frontiers | Frontiers in Energy Research

TYPE Original Research
PUBLISHED 24 June 2024
DOI 10.3389/fenrg.2024.1414516

Check for updates

OPEN ACCESS

EDITED BY
Chaojie Li,
University of New South Wales, Australia

REVIEWED BY
Muhammad Tariq,
National University of Computer and Emerging Sciences, Pakistan
Keping Yu,
Hosei University, Japan

*CORRESPONDENCE
Zhan Shi,
w_1234567892021@163.com

RECEIVED 09 April 2024
ACCEPTED 02 May 2024
PUBLISHED 24 June 2024

CITATION
Shi Z (2024), Edge–end collaborative secure and rapid response method for multi-flow aggregated energy dispatch service in a distribution grid.
Front. Energy Res. 12:1414516.
doi: 10.3389/fenrg.2024.1414516

Edge–end collaborative secure and rapid response method for multi-flow aggregated energy dispatch service in a distribution grid

Zhan Shi*

Power Dispatching and Controlling Center of Guangdong Power Grid Company Limited, Guangdong, Guangzhou, China

With a high proportion of distributed source–grid–load–storage resources penetrating into the distribution network, multi-flow aggregated energy dispatch is essential to enhance renewable energy consumption capacity and maintain grid stability. However, intelligent energy dispatch has stringent requirements for low latency and security, which necessitates the development of secure and rapid response methods. In this paper, we combine the edge–end collaboration with container microservices architecture and propose a container and microservice empowered edge–end collaborative secure and rapid response framework for multi-flow aggregated energy dispatch service in a distribution grid. Then, a container selection optimization problem is formulated to minimize the total microservices execution delay. To cope with the dynamic environment such as electromagnetic interference, noise, and workload variation, we propose a microservice container selection algorithm based on an enhanced ant colony with empirical SINR and delay performance awareness. The proposed algorithm benefits from the local and global integrated pheromone updating methods and the empirical performance-based dynamic pheromone and heuristic information updating mechanism. Simulation results demonstrate that the proposed algorithm outperforms the existing methods in average service execution delay and convergence speed.

KEYWORDS

enhanced ant colony algorithm, multi-flow aggregated energy dispatch, end–edge collaboration, container selection, microservice computing, 5G edge computing

1 Introduction

As a key link in the construction of a new type of power system, the distribution grid, which is located between the power transmission system and the power consumption equipment, combines various advanced communication technologies such as 5G and intelligent management systems to provide users with high-quality and reliable power supply (Shunxin et al., 2022). With an ultra-high percentage of intermittent distributed renewable energy penetrating the grid, the distribution grid undertakes more and more responsibility for aggregating and dispatching energy to accommodate bidirectional power demand on both the demand side and load side (Wu et al., 2019; Li et al., 2022). Aggregated energy dispatch involves the integration and collaborative management of dispersed resources such as energy sources, loads, and energy storage through technical means to achieve efficient operation of the power system and energy utilization (Tariq and Poor, 2018; Yang et al., 2020). On one hand, it refers to multiple flows, which includes the data flow generated by distributed electrical equipment and offloaded to edge servers for computing, the energy flow formed by the intelligent dispatching with the interaction of source–grid–load–storage, and the service flow established in a chain form through the processing of decomposed services (Shen et al., 2023; Xue et al., 2023). On the other hand, since decentralized distribution grid operators of source, load, and storage participate in the dispatch as independent market players, the complex interactive game among them needs to aggregate global information as a support for dispatching strategy. Oftentimes, aggregated energy dispatch services impose stringent low latency requirements, prompting a pressing need for the study of rapid response methods (Dong et al., 2016).

