Computer Systems Science & Engineering
DOI:10.32604/csse.2022.024021
Article

Tech Science Press

Flexible Task Scheduling Based on Edge Computing and Cloud Collaboration

Suzhen Wang[1,*], Wenli Wang[1], Zhiting Jia[1] and Chaoyi Pang[2]

[1]Hebei University of Economics and Business, Shijiazhuang, 050061, China
[2]The Australian e-Health Research Centre, ICT Centre, CSIRO, Australia
*Corresponding Author: Suzhen Wang. Email: wsuz@163.com
Received: 30 September 2021; Accepted: 26 November 2021

Abstract: With the rapid development and popularization of 5G and the Internet of Things, a number of new applications have emerged, such as driverless cars. Most of these applications are time-delay sensitive, and some deficiencies were found during data processing through the cloud centric architecture. The data generated by terminals at the edge of the network is an urgent problem to be solved at present. In 5 g environments, edge computing can better meet the needs of low delay and wide connection applications, and support the fast request of terminal users. However, edge computing only has the edge layer computing advantage, and it is difficult to achieve global resource scheduling and configuration, which may lead to the problems of low resource utilization rate, long task processing delay and unbalanced system load, so as to lead to affect the service quality of users. To solve this problem, this paper studies task scheduling and resource collaboration based on a Cloud-Edge-Terminal collaborative architecture, proposes a genetic simulated annealing fusion algorithm, called GSA-EDGE, to achieve task scheduling and resource allocation, and designs a series of experiments to verify the effectiveness of the GSA-EDGE algorithm. The experimental results show that the proposed method can reduce the time delay of task processing compared with the local task processing method and the task average allocation method.

Keywords: Edge computing; “cloud-edge-terminal” framework; task scheduling and resource allocation

1 Introduction

Most of the emerging application scenarios based on 5G technology are either computationally intensive or data intensive, presenting “wide link” and requiring “low latency”. One of the goals of edge computing [1,2] is to process data close to the network edge generated by data, so as to meet the needs of users for low delay, low energy consumption, security and privacy.

At present, there is no uniform definition of edge computing in academic and industrial circles. The European Telecommunications Standards Institute (ETSI) and the Edge Computing Consortium (ECC) have proposed different concepts of edge computing. These concepts have reached a consensus: complete data processing near the data generation terminal and provide services nearby. In [2], edge computing is divided into three stages, and the basic concept of edge computing is function caching. Reference [3]

considers edge computing as an enabling technology to meet the needs of the industry in terms of agile connectivity, security and privacy. Literature [4,5] summarizes the scenario and architecture of edge computing, and describes future development trends as well as potential problems. The paper [6–8] summarizes the task offloading strategies for mobile edge computing. Reference [9] studies service requests in mobile edge computing and proposes GAMEC algorithm. Reference [10] abstracts the task unloading problem as a nonlinear programming problem by using the concept of software defined network. Reference [11,12] proposes a scheduling strategy based on task priority. Reference [13–18] looks at the edge computing applications in security, AI and such as.

The related research on edge computing, often fails to consider the collaborative participation of cloud center. Most of the existing scheduling schemes are based on single degree quantity, such as delay, and energy consumption. As a result, it is imperative to study the theory and method of task scheduling based on a Cloud-Edge-Terminal three-tier collaborative framework.

This paper first describes the concept and architecture of edge computing, then introduces the three-tier architecture of "Cloud-Edge-Terminal", and presents a task scheduling strategy based on genetic simulated annealing algorithm. Finally, the model is constructed and the simulation experiment is designed to confirm the effectiveness of the proposed task scheduling strategy.

2 "Cloud-Edge-Terminal" Framework Design

Although the definitions given by different organizations may differ, but they express a common point of view, that is, to process data nearby at the edge of the network. The mainstream definitions of edge computing is listed in Tab. 1.

Table 1: Concept of edge computing

Organization	Definition
ISO/IEC JTC1/ SC38	Edge computing is a form of distributed computing that places the main processing and data storage at the edge nodes of the network.
ECC	Edge computing refers to an open platform that integrates core capabilities of network, computing, storage, and applications on the edge of the network close to the source of things or data, and provides edge intelligent services nearby to meet the needs of industry digitization in agile connection, real-time business, data optimization, application intelligence, Key requirements for security and privacy protection.
ETSI	Provide IT service environment and computing capabilities at the edge of mobile networks, and emphasize proximity to mobile users to reduce network operation and service delivery delays and improve user experience.

