ISSN:2790-1688 DOI: 10.56028/aetr.4.1.12.2023

# Edge-side task scheduling: Auction mechanism and genetic algorithm based methods

Peng Ren [1, a], Ruiyou Zhang [1, b], Zhiyou Li [1, *]

[1]SCollege of Information Science and Engineering, Northeastern University , Shenyang, China.

[a] 2000829@stu.neu.edu.cn, [b] zhangruiyou@ise.neu.edu.cn, [c] 2110354@stu.neu.edu.cn

**Abstract.** Edge computing is an emerging computing architecture. The scheduling and optimization of the tasks on the edge side of the smart factory can effectively reduce the processing delay and improve the utilization efficiency of servers. This study focuses on the problem of edge-side task scheduling with the goal of minimizing the maximum completion time of the tasks. A first-price sealed-bid auction based algorithm and a genetic algorithm with elite retention strategy are designed to solve the problem. The experimental results indicate that the auction-based scheduling algorithm has better real-time performances compared to the genetic algorithm.

**Keywords:** edge computing; task scheduling; optimization; auction mechanism; genetic algorithm; smart factory.

## 1. Introduction

Edge computing is an emerging computing architecture proposed on the basis of cloud computing [1]. With the rapid development of Internet of Things and artificial intelligence technologies, the terminal devices distributed in the smart factory expect services with lower latencies and lower energy consumptions. Edge computing can meet these requirements by moving the computation and storage capacity of the computing platform to the edge-side of the factory [2, 3].

The scheduling of the edge computing tasks among the edge servers faces great challenges. Firstly, the servers are often heterogeneous and multiple types of resources constrain the allocation of the tasks, which make the problem complicated. Secondly, the scheduling of the tasks raises higher requirements for the real-time performance of the algorithm.

This work makes the following contributions to the field. Firstly, the edge-side task scheduling problem is introduced minimizing the maximum completion time of the tasks. Secondly, an auction-based algorithm is proposed to solve the problem and a genetic algorithm is also designed for comparison. Finally, all the solving methods are validated and evaluated based on several randomly generated instances with some concluding remarks proposed.

The rest of the paper is organized as follows. Section II discusses the related literature. The edge-side task scheduling problem is described in Section III. The auction-based scheduling algorithm and the genetic algorithm are designed in Section IV and Section V, respectively. In Section VI, validations and evaluations of the solving methods are performed. Finally, Section VII summarizes the whole paper and proposes future directions.

## 2. Literature Review

In recent years, many researchers studied the task scheduling problem in edge computing. For example, reference [4] divided the task scheduling problem of edge computing into three categories, including the independent task offloading, the resource-constrained task offloading, and the multitasking offloading. Reference [5] considered the average latency of the tasks and the average power consumption of the devices, and proposed an approach based on the Markov decision process to minimize the power-constrained delay. Reference [6] developed an innovative framework based on the time division multiple access protocol to minimize the total energy consumption of the servers. Reference [7] established a distributed game model to optimize the

ISSN:2790-1688 DOI: 10.56028/aetr.4.1.12.2023
energy consumption of servers and the total completion time of tasks. Reference [8] established a mixed-integer linear programming model to maximize the utilization of servers under deadlines of tasks. Reference [9] developed a deep reinforcement learning based method to investigate the offloading of the tasks.

Some scholars adopted the auction-based method to the task scheduling problem. For instance, reference [10] considered the task processing latency and power consumption constraints of mobile devices. With the goal of maximizing the profit of edge servers, they established a mathematical model based on the auction mechanism to offload the tasks. Differently from this article, we investigate the task scheduling problem in the edge-side of the smart factory minimizing the maximum completion time of the tasks.

In summary, few studies used the auction mechanism based method to solve the edge-side task scheduling problem. Differently from the existing articles, this paper presents a scheduling algorithm based on the first-price sealed-bid auction and designs a genetic algorithm for comparison.