The combination of edge–end collaboration technology and container microservices architecture provides a viable solution for achieving rapid responses in aggregated energy dispatch (Buzato et al., 2018). Microservice architecture divides complex service into multiple simple microservices with small size, low consumption, and independent operation, which forms a chain structure and significantly improves the computing efficiency (Naik et al., 2021; Zhou et al., 2023a; Lyu et al., 2020). Container technology is a lightweight kernel-level virtualization technology in the operating system layer. Each container has independent resources and is not interfered with by the other processes (Al-Debagy and Martinek, 2018; Wu et al., 2018). Through edge–end collaboration, Internet of Things (IoT) devices offload the microservices to containers on edge servers for further processing (Li et al., 2021). Nevertheless, as containers are limited in their ability to handle specific types of microservices, edge–end collaboration struggles with microservices' heterogeneity, leading to uneven resource utilization across edge servers and significant service execution delay. Edge–edge collaboration serves as an extension of edge–end collaboration, allowing the migration of microservices among edge servers via a 5G network (Wang et al., 2022). This facilitates service execution, load balancing, and better utilization of computing resources. To achieve an organic combination of fast response and edge–end collaboration, the core is to choose appropriate containers for microservices to minimize the response time (Zhou et al., 2023b).

Edge–end collaborative container selection for power services should be carefully conducted due to the limited resources and latency concerns in a distribution grid. To this end, some works have been devoted to developing an optimized container selection solution. Chhikara et al. (2021) proposed a best-fit container selection algorithm for finding the best suitable destination host for the migration process and used the heap data structure to get the lowest overhead node with constant time. Tan et al. (2022) proposed a cooperative coevolution genetic programming hyper-heuristic approach to solve the container selection problem and reduce energy consumption. However, they are not applicable to the multi-flow aggregated energy dispatch scenario with a coupled relationship of microservices and containers. Tang et al. (2019) modeled the container migration strategy as multiple-dimensional Markov decision process spaces and proposed a deep reinforcement learning algorithm to realize rapid container selection. Gao et al. (2020) described a microservice composition problem for multi-cloud environments and proposed an artificial immune algorithm for optimal container strategies. Although these approaches provide some valid ideas for container selection, their solving ability fall short in the huge solution space with multiple microservices and containers.

The ant colony algorithm is an intelligent heuristic algorithm, which offers advantages such as distributed computing capacity, high parallelism, and adaptability in container selection problem. Han et al. (2021) investigated interference-aware online multi-component service placement in edge cloud networks and translated it into an ant colony optimization problem to provide computational offloading services with quality of service guarantees. Cabrera et al. (2023) presented a mobility-aware, priority-driven, ant colony algorithm-based service placement model that prioritizes according to their criticality and minimizes service delays. However, they struggle to cope with dynamic environmental changes such as load fluctuations and electromagnetic interference. Additionally, since some key parameters and initial values have a large impact on the performance of the algorithm, their search ability tends to be unstable and prone to local optimal solutions (Tariq et al., 2021).

There are still several challenges in the problem of optimizing container selection for microservices of multi-flow aggregated energy dispatch. First, the relationship between the containers and microservices is a complicated coupling. On one hand, there are complex interdependencies between successive microservices generated by the same device. On the other hand, microservices generated by different devices but processed in the same container are also intertwined. The coupling relationship brings about the problem of a large solution space and the curse of dimensionality, leading to the inability to apply existing model-based algorithms and closed-form solutions. Second, although the ant colony algorithm has the advantages of strong distributed computing capability, high parallelism, and adaptability, it suffers from local optimality due to strong randomness and often leads to slower convergence. Thus, it is difficult to directly apply the traditional ant colony algorithm to find a global optimal solution for the complex coupled microservice selection problem. In addition, the presence of uncertainties such as electromagnetic interference, noise, and workload variation leads to large fluctuations in performance, which further reduces the learning efficiency of the ant colony algorithm. Finally, the execution of microservices processing encounters multiple security and privacy challenges. Malicious devices or edge servers pose threats through various attacks, aiming to maximize their gains. Concurrently, certain malicious entities attempt to extract sensitive information from intercepted data exchanged between the devices and edge servers. Without resolving these security and privacy concerns, devices and edge servers may hesitate to accept the edge–end computing framework.

Security of microservice processing requires a reliable data management scheme, and the traditional data management scheme adopts centralized storage, which is easy to manage and maintain the data by storing the data in the cloud data center after unified encryption. However, due to the complex structure of the new distribution network and the high privacy of the data, the application of centralized storage will face data security risks, which are mainly manifested in the poor security of the centralized data center and its vulnerability to single-point-of-failure attacks. The centralized storage data have the risk of privacy leakage and are vulnerable to malicious tampering by attackers.