2.1 Connotation of Edge Computing

The mobility of edge computing is the primary concern, but edge computing should not be limited to mobile edge computing (MEC). It is worth mentioning that some scholars "MEC" multi access edge computing.

2.2 Framework of Edge Computing

The edge computing industry alliance has proposed a feasible architecture for edge computing, which is widely used in existing research. A simplified schematic diagram is shown in Fig. 1.

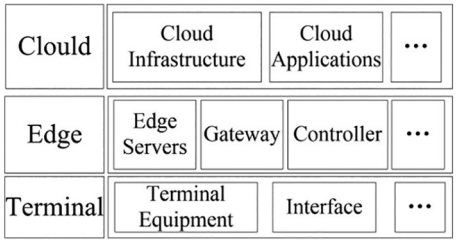

Figure 1: Architecture of edge computing

The terminal layer is located at the edge of the network, which is composed of various terminal devices and generates a large amount of heterogeneous data. Limited by the resources of terminal devices, most terminal devices cannot complete those tasks with a long duration and a large amount of computation.

The edge layer acts as a bridge between the cloud layer and the terminal layer, which is closer to the terminal devices at the edge of the network. The edge layer completes the processing of data generated by terminal devices and supports terminal layer services nearby.

Cloud is rich in computing, storage and other resources, and is good at dealing with long-term, large amounts of data and complex computing. However, with the rich types and number of terminals, the data generated shows an explosive growth trend. Due to network transmission costs, cloud computing has some deficiencies in supporting emerging low latency applications.

2.3 "Cloud-Edge-Terminal" Framework

Edge computing and cloud computing are complementary technologies, and each has its advantages. The emergence of edge computing extends cloud services to the edge of the network. It will become a new development trend of cloud computing technology to process the data at the edge nearby and cooperate in the cloud center. Reference [19] propose that the value of edge computing and cloud computing can be magnified, and they also provide reference framework and application scenarios of "edge cloud" collaboration.

Most of the existing edge computing researches focus on the edge layer, and form the "Edge-Terminal" binary architecture with the terminal devices, which has some limitations in the global and integrity [6,7,11,12]. In this paper, a Cloud-Edge-Terminal collaborative framework is proposed. The cloud center is introduced into the traditional "Edge-Terminal" binary architecture, and the edge computing and cloud computing are organically combined to construct the "Cloud-Edge-Terminal" collaborative framework. This paper proposes a framework for bringing the advantages of cloud computing resources to the network edge layer, and at the same time synergizing the local advantages of edge computing and cloud computing. The diagram of Cloud-Edge-Terminal framework is shown in Fig. 2.

This paper focuses on the process of the terminal user unloading the task to the edge server. The edge computing server is closer to the data generation side, but its resources are still limited compared with those of the cloud center. Therefore, it is necessary to find a suitable edge computing server to allow the terminal user to offload the task.

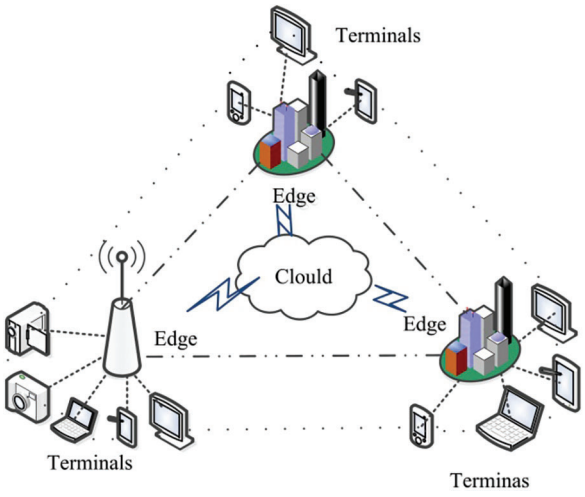

Figure 2: "Cloud-Edge-Terminal" collaboration diagram

Considering that in practice, the priorities for offloading requests of different user tasks is different, and the resources provided by the server at the edge layer are often different at different times. At the same time, considering the user demand and the load of the edge server can improve the efficiency of task offloading, as well as reduce service time and energy consumption. Taking into account the possibility of further task segmentation, the task is further divided into more fine-grained tasks, so as to explore the parallelism between subtasks.

This paper mainly focuses on resource collaboration in cloud edge collaboration. The task unloading of terminal devices first selects the edge server cluster closer to each other. The cloud is responsible for the scheduling of task unloading, making the unloading decision, processing the unloading request of the terminal, matching the best edge server for the unloading task, and participating in the collaborative computing when the edge device resources are insufficient. The process is shown in Fig. 3.