## 3. Edge-Side Task Scheduling Problem

An intelligent factory deploys an edge computing network with several edge servers for its production lines. The data collected on the industrial site is packaged as edge computing tasks. The devices in industrial field, managed by a dispatch center, can transmit the tasks to the edge servers for processing. Different tasks constitute the set $N$. The amount of data to be transmitted and the computational workload for task $i \in N$ are $D_i(\geq 0)$ and $C_i(\geq 0)$, respectively. The memory required by task $i$ is $R_i(\geq 0)$. The set of software and hardware resources required by task $i$ is $H_i$.

The available edge servers constitute the set $M$. The memory capacity and processing speed of server $j \in M$ are $Q_j(\geq 0)$ and $f_j(\geq 0)$, respectively. The set of software and hardware resources possessed by server $j$ is $S_j$. Each server can communicate with the devices in the industrial field directly. The data transfer speed between any field device and server $j$ is $B_j(\geq 0)$. It is assumed that each server can only process one task at the same time. For the tasks assigned to the same server, the dispatch center will inform the corresponding field devices to transmit them in sequence and a new task is transmitted once the execution of the previous task is completed except for the first one.

The task scheduling problem dispatches the tasks to the edge servers with the goal of minimizing the maximum completion time of the tasks. The allocation of the tasks is constrained by the available resources of the servers. Server $j$ can process task $i$ if and only if $R_i \leq Q_j$ and $H_i \subseteq S_j$ both hold.

## 4. Auction-Based Scheduling Algorithm

The auction-based scheduling algorithm regards the tasks as commodities and regards the servers as bidders. The tasks are auctioned in multiple rounds. The number of tasks able to be auctioned in any round is $w(\geq 0)$. Each server can get at most one task in a round of the auction. For each round of the algorithm, all the servers bid for all the tasks able to be auctioned and these tasks are successively distributed to the servers according to the ascending order of their computational workload. Each task is assigned to the server with the highest valid bid for it.

The tasks assigned to the same server are transmitted in the order of their successful transaction. If server $j$ can process task $i$, the bid of server $j$ for task $i$ is inversely proportional to the completion time of task $i$ on server $j$, which is the summation of the data transmission time of task $i$ to server $j$, the execution time of task $i$ on server $j$ and the cumulative queue time on server $j$. Otherwise, the bid of server $j$ for task $i$ is equal to zero.

Given $M$ as the set of the available edge servers and $N$ as the set of all the assignable tasks, the process of the algorithm is described as follows.

ISSN:2790-1688 DOI: 10.56028/aetr.4.1.12.2023

Step 1. Let $q_j$ be the cumulative queue time on server j. Initialize $q_j = 0$ for any $j \in M$ and $\epsilon = 1$.

Step 2. Start round $\epsilon$ of the auction. Let $g_\epsilon = 0$ and $Z_\epsilon = \emptyset$.

Step 3. Add the task with the minimum computational workload in $N \setminus Z_\epsilon$ to $Z_\epsilon$. If $|Z_\epsilon| = \min\{w, |N|\}$, go to Step 4; otherwise, return to Step 3.

Step 4. Let $p_{ij}$ be the bid of server j for task i. Calculate $p_{ij}$ of any server $j \in M$ for any task $i \in Z_\epsilon$ according to the following bidding rule.

$$p_{ij} = \begin{cases} \left(\frac{D_i}{B_j} + \frac{C_i}{f_j} + q_j\right)^{-1}, & \text{if } R_i \le Q_j \text{ and } H_i \subseteq S_j, \\ 0, & \text{otherwise,} \end{cases}$$

Step 5. Let k be the task with the minimum computational workload in $Z_\epsilon$. Let m be the server with the highest bid for task k in M. If $p_{km} > 0$, assign task k to server m, and let $g_\epsilon = g_\epsilon + 1$, $q_m = (p_{km})^{-1}$, $N = N \setminus \{k\}$ and $p_{im} = 0$ for any $i \in Z_\epsilon$; otherwise, task k cannot be assigned to any server in round $\epsilon$ of the auction.

Step 6. Let $Z_\epsilon = Z_\epsilon \setminus \{k\}$. If $Z_\epsilon$ is empty or $g_\epsilon = |M|$, terminate round $\epsilon$ of the auction and go to Step 7; otherwise, return to Step 5.

Step 7. If N is not empty, let $\epsilon = \epsilon + 1$ and return to Step 2; otherwise, output the results and terminate the algorithm.