Blockchain, as a distributed ledger technology, has the advantages of decentralization, data traceability, and being tamper-proof. Unlike in centralized storage, data in a blockchain are encrypted and stored in the form of transactions in the nodes running the blockchain. Transactions are constantly assembled to form blocks, and blocks are linked to each other through hash values to form a blockchain. Each transaction and each block has corresponding timestamp information, and the consistency of the stored data among the nodes is guaranteed by the consensus protocol. Benefiting from the good characteristics of blockchain, the blockchain-based service offloading management scheme has gradually become the mainstream scheme.

In response to the above issues, this paper proposes an edge–end collaborative secure and rapid response method for multi-flow aggregated energy dispatch service in a distribution grid. First, the container and microservice-empowered edge–end collaborative secure and rapid response framework for multi-flow aggregated energy dispatch service in a distribution grid is proposed. On this basis, we propose a scheme for smart contract design and blockchain construction. In addition, the models of end–edge microservice offloading delay, edge–edge microservice migration delay, microservice data queuing and computing delay, and total execution delay of microservices are established. Then, the optimization problem is formulated, which aims to minimize the time-averaged total execution delay under the constraints of container selection and resources. A microservice container selection algorithm based on the enhanced ant colony with empirical SINR and delay performance awareness is proposed to solve the optimization problem. By incorporating heuristic information updating based on empirical performance awareness into the conventional ant colony and combining local and global integrated pheromone updating, the algorithm improves the searching efficiency and convergence speed and realizes the efficient and flexible selection of containers for microservices. Finally, the superiority of the proposed algorithm is verified through simulations. The contributions are summarized as follows:

- *Improved searching efficiency under coupled solution space:* we employ the efficient searching ability of the heuristic algorithms to solve the issues caused by coupled solution space. Specifically, we incorporate the ant colony algorithm into the searching process of our container selection optimization problem, where paths for ants are mapped to the container selection strategies for microservices. By modeling the information transfer and feedback mechanism of ants, the searching direction can be guided by updating the pheromones and heuristics on the path, which helps speed up the searching efficiency.
- *Adaptive path selection with empirical performance awareness:* considering the uncertainties in the environment, we calculate the historical average SINR between the edge servers and the historical average queuing delay and computing delay of all the microservices processed in the containers. Based on these empirical performances, local pheromone and heuristic information can be dynamically updated to accommodate uncertainties for better convergence performance.
- *Blockchain-based secure microservice processing:* we propose a secure microservice processing scheme based on blockchain construction and smart contract design aimed at guaranteeing privacy, fairness, and security. Our approach leverages the Merkle hash tree and smart contracts to implement "proof-of-computing" and mitigate risks.

2 System model

In this section, we introduce the system model, which includes the proposed secure and rapid response framework, smart contract design and blockchain construction, and the delay model.

2.1 Container and microservice empowered edge–end collaborative secure and rapid response framework

The container and microservice-empowered edge–end collaborative secure and rapid response framework for multi-flow aggregated energy dispatch service in a distribution grid is shown in Figure 1, which consists of the device layer and edge layer. In the device layer, there are primarily three types of electric equipment: power generation equipment such as distributed photovoltaic (PV) and thermal power units, load equipment such as intelligent charging piles and street lights, and energy storage equipment such as batteries (Meshram et al., 2022). Multi-flow energy dispatch services achieve energy supply and demand balance through the coordination of PV output and energy storage charging and discharging with load demand. The dispatch service can be further decomposed into a chain of interrelated microservices, which contain various types, such as device status collection, active power control, and historical data storage. A number of IoT devices are deployed on electric equipment to collect the data on energy dispatch microservice like voltage, current, and temperature. Afterward, the collected data are offloaded to the containers located in the edge layer through multimodal channels including power line communication (PLC) and high-speed radio frequency (HRF). The edge layer is composed of edge servers, which communicate with each other through 5G base stations and harness container technology for the processing of microservice data offloaded by the devices. Container technology facilitates virtualization by isolating the resources using a shared operating system kernel and the related toolsets. The orchestration of multiple containers is managed systematically to enhance the efficiency of microservice data processing, thereby achieving a secure and rapid response for multi-flow aggregated energy dispatch.