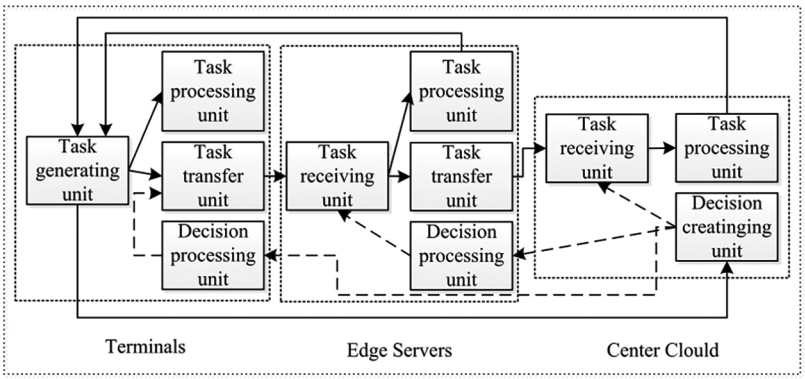

Figure 3: "Cloud-Edge-Terminal" collaborative processing

3 Model Building

In order to further study, this section will abstract the edge computing and cloud collaboration framework model, and formalize the problem.

3.1 Device Model

Definition 1 Center Cloud

The center cloud is represented as a 3-tuple cloud (m, c, b).

(1) "m" represents the memory resources that the center cloud can provide;
(2) "c" represents the CPU resources that the center cloud can provide;
(3) "b" represents the bandwidth resources that the center cloud can provide.

Definition 2 Edge Server

The edge server is represented as a 4-tuple E_i (m, c, b, t_0).

(1) "m" represents the memory resources that the edge server can provide;
(2) "c" represents the CPU resources that the edge server can provide;
(3) "b" represents the bandwidth resources that the edge server can provide;
(4) "t_0" represents the estimated processing time of tasks on the server.

Definition 3 Terminal

The terminal is represented as a 2-tuple Ter_i (m, c).

(1) "m" represents the memory resources that the terminal can provide;
(2) "c" represents the CPU resources that the terminal can provide.

3.2 Task Model

Task division aims to divide task J into multiple subtasks ω_i(I = 1, 2, …). Then according to whether unloading is allowed or not, it can be divided into two categories: allowing unloading and not allowing unloading. In this paper, directed acyclic graphs are used represent the task after partition. D (V, E):V represents subtasks and E is the directed edge between nodes, as shown in the Fig. 4.

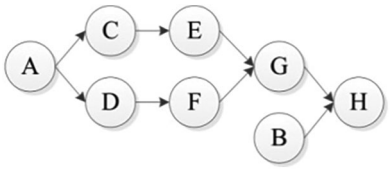

Figure 4: Task division

Definition 4 Task

The task is represented as a 4-tuple $Task_i$ (m, c, t_{max}, *Mob*).

(1) "m" represents the memory requirement of this task;
(2) "c" represents the CPU requirement of this task;
(3) "t_{max}"represents the latest completion time of the task;
(4) "Mob" represents the terminal that generated the task.

Definition 5 Subtask

The task is represented as a 6-tuple $Subtask_j$ (m, c, t_{max}, $task_i$, d_{in}, d_{out}).

(1) "m" represents the memory requirement of this task;
(2) "c" represents the CPU requirement of this task;
(3) "t_{max}"represents the latest completion time of the task;

(4) "$task_i$" represents the task before the subtask is partitioned;
(5) "d_{in}" represents the predecessor subtask of the subtask;
(6) "d_{out}" represents the successor subtask of the subtask.

Subtasks should meet the two constraints:

(1) The starting subtask, that is, the task without degree must be executed locally;
(2) The subtasks except (1) are allowed to be offloaded to the edge server or center cloud for execution.

According to the constraint rules, subtasks A and B in Fig. 5 must be executed locally at the terminal, and tasks C through H are allowed to be unloaded. The specific execution location depends on the remaining resources of each server and the network conditions.

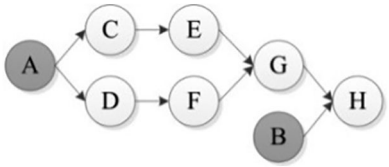

Figure 5: Task classification

3.3 Time and Energy Consumption Model

Definition 6 Delay

The total task processing delay is defined as follows:

$$T_{all} = t_c + t_0 + t_w \tag{1}$$

where t_c represents task $task_i$ unload to edge computing server E_j transmission delay and total return delay of calculation results; t_0 represents the processing delay on the edge server; and t_w is the waiting delay of unloading process.

