study of the condition monitoring of large mechanical equipment based on a health management theory for mechanical systems. Insight Nondestr. Test. Condition Monit. **61**(8), 448–457 (2019)
15. Zhang, J.X.: Cloud collaborative computing framework for a service robot based on ROS. Comput. Syst. Appl. **25**(9), 85–91 (2016)
16. Merlino, G.: Enabling workload engineering in edge, fog, and cloud computing through OpenStack-based middleware. ACM Trans. Internet Technol. **19**(2), 28–30 (2019)
17. An overview of the StarlingX project. https://www.starlingx.io/learn/. Accessed 15 May 2020
18. Zhu, J.: Research on data mining of electric power system based on Hadoop cloud computing platform. Int. J. Comput. Appl. **41**(4), 289–295 (2019)
19. Yamato, Y.: Fast and reliable restoration method of virtual resources on OpenStack. IEEE Trans. Cloud Comput. **6**(2), 572–576 (2018)

20. Yi, C.: Quaternion singular spectrum analysis using convex optimization and its application to fault diagnosis of rolling bearing. Measurement **103**(6), 321–323 (2017)
21. Chen, F., Fu, Z., Zhen, L.: Thermal power generation fault diagnosis and prediction model based on deep learning and multimedia systems. Multimedia Tools Appl. **78**(4), 4673–4692 (2018). https://doi.org/10.1007/s11042-018-6601-5
22. Huang, Y.: M/M/n/m queuing model under nonpreemptive limited-priority. Chin. J. Appl. Probab. Stat. **34**(4), 364–368 (2018)
23. Peng, J., Chen, J., Kong, S.: Resource optimization strategy for CPU intensive applications in cloud computing environment. In: IEEE 3rd International Conference on Cyber Security and Cloud Computing 2016, CSCloud, Beijing, vol. 10134, pp. 124–128. IEEE (2016)
24. Hu, N.: Power equipment status information parallel fault diagnosis of based on MapReduce. J. Comput. Methods Sci. Eng. **19**(1), 165–170 (2019)
25. Zhi, Y.: Balance resource allocation for spark jobs based on prediction of the optimal resource. Tsinghua Sci. Technol. **25**(04), 487–497 (2020)
26. Zhang, J.: Big data driven intelligent manufacturing. China Mech. Eng. **30**(2), 127–133 (2019)
27. StarlingX Enhancements for Edge Networking, [EB/OL] (2018). https://www.openstack.org/videos/summits/berlin-2018/starlingx-enhancements-for-edge-networknet. Accessed 15 May 2020
28. Guo, W., Kuang, P., Jiang, Y., Xu, X., Tian, W.: SAVE: self-adaptive consolidation of virtual machines for energy efficiency of CPU-intensive applications in the cloud. J. Supercomput. **75**(11), 7076–7100 (2019). https://doi.org/10.1007/s11227-019-02927-1
29. Merlino, G.: Enabling workload engineering in edge, fog, and cloud computing through OpenStack-based middleware. ACM Trans. Internet Technol. (TOIT) **19**(2), 1–22 (2019)
30. Al-Tarazi, M., Chang, J.M.: Network-aware energy saving multi-objective optimization in virtualized data centers. Clust. Comput. **22**(2), 635–647 (2018). https://doi.org/10.1007/s10586-018-2869-5
31. Cao, Y.: Communication optimisation for intermediate data of MapReduce computing model. Int. J. Comput. Sci. Eng. **21**(2), 226–233 (2020)