Multi-flow aggregated energy dispatch involves the interaction of data flow, energy flow, and service flow. Data flow is formed by the microservice data collected by IoT devices and offloaded from the device layer to the edge layer for subsequent processing. These data are further processed in containers to support the energy dispatch service. Energy flow is formed by intelligently dispatching distributed energy sources, grid, load, and energy storage to realize the energy demand–supply balance. Service flow is formed by processing chain-structured microservices and feeding back service performance to improve the strategy of container selection of data flow.

Considering a total of T slots, the slot set is denoted as $\mathcal{T} = \{1,\ldots,t,\ldots,T\}$. During each slot, it is assumed that the system state such as channel gain, container computing resource, bandwidth,

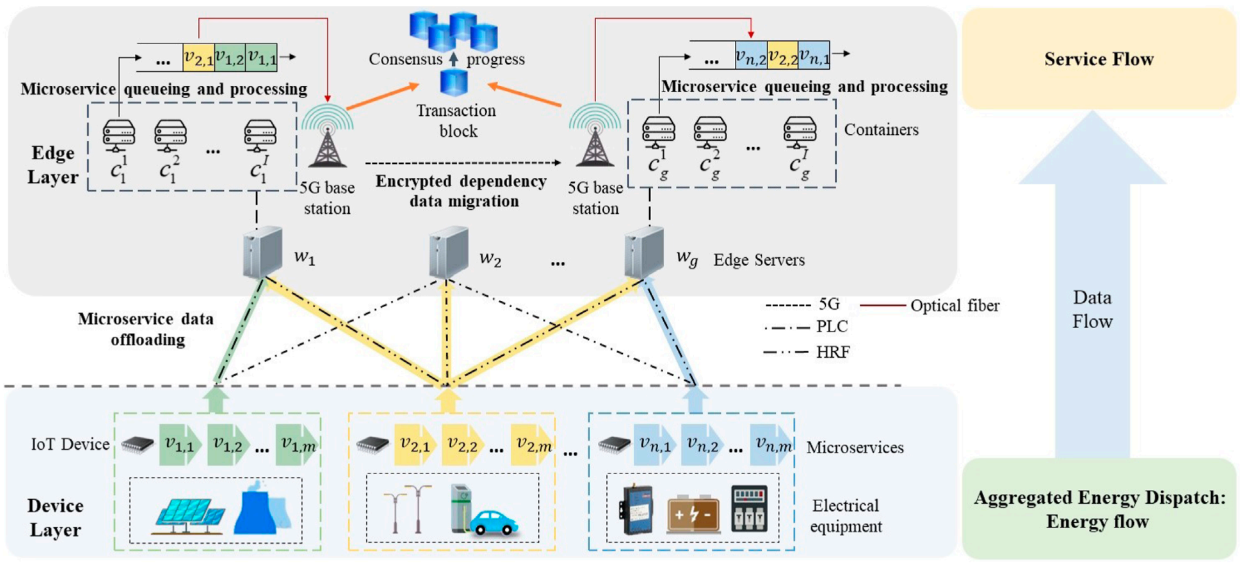

FIGURE 1
Container and microservice-empowered edge–end collaborative secure and rapid response framework for multi-flow aggregated energy dispatch service in the distribution grid.

and transmission power remains unchanged. There are a total of N devices, and the n-th device generates M microservices. The set of microservices is defined as $\mathcal{V}_n = \{v_{n,1}, \ldots, v_{n,m}, \ldots, v_{n,M}\}$, where $v_{n,m}$ denotes the m-th microservice of the n-th IoT device. We assume that the edge layer consists of G edge servers, and its set is denoted as $\mathcal{W} = \{w_1, \ldots, w_g, \ldots, w_G\}$. Each edge server virtualizes the computing resources into I containers. Specifically, the container set of edge server w_g is denoted as $\mathcal{C}_g = \{c_g^1, \ldots, c_g^i, \ldots, c_g^I\}$, where c_g^i represents the i-th container of w_g.