Definition 7 Energy Consumption

The total task processing energy consumption is defined as follows:

$$\mathrm{En} = \sum_{i=1}^{n} \mathrm{P_i} * \mathrm{T_i} \tag{2}$$

where P_i represents the power of device i; T_i represents the continuous running time of the device i.

3.4 Description of Problem and Nondeterministic Polynomial (NP) Analysis

3.4.1 Problem Description

The goal of the task scheduling method is to minimize the task delay under the premise of ensuring the user's delay requirements. Therefore, the problem is modeled as the problem of minimizing the delay under the constraint conditions to ensure the user's agile connectivity experience. This section refines the constraints for this problem.

Definition 8 Execution Cost

The weighted sum of time and energy consumption is defined as the task execution cost.

$$\mathrm{Cost\_Func} = \omega_1 \mathrm{T} + \omega_2 \mathrm{E}, \quad \omega_1 + \omega_2 = 1 \tag{3}$$

where, ω_1 and ω_2 represent the time weighting factor and energy consumption weighting factor respectively,

indicating the sensitivity of the system to this factor. The larger the value, the higher the sensitivity. When $\omega_2 = 0$, the task execution cost is the task delay. Then, the problem is transformed into an optimization problem with constraints.

$$\text{s.t.} \begin{cases} T_{all} \leq T_{max} \\ Resource_i \leq Edge_i J \end{cases} \tag{4}$$

3.4.2 NP Analysis of the Problem

When the number of tasks is n and the number of edge servers is m, for a task scheduling problem in an edge computing and cloud collaborative environment, its time complexity is $O(m^n)$. If m significantly larger, the above problems cannot be solved in polynomial time. So task scheduling problem becomes an NP problem.

The task scheduling problem in the Cloud-Edge-Terminal collaborative framework can be abstracted as n tasks assigned to m targets, which can be regarded as a knapsack problem (KP) [20]. It represents the 0–1 knapsack problem (0–1 KP), which is a combinatorial optimization problem, and has been proved to be a NP complete (NPC) problem.

Because the NPC problem cannot obtain exact solution in polynomial time, the performance of existing task scheduling methods still needs to be improved. In this paper, a genetic algorithm and simulated annealing algorithm are used to optimize the transformed combinatorial optimization problem.

3.5 Algorithm Design

This paper uses an optimized genetic algorithm to solve the problem. The flowchart of the algorithm is shown in Fig. 6. Genetic algorithm uses roulette selection, and fitness function selects the reciprocal of time. The mutation selection is random point crossover.

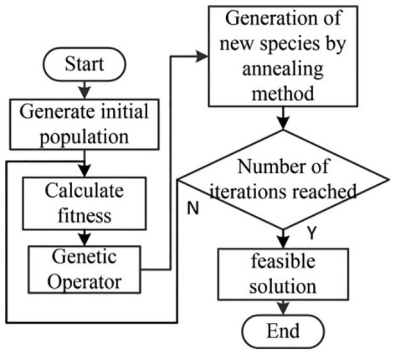

Figure 6: Algorithm flowchart

(1) Coding guidelines

The goal of the task scheduling method is to match the most suitable destination server for the task, and Dests is used to represent the final unloading method. For the sake of simplicity, the local processing task code is 0, the cloud server number is 1, and the edge servers are coded from 2 in turn. The target server uses binary encoding, and the final result is the final scheduling method.

DestS = [0000 0001 0002 0003 …]

(2) Selection

In this paper, roulette algorithm is used as the generation method, and the reciprocal of the system cost is defined as the fitness function.

$$\text{Fitness} = \frac{1}{\text{Cost\_Func}} \tag{5}$$

(3) Crossover

The process of cross operation and chromosome exchange to produce new offspring is one of the processes of population producing new offspring. The common crossover operations include single point crossover, multi-point crossover and extended crossover.

(4) Mutation

Single point mutation and multiple point mutation are common methods for mutation operation and chromosome individual gene mutation. Single point mutation is used in this paper.

(5) Annealing process

In order to reduce the probability of genetic algorithm falling into local optimal solution in the early stage, a simulated annealing factor is introduced.