## 5. Genetic Algorithm

Genetic algorithm is a kind of metaheuristic algorithm inspired by the theories of biological evolution. It selects new populations based on the excellent chromosomes in the parents and expands the searching space by changing the gene sequences of the chromosomes using crossover and mutation operators.

### 5.1 Encoding and decoding

The designed genetic algorithm uses the integer coding scheme, the length of which is the number of the tasks. Each position in the scheme represents a task. The element in each position represents the server processing the corresponding task. For example, an instance of the edge-side task scheduling problem with five tasks and four servers can be encoded as

$X = (1,2,1,3,4)$,

which means that Task 1 and Task 3 are assigned to Server 1, Task 2 is assigned to Server 2, Task 4 is assigned to Server 3, and Task 5 is assigned to Server 4.

The fitness of a chromosome is the maximum completion time of the tasks, which is equal to the maximum running time of the servers. Given an allocating scheme of the tasks, the running time of each server is the summation of the execution time and data transmission time for all the tasks assigned to that server.

### 5.2 Initial population

The initial population is composed of a greedy solution and several randomly generated chromosomes. In the greedy solution, the tasks assigned to the same server are assumed to be transmitted according to the ascending order of their computational workload. Given N as the set of all the assignable tasks, the procedure of the generation of the greedy solution is as follows.

Step 1. Allocate the task with the minimum computational workload in N to the available server with the earliest completion time. Update set N.

Step 2. If N is empty, output the solution and terminate the greedy algorithm; otherwise, return to Step 1.

In the remaining chromosomes, each task is assigned to any available server.

ISSN:2790-1688 DOI: 10.56028/aetr.4.1.12.2023

### 5.3 Selection, crossover and mutation

The selection operator of the genetic algorithm adopts the roulette wheel mechanism with the elite retention strategy. The crossover operator uses the single-point crossover method. The mutation operator is to randomly move a task on the server with the longest running time to any other available server. The genetic algorithm is terminated if the maximum iteration is reached.

## 6. Experiments and Analyses

All experiments were carried out on a personal computer with AMD R7-5800H, octa-core CPU (3.20 GHz), 16.0G RAM, and a 64-bit Windows 11 operating system. The Java language in IntelliJ IDEA 2021.3.2 was used to code the auction-based scheduling algorithm and the genetic algorithm.

### 6.1 Setting of experiments

There are multiple distributed servers and multiple terminal devices generating tasks on the edge side of the smart factory. Six small-scaled instances named Instances S1-S6 and six large-scaled instances named Instances L1-L6 were generated randomly. The numbers of edge servers in small- and large-scaled instances were set as five and ten, respectively. The numbers of tasks for small- and large-scaled instances were set to ten and one hundred, respectively.

The computational workload for each task was randomly generated between 5-25 million instructions (MI). The memory capacity required by each task was randomly generated between 50MB-150MB. The amount of data needed to be transmitted for each task was randomly generated between 5kB-10kB. The processing of the tasks requires software resources such as MATLAB and SPSS, as well as hardware resources such as GPU. The hardware and software resources are numbered sequentially. The set of hardware and software resources required by each task was randomly generated as a subset of $\{1, 2, 3, 4\}$.

The processing speed of each server was randomly generated between 100MIPS-350MIPS. The memory capacity of each server was randomly selected from 128MB, 256MB, 512MB and 1024MB. The data transfer speed between the terminal devices and each edge server was randomly selected from 128kB/s, 256kB/s, and 512kB/s. The set of hardware and software resources possessed by each server was randomly generated as a subset of $\{1, 2, 3, 4\}$.

$w$ in the auction-based scheduling algorithm were set as five for small-scaled instances and ten for large-scaled instances. The crossover rate and mutation rate of the genetic algorithm were set to 0.8 and 0.4, respectively. The population size and maximum iteration of the genetic algorithm were set to 200 and 2000, respectively.

### 6.2 Experimental analyses

Table I displays the computational results of the small-scaled instances. According to Table I, there are no differences between the objectives of the solutions provided by the two algorithms for five (i.e., Instances S2-S6) out of the six small-scaled instances. For Instance S1, the auction-based scheduling algorithm got a worse solution compared to the genetic algorithm. However, the running time of the auction-based scheduling algorithm was greatly reduced (i.e., less than 1ms).