In each slot, devices offload microservice data to edge containers for processing. A slot ends until all the generated microservices of devices are processed. The variable of microservice container selection is defined as $s_{n,m}^{g,i}(t) \in \{0,1\}$. If the microservice $v_{n,m}$ of the n-th device is offloaded to the container c_g^i of the edge server w_g, then $s_{n,m}^{g,i}(t) = 1$. Otherwise, $s_{n,m}^{g,i}(t) = 0$. Due to constrained computing and storage resources, a container can only process a limited type of microservice data. The type matching variable of microservice is defined as $z_{n,m}^{g,i} \in \{0,1\}$. If the container c_g^i can process the type of microservice $v_{n,m}$, then $z_{n,m}^{g,i} = 1$, and otherwise, $z_{n,m}^{g,i} = 0$. Therefore, consecutive microservices can be processed at containers of different edge servers through edge–edge collaboration. The complementary integration of end–edge offloading and edge–edge collaborative processing enables the achievement of better load balance and more flexible utilization of container computing resources.

In order to ensure the security of microservice processing and prevent privacy leakage, we employ the blockchain framework. Authorized base stations act as the consensus nodes to maintain the blockchain. Base stations include the register authority component, computation component, and storage component. The register authority component, tasked with overseeing registration and identity management, derives its jurisdiction from the governmental departments. It allocates a distinct digital certificate to individual devices to validate their authenticity. The computation component is accountable for formulating and executing smart contracts. Additionally, it engages in blockchain mining to obtain rewards. The storage component preserves the complete blockchain ledger, which is essential for verifying the authenticity of the blocks and facilitating microservices offloading and migration validation.

The total microservices computing process based on the blockchain framework is introduced as follows. First, all devices, edge servers, and base stations acquire their secure wallets, which contain a certain amount of digital currency for settling microservice offloading transactions. Each device generates its own key pair, including a public key and a private key, which are responsible for data encryption and decryption. The base station employs its public and private keys for the generation and verification of digital signatures. For the three components of the edge server, their key pairs are identical. Each device will also be registered with the register authority component for certification. Second, the device selects an available container for the microservice and sends its offloading or migration request to the edge server. Then, the edge server executes the smart contract, and the device offloads or migrates the encrypted data to the selected container. Third, the edge server checks the entire microservices computing process to check for any malicious behavior. Subsequently, honest microservices are rewarded in the form of digital currency, while malicious microservices will be penalized according to the smart contract. Finally, the edge server constructs a transaction block and uploads it to the blockchain. Other edge servers engage in competition to discover a valid proof-of-work, with the initial discovery being broadcasted to the remaining edge servers for verification. Upon receiving acceptance from the majority of the edge servers, the block is then appended to the end of the blockchain.

When processing chain-structured microservices at different edge servers, there is a dependency relationship between consecutive microservices. The subsequent microservice must wait for the completion of its preceding microservice processing and the migration of dependency data before it can be executed. Therefore, the total execution delay of microservices is composed of four parts: end–edge microservice offloading delay, edge–edge microservice migration delay, microservice queuing delay, and microservice computing delay. The paper aims to choose the appropriate container selection strategy $s_{n,m}^{g,i}(t)$ to minimize the total execution delay, thereby achieving secure and rapid response for multi-flow aggregated energy dispatch service in the distribution grid.

2.2 Smart contract design and blockchain construction

We design a smart contract to ensure the transparency and trustworthiness of the transaction. Smart contracts are jointly verified and executed by nodes on the blockchain network. Take edge server w_g as an example. Define the public and private key of the base station as K_{bs}^{pub} and K_{bs}^{pri}. Similarly, the key pair of the n-th device is denoted as $(K_{dn}^{pub}, K_{dn}^{pri})$ and that of the edge server w_g is denoted as $(K_{wg}^{pub}, \{K_{wg}^{pri}\})$, where $\{K_{wg}^{pri}\}$ represents the private keys of all the communication links linked to w_g. Signatures that are generated by the register authority component for the device and edge server are defined as SIG_{dn} and SIG_{wg}.

2.2.1 Microservice offloading request

Devices select w_g for microservice offloading and send the request $SIG_{dn}\|SIG_{sg}\|\tau_0$ to the computation component, where τ_0 is the delay requirement. Upon receiving the microservice offloading request, the device, edge server, and computation component each contribute a portion of coins to establish a deposit within τ_0. This deposit is held by the computation component during the execution of the smart contract. Following this, the computation component forwards the signatures of the device and edge server to the register authority component. Subsequently, the register authority component provides the certificate of the device to the edge server CER_{wg} and the certificate of the edge server to the device CER_{dn}, enabling both parties to mutually verify each other's validity, which is given by Eq. 1,

$$\begin{aligned} CER_{wg} &= K_{sg}^{pri}\|SIG_{wg}, \\ CER_{dn} &= K_{dn}^{pri}\|SIG_{dn}, \end{aligned} \tag{1}$$

where || denotes the combination of the public key and the signature to obtain the corresponding certificate.