Disturbance probability is introduced in Eq. (6).

$$p = \exp(-\Delta\text{Fitness}) \tag{6}$$

If P/(K*T) > rand, accept the suboptimal population. The "rand" is a rand number, where "K" is constant, and "T" is the current temperature. The temperature drop follows the rule Eq. (8), "σ" annealing coefficient.

$$\text{rand} \in (0,\ 1) \tag{7}$$

$$T(t + 1) = \sigma T(t) \tag{8}$$

4 Experimental Design and Analysis of Result

4.1 Experimental Design

In this section, the above is tested and verified. The experiment is based on the Python programming language, and the simulation platform parameter information is shown in Tab. 2.

Table 2: Experimental platform information

Parameter	Value
Operating system	Windows 10 Professional
Processor	AMD Ryzen 5 3600 3.59 GHz
Memory	16GB
Version	Python 3.7

The experiment adopts the current mainstream method, of using Python for data simulation, and generating experimental parameters through random methods [11,12]. The parameter information is shown in Tabs. 3 and 4.

Table 3: Resource parameters

Parameter	Value range
Number of edge servers	10–100
Number of terminals	100–1000
Edge servers CPU resources	10^2–10^3
Edge servers memory resources	10^9–10^{10}
Terminals CPU resources	1–1.5
CPU required by the tasks	1–1.5
Memory required by the tasks	10^6–10^7
Terminal-Edge transmission bandwidth	10^8
Cloud-Edge transmission bandwidth	10^8–$9*10^8$
Terminal-Cloud transmission bandwidth	10^9

Table 4: Energy consumption parameters

Parameter	Value range
Terminal	0.10–0.25
Edge servers	100–150
Cloud	1500–2500

This paper verifies for experimental sensitive applications. The experimental design consists comparing task localization processing with task random offloading processing in order to prove the necessity of task offloading; then verifying the task scheduling method proposed in this paper, and finally comparing the algorithm proposed in this paper with the task average distribution algorithm.

The equipment energy consumption parameters are shown in Tab. 4.

4.2 Analysis of Results

In order to determine the effectiveness of the algorithm, we first compared the performance of the improved genetic algorithm. The optimal scheduling results were analyzed under the condition that the number of edge server nodes is 10 and the number of cloud center nodes is 1. The algorithm parameters are shown in Tab. 5.

Table 5: Algorithm parameters

Parameter	Value
Number of iterations	500
Crossover probability	0.7
Mutation probability	0.1
Initial temperature	1000
σ	0.98

The experimental results are shown in Tab. 6.

Table 6: Experimental results

Number of tasks	Algorithm	Results(s)
100	GA	0.077009
	GSA-EDGE	0.071019
200	GA	0.1145118
	GSA-EDGE	0.112412
300	GA	0.142701
	GSA-EDGE	0.13808
400	GA	0.290745
	GSA-EDGE	0.28089
500	GA	0.47644
	GSA-EDGE	0.423638
600	GA	0.498386
	GSA-EDGE	0.426664
700	GA	0.689837
	GSA-EDGE	0.647761

The experimental results show that under the conditions of this experiment setting, as the number of tasks increased from 100 to 700, the improved algorithm becomes more effective.

Next the relationship between the optimal iteration results of the algorithm and the number of iterations were compared. The experiment was carried out under the conditions that the number of tasks was 600, the number of edge computing servers was 10, and the number of cloud centers is 1. To demonstrate the optimization performance of the algorithm, the experiment was repeated 50 times, and the comparison between the average optimal result and the optimal result was conducted. The results are shown in Tab. 7.

The experimental results show that the improved genetic algorithm has a smaller adaptation value and better searching effect under the same number of iterations.

Table 7: Analysis of algorithm performance

Number of iterations	Algorithm	Optimal results(s)	First appears	Average results(s)	Average number of iterations
100	GA	0.53149605	90	0. 55634206	88.2
	GSA-EDGE	**0.52202637**	96	0.540611662	90.2
200	GA	0.49449435	194	0.51066	182.5
	GSA-EDGE	**0.48609051**	198	0.505746252	185.6
300	GA	0.48294193	238	0.491247565	273.25
	GSA-EDGE	**0.47822559**	293	0.48864317	277.2
400	GA	0.47208516	370	0.481051	374.5
	GSA-EDGE	**0.46378412**	374	0.477597682	374.4
500	GA	0.46664622	469	0.47015579	476.2
	GSA-EDGE	**0.46281636**	482	0.468687904	482
600	GA	0.46284699	567	0.467606	554.75
	GSA-EDGE	**0.45722705**	587	0.46553027	561.2
700	GA	0.45745619	663	0.462613	641.25
	GSA-EDGE	**0.45045954**	680	0.460182298	649
800	GA	0.45542874	780	0.460006	782.5
	GSA-EDGE	**0.44958811**	789	0.45809669	762
900	GA	0.45398229	850	0.458569	829
	GSA-EDGE	**0.44845254**	880	0.45694739	799.2
1000	GA	0.44791635	913	0.45156852	865
	GSA-EDGE	**0.44664402**	930	0.45434687	834

Fig. 7 shows the fitness of the improved genetic algorithm when the number of tasks was 600 and the number of iterations was 500.