Table II shows the computational results of the large-scaled instances. The auction-based scheduling algorithm provided worse solutions for five (i.e., Instances L1-L3 and L5-L6) out of the six large-scaled instances, compared to the genetic algorithm. However, the auction-based scheduling algorithm has great advantages considering the running time (i.e., less than 1ms). Moreover, the objectives of the large-scaled instances were increased compared to the small-scaled instances. This was due to the raise of the numbers of tasks.

Instance S1 is further used to illustrate the properties of the task scheduling results provided by the auction-based scheduling algorithm. The information about Instance S1 is shown in Table III and Table IV. The solution of Instance S1 provided by the auction-based algorithm is shown in Table V. It can be seen that Server 2 and Server 4 with sufficient resources and higher processing

ISSN:2790-1688 DOI: 10.56028/aetr.4.1.12.2023
speed obtained most tasks, which can reduce the processing time of these tasks. Moreover, Server 3 did not obtain any task. The reason behind this phenomenon was that the hardware and software resources of Server 3 were limited and the processing speed of Server 3 was low. Although Server 3 had the ability to process Task 3 and Task 10 according to Table III and Table IV, these tasks were finally assigned to Server 5 with higher processing speed and higher data transmission speed. Under the premise of satisfying the resource constraints, a shorter completion time of the tasks can be obtained by reducing their transmission time and processing time.

Table1 Results of Small-Scaled Instances.

| Instance | Auction-based scheduling algorithm | | Genetic algorithm | |
|---|---|---|---|---|
| | OBJ. (s) | CPU time | OBJ. (s) | CPU time (ms) |
| S1 | 0.305 | <1ms | 0.285 | 235 |
| S2 | 0.930 | | 0.930 | 220 |
| S3 | 0.853 | | 0.853 | 220 |
| S4 | 0.616 | | 0.616 | 204 |
| S5 | 0.574 | | 0.574 | 221 |
| S6 | 0.667 | | 0.667 | 220 |

Table2 Results of Large-Scaled Instances.

| Instance | Auction-based scheduling algorithm | | Genetic algorithm | |
|---|---|---|---|---|
| | OBJ. (s) | CPU time | OBJ. (s) | CPU time (ms) |
| L1 | 2.330 | <1ms | 1.351 | 815 |
| L2 | 1.758 | | 1.368 | 832 |
| L3 | 2.596 | | 2.257 | 816 |
| L4 | 3.082 | | 3.082 | 812 |
| L5 | 3.471 | | 3.021 | 816 |
| L6 | 1.702 | | 1.391 | 816 |

Table3 Properties of Servers in Instance S1.

| Server | Processing speed (MIPS) | Data transfer speed (kB/s) | Memory capacity (MB) | Software and hardware resources |
|---|---|---|---|---|
| 1 | 171 | 256 | 256 | 1, 2, 3 |
| 2 | 345 | 128 | 128 | 1, 2, 3, 4 |
| 3 | 123 | 128 | 1024 | 3, 4 |
| 4 | 213 | 512 | 512 | 1, 2, 3, 4 |
| 5 | 252 | 256 | 128 | 3 |

Table4 Properties of Tasks in Instance S1.

| Task | Computational workload (MI) | Amount of data (kB) | Required memory capacity (MB) | Required software and hardware resources |
|---|---|---|---|---|
| 1 | 8 | 7 | 85 | 1, 2, 3, 4 |
| 2 | 18 | 8 | 74 | 1, 2, 3, 4 |
| 3 | 21 | 6 | 63 | 3 |
| 4 | 10 | 8 | 55 | 1, 2, 4 |
| 5 | 9 | 9 | 79 | 1 |
| 6 | 7 | 6 | 68 | 1, 2, 3, 4 |
| 7 | 16 | 5 | 141 | 2, 4 |
| 8 | 12 | 9 | 52 | 1, 2, 3, 4 |
| 9 | 22 | 5 | 97 | 1, 2, 3, 4 |
| 10 | 21 | 5 | 95 | 3 |