2.2.2 Microservice offloading and migration

Assume that each microservice of the device comprises h basic fragments. These fragments are employed to construct a Merkle hash tree as the leaf nodes, facilitating the verification of data offloading and migration. To enhance security during transmission or migration, the device or edge server encrypts data using the public key of the next communication node w_g'. This encryption is represented as $ENC_{K_{wg'}^{pub}}(data)$, and the encrypted data are subsequently transmitted to the next node. In addition, the Merkle hash root value $ROOT(m_1)$ is generated by the starting point. This value is then forwarded to the computation component as $SIG_{dn}\|ROOT(m_1)$ from the device or $SIG_{wg}\|ROOT(m_1)$ from the edge server.

2.2.3 Microservice computation

The edge server w_g' utilizes its private key K_{wg}^{pri*} to decrypt the microservice data it receives, thus initiating the computation process. Subsequently, using both the microservice data and computation outcomes, w_g' generates the Merkle hash root value $ROOT(m_2)$.

2.2.4 Outcome feedback

$SIG_{wg'}\|ROOT(m_2)$ is transmitted to the computation component, which employs a Merkle tree-based proof-of-computing check mechanism to verify the computation results. This involves comparing $ROOT(m_1)$ with $ROOT(m_2)$. If $ROOT(m_2) = ROOT(m_1)$, it indicates an attempt by the edge server w_g' to deceive the computation component by directly using offloaded or migrated data to generate $ROOT(m_2)$. In case of equality or failure by w_g' to submit $ROOT(m_2)$ to the computation component within τ_0, the smart contract will terminate the transaction automatically, identifying w_g' as engaging in malicious behavior. w_g' will face penalties and be required to compensate the computation component with a payment of some currencies. The currencies from the deposit will be refunded to the others' wallets. The microservice offloading or migration failure event of w_g' will be logged in the block. Conversely, successful completion of microservices computing is acknowledged if no discrepancies arise.

Subsequently, w_g' employs the public key of the transmission starting point in its certificate to encrypt the results. These encrypted results are then forwarded back to the device as $ENC_{K_{dn}^{pub}}(outcome)$ or to the edge server as $ENC_{K_{wg'}^{pub}}(outcome)$.

2.2.5 Transaction settlement

Following the outcome feedback, the transmission starting point utilizes its private key for outcome decryption and then computes a new Merkle hash root value $ROOT(m3)$ based on $ROOT(m_1)$ and the received outcomes. Subsequently, $ROOT(m3)$ is transmitted to the computation component, either as $SIG_{dn}\|ROOT(m_3)$ from the device or $SIG_{wg}\|ROOT(m_3)$ from the edge server. The computation component settles the transaction by comparing $ROOT(m3)$ with $ROOT(m_2)$. If they are equal, it signifies that the edge server w_g' received the outcome within the stipulated delay. Both the communication parties receive a portion of currencies from the deposit, while the remainder is refunded to the base station's wallet. The successful microservice offloading and migration event of w_g' is recorded in the block. If they are not equal, the transmission starting point is penalized and required to compensate the computation component with a payment of currencies. The currencies in the deposit are then returned to the respective entities' wallets. The successful microservice offloading and migration event of w_g' is recorded in the block.

2.2.6 Blockchain construction

The base station initiates the creation of a block to record the transaction and submits it to the blockchain. Each block header comprises four key elements: a timestamp, difficulty level, the previous block's hash value, and the Merkle hash root value of the entire block body. The timestamp and difficulty are predetermined by the blockchain network. Every authorized base station endeavors to discover its unique proof-of-work by computing the block's hash value using a random variable *phi* and other pertinent data from the block header, including the timestamp, Merkle hash root value, and the previous block's hash value, denoted as $data_{head}$. This calculation aims to satisfy the condition $H(phi + data_{head}) < Difficulty$. The base station that first identifies a valid proof-of-work, represented by *phi*, broadcasts both the block and *phi* to other base stations within the blockchain network for verification. Upon consensus from the majority of the base stations regarding the validity of the proof-of-work, the block is permanently appended to the blockchain. The base station responsible for discovering this proof-of-work is rewarded with mining rewards.