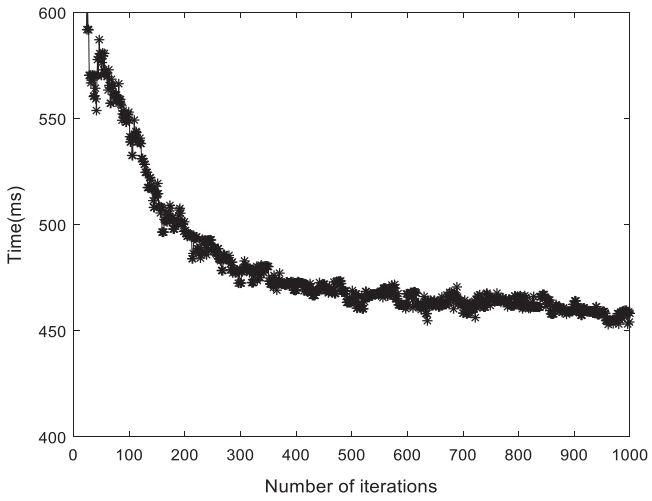

Figure 7: Algorithm fitness

Because genetic algorithms have relatively few parameters, they are easy to implement. Therefore, for to analyze the influence of crossover and variation factors on the parameters of the improved genetic algorithm in term of the optimization results, the experiment was carried out using 500 iterations and 600 tasks. The results are shown in Tab. 8

Table 8: Analysis of algorithm performance

Mutation probability	0.1	0.2	0.3	0.4	0.5	0.6
Fitness	0.46281636	0.49259758	0.474520742	0.463950786	0.476446722	0.475106816

Next, the unload necessity proof experiment was carried out to compare the unload processing with local task processing and unscheduled task processing. The experimental results are shown in Fig. 8.

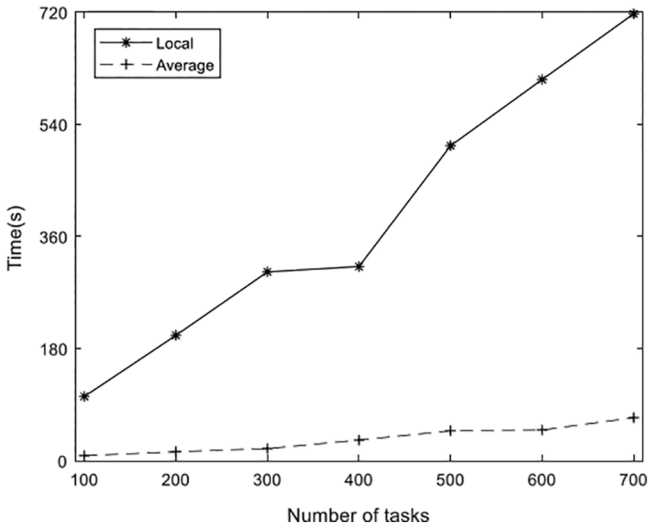

Figure 8: Total time of task local processing and random offloading

Fig. 8 shows the comparison of task localization processing and the total time delay of task randomly unloading task processing under the "Cloud-Edge-Terminal" framework proposed in this paper. Analysis of the experimental results, show that random task offloading can reduce the total task processing time by 91.4%. Therefore, edge computing for task offloading can reduce the total task duration.

Then, for the number of tasks from 100 to 700, under the condition of the number of edge servers is 10 and the number of cloud centers is 1.

The delay weighting factor $\omega_1 = 1$ was selected in experiment, and the longest processing time of the device was regarded as the duration of the method. The longest processing time of the device was selected as the processing delay, the experiment was verified, and the task local terminal processing, average offloading and the offloading method designed in this paper were compared. The experimental results shown in Fig. 9, confirm that the offloading algorithm proposed in this paper can reduce the task processing time by 43.35% on average compared with the average offloading algorithm, and reduce the task processing time by 77.67% with task localization.