ISSN:2790-1688 DOI: 10.56028/aetr.4.1.12.2023

Table5 Results of Task Scheduling in Instance S1.

| Server | Assigned task |
|---|---|
| 1 | 5 |
| 2 | 1, 2, 8 |
| 3 | None |
| 4 | 4, 6, 7, 9 |
| 5 | 3, 10 |

## 7. Conclusions and Future Directions

This paper studies the edge-side task scheduling problem with heterogeneous edge servers in the smart factory. The goal is to minimize the maximum completion time of the tasks. An auction-based scheduling algorithm and a genetic algorithm are designed to solve the problem. The experimental results show that the auction-based scheduling algorithm has better real-time performances than the genetic algorithm although the solution provided by the auction-based algorithm may be worse.

At present, there are still some deficiencies in this paper. For example, the load balance degree of servers significantly impacts the efficiency of the edge computing network. The load balance degree for servers and the maximum completion time of tasks can be optimized simultaneously and some multi-objective optimization methods can be used to solve this problem.

## Acknowledgment

This work was partially supported by the National Key R&D Program of China (2019YFB1705003).

## References

[1] H. H. Pang and K-L. Tan, “Authenticating query results in edge computing,” Proc. 20th International Conference on Data Engineering (ICDE 04), IEEE press, 2004, pp. 560–571, doi: 10.1109/ICDE.2004.1320027.

[2] F. Bonomi, R. Milito, J. Zhu and S. Addepalli, “Fog computing and its role in the Internet of Things,” Proc. 1st Edition of the MCC Workshop on Mobile Cloud Computing (MCC 12), ACM press, 2012, pp. 13-15, doi: 10.1145/2342509.2342513.

[3] L. M. Vaquero and L. Rodero-Merino, “Finding your way in the fog: Towards a comprehensive definition of fog computing,” ACM SIGCOMM Computer Communication Review, vol. 44, no. 5, Oct. 2014, pp. 27-32, doi: 10.1145/2677046.2677052.

[4] 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, Oct. 2016, pp. 637-646, doi: 10.1109/JIOT.2016.2579198.

[5] J. Liu, Y. Mao, J. Zhang and K. B. Letaief, “Delay-optimal computation task scheduling for mobile-edge computing systems,” Proc. IEEE International Symp. Information Theory (ISIT 16), IEEE press, 2016, pp. 1451-1455, doi: 10.1109/ISIT.2016.7541539.

[6] F. Wang, J. Xu, X. Wang and S. Cui, “Joint offloading and computing optimization in wireless powered mobile-edge computing systems,” IEEE Transactions on Wireless Communications, vol. 17, no. 3, Mar. 2018, pp. 1784-1797, doi: 10.1109/TWC.2017.2785305.

[7] J. Zhang, W. Xia, F. Yan and L. Shen, “Joint computation offloading and resource allocation optimization in heterogeneous networks with mobile edge computing,” IEEE Access, vol. 6, 2018, pp. 19324-19337, doi: 10.1109/ACCESS.2018.2819690.

[8] O. Skarlat, M. Nardelli, S. Schulte and S. Dustdar, “Towards QoS-aware fog service placement,” Proc. IEEE 1st International Conference on Fog and Edge Computing (ICFEC 17), IEEE press, 2017, pp. 89-96, doi: 10.1109/ICFEC.2017.12.

ISSN:2790-1688 DOI: 10.56028/aetr.4.1.12.2023
[9] Y. Zhu, Y. Hu, T. Yang, T. Yang, J. Vogt, et al., "Reliability-optimal offloading in low-latency edge computing networks: Analytical and reinforcement learning based designs," IEEE Transactions on Vehicular Technology, vol. 70, no. 6, Jun. 2021, pp. 6058-6072, doi: 10.1109/TVT.2021.3073791.

[10] F. Mashhadi, S. A. S. Monroy, A. Bozorgchenani and D. Tarchi, "Optimal auction for delay and energy constrained task offloading in mobile edge computing," Computer Networks, vol. 183, Art. no. 107527, Dec. 2020, pp. 1-10, doi: 10.1016/j.comnet.2020.107527.