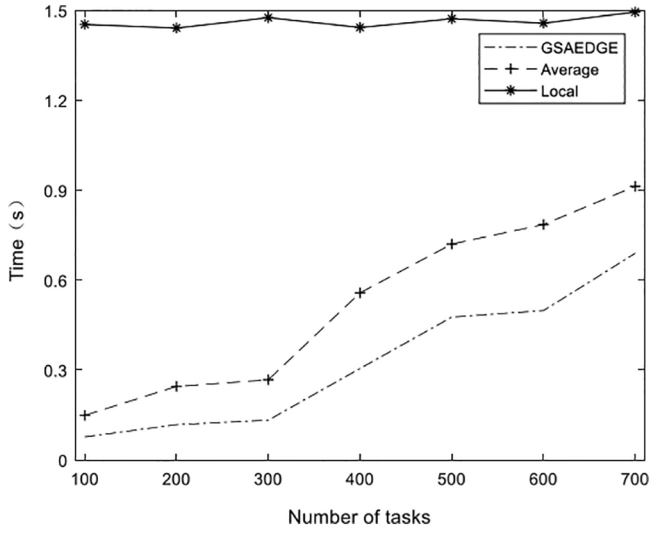

Figure 9: Comparison of three algorithms in terms of task processing delay

Task local processing means that the tasks generated by the terminal are processed on the terminal, and the task processing time is the genetic algorithm-based scheduling algorithm parameters discussed in this paper, as shown in Tab. 5.

When the number of tasks is 500, verify the relationship between the number of tasks and the number of edge servers. The experiment selects the number of edge servers [10, 100], and the specific results are shown in Fig. 10. The experimental results show that when the number of edge servers is small, the average computational delay of the algorithm proposed in this paper is decreased significantly.

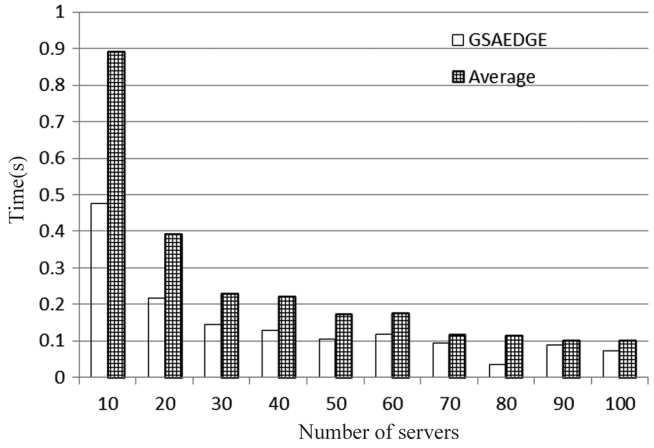

Figure 10: Relationship between calculation delay and the number of servers

Finally, the experiment vertifies the relationship between the task delay and the number of algorithm iterations in an environment where the number of tasks is 100, 200, and 500, and the number of edge servers is 50. The experimental results show that the task processing time of the 100 to 300 generations of iterations is significantly reduced. Starting with the 300th generation, the time has been reduced slowly. The experimental results are shown in Fig. 11.

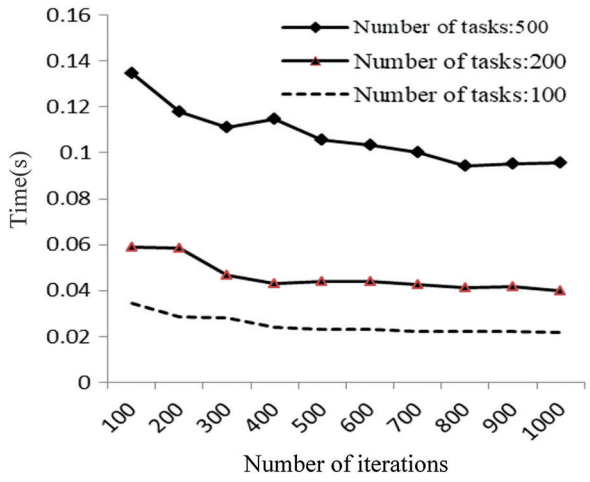

Figure 11: Relationship between calculation delay and number of iterations

5 Conclusion

In this paper, the concept, framework and application scenarios of edge computing are defined, and a Cloud-Edge- Terminal collaborative framework is proposed. Based on this, a task scheduling strategy based on time delay and energy consumption is proposed, which shows an improvement compared with task localization processing, task random offloading and comparison.

Going forward, we will consider optimizing the experimental environment and building an edge computing environment using big data processing platforms such as Spark and Hadoop. Furthermore, because multi cloud access is one of the technology trends in edge computing "cloud-edge" collaborative computing in a multi-cloud environment will be another work direction.

Acknowledgement: Thanks to the project team Shanshan Geng and Zhanfeng Zhang for their contributions to this article.

Funding Statement: This paper is partially supported by the Social Science Foundation of Hebei Province (No. HB19JL007), and the Education technology Foundation of the Ministry of Education (No. 2017A01020), and the Natural Science Foundation of Hebei Province (F2021207005).

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] W. Shi and S. Dustdar, "The promise of edge computing," *Computer*, vol. 49, no. 5, pp. 78–81, 2016.

[2] W. Shi, J. Cao, Q. Zhang, Y. Li and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[3] C. Ding, J. Cao, L. Yang and S. Wang, "Edge computing: Applications, state-of-the-art and challenges," *ZTE Technology Journal*, vol. 25, no. 3, pp. 2–7, 2019.

[4] Z. Zhao, F. Liu, Z. Cai and N. Xiao, "Edge computing: Platforms, applications and challenges," *Journal of Computer Research and Development*, vol. 55, no. 2, pp. 327–337, 2018.

[5] W. Shi, X. Zhang, Y. Wang and Q. Zhang, "Edge computing: State-of-the-art and future directions," *Journal of Computer Research and Development*, vol. 56, no. 1, pp. 69–89, 2019.

[6] S. Dong, H. Li, Y. Qu and L. Hu, "Survey of research on computation offloading strategy in mobile edge computing," *Computer Science*, vol. 46, no. 11, pp. 32–40, 2019.

[7] R. Xie, X. Lian, Q. Jia, T. Huang and Y. Liu, "Survey on computation offloading in mobile edge computing," *Journal on Communications*, vol. 39, no. 11, pp. 138–155, 2018.

[8] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1628–1656, 2017.

[9] H. Wu, S. Deng, W. Li, M. Fu, J. Yin *et al.,* "Service selection for composition in mobile edge computing systems," in *IEEE 2018 IEEE Int. Conf. on Web Services (ICWS)*, San Francisco, CA, USA, PP. pp. 355–358, 2018.

[10] M. Chen and Y. Hao, "Task offloading for mobile edge computing in software defined ultra-dense network," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 3, pp. 587–597, 2018.

[11] S. Dong, J. Wu, H. Li, L. Hu and Y. Qu, "Task scheduling policy for mobile edge computing with user priority," *Application Research of Computers*, vol. 37, no. 9, pp. 2701–2705, 2020.

[12] S. Dong, J. Wu, H. Li, Y. Qu and L. Hu. "Resource allocation strategy for mobile edge computing of multi-priority tasks," *Computer Engineering*, vol. 46, no. 3, pp. 18–23, 2020.

[13] A. Gumaei, M. Al-Rakhami and H. AlSalman, "Dl-har: Deep learning-based human activity recognition framework for edge computing," *Computers, Materials & Continua*, vol. 65, no. 2, pp. 1033–1057, 2020.

[14] C. Qian, X. Li, N. Sun and Y. Tian, "Data security defense and algorithm for edge computing based on mean field game," *Journal of Cyber Security*, vol. 2, no. 2, pp. 97–106, 2020.

[15] J. Qi, Q. Song and J. Feng, "A novel edge computing based area navigation scheme," *Computers, Materials & Continua*, vol. 65, no. 3, pp. 2385–2396, 2020.

[16] H. Wang, J. Yong, Q. Liu and A. Yang, "A novel GLS consensus algorithm for alliance chain in edge computing environment," *Computers, Materials & Continua*, vol. 65, no. 1, pp. 963–976, 2020.

[17] T. Yang, X. Shi, Y. Li, B. Huang, H. Xie *et al.,* "Workload allocation based on user mobility in mobile edge computing," *Journal on Big Data*, vol. 2, no. 3, pp. 105–111, 2020.

[18] L. Jiang and Z. Fu, "Privacy-preserving genetic algorithm outsourcing in cloud computing," *Journal of Cyber Security*, vol. 2, no. 1, pp. 49–61, 2020.

[19] Edge computing and cloud synergy white paper, 2018. [Online]. Available: http://www.ecconsortium.org/Uploads/file/20190221/1550718911180625.pdf.

[20] G. B. DANTZIG, "Discrete variable extremum problems," *Operations Research*, vol. 5, no. 2, pp. 266–277, 1957.