2.3 End–edge microservice offloading delay

The delay of offloading the first microservice generated by the n-th device to the container c_g^i on the edge server w_g is given Eq. 2:

$$\tau_{n,g}^{off}(t) = s_{n,1}^{g,i}(t)\frac{a_{n,1}(t)}{R_{n,g}^{off}(t)}, \tag{2}$$

where $a_{n,1}(t)$ denotes the data size of the first microservice $v_{n,1}$ and $R_{n,g}^{off}(t)$ denotes the transmission rate from the n-th device to edge server w_g, which is calculated by Eq. 3:

$$R_{n,g}^{off}(t) = B_{n,g}\log_2\left(1 + SINR_{n,g}(t)\right). \tag{3}$$

where $B_{n,g}$ denotes the transmission bandwidth and $SINR_{n,g}(t)$ denotes the signal to interference plus noise ratio (SINR) from the n-th device to the edge server w_g. $SINR_{n,g}(t)$ is calculated by Eq. 4:

$$SINR_{n,g}(t) = \frac{P_n^{tran}(t)\,h_{n,g}(t)}{\delta + e_{n,g}(t)}, \tag{4}$$

where $P_n^{tran}(t)$ is the transmission power of the n-th device, $h_{n,g}(t)$ is the channel gain, δ is the white noise power, and $e_{n,g}(t)$ is the electromagnetic interference power. The alpha stable state distribution is used to characterize the electromagnetic interference (Zhou et al., 2016), and its characteristic function is calculated as Eq. 5:

$$E\left[\exp\left(\chi\varphi e_{n,g}(t)\right)\right] = \begin{cases} \exp\left(\chi\mu_{n,g}\varphi - \xi_{n,g}|\varphi|^{\alpha_{n,g}}\right. \\ \left.\left(1 - \chi\beta_{n,g}\mathrm{sgn}(\varphi)\tan\frac{\alpha_{n,g}\pi}{2}\right)\right), & \alpha_{n,g} \neq 1 \\ \exp\left(\chi\mu_{n,g}\varphi - \xi_{n,g}|\varphi|\right. \\ \left.-\chi\beta_{n,g}\mathrm{sgn}(\varphi)\ln|\varphi|\frac{2}{\pi}\right), & \alpha_{n,g} = 1 \end{cases}, \tag{5}$$

where χ and φ are characteristic exponents of the distribution. $\alpha_{n,g}$, $\beta_{n,g}$, $\xi_{n,g}$, and $\mu_{n,g}$ are the characteristic factors, skew parameters, scale parameters, and position parameters of electromagnetic interference, respectively.

2.4 Edge–edge microservice migration delay

When the microservice $v_{n,m}$ and its subsequent microservice $v_{n,m+1}$ are executed at different edge servers, after the execution of $v_{n,m}$, it is necessary to migrate the dependency data from the container where $v_{n,m}$ has been processed to the container where $v_{n,m+1}$ is processed. When $v_{n,m}$ and $v_{n,m+1}$ are processed at the same edge server, edge–edge microservice migration delay can be ignored. When $v_{n,m}$ and $v_{n,m+1}$ are processed at different edge servers, considering the data security in the 5G public network environment, data encryption must be performed when migrating data between different edge servers. Furthermore, the data encryption process involves computing the ciphertext, which includes extensive numerical computations and logical operations. In contrast, the decryption process simply requires the confirmation of a match between the ciphertext and the key, without requiring extensive computations. Hence, we only consider the delay induced by data encryption (Mota et al., 2017). The data encryption process of microservice $v_{n,m}$ on container c_g^i includes three parts: generating encrypted files, generating file summary, and generating digital signature.

To protect the confidentiality and integrity of sensitive microservice data, the edge server converts the original raw data into a coded format and generates encrypted files. The delay of generating encrypted files is calculated by Eq. 6:

$$\tau_{n,m,g,i}^{enc,1}(t) = \frac{a_{n,m+1}^{dep}(t)\,\chi_{enc}}{\xi_g^i}, \tag{6}$$

where $a_{n,m+1}^{dep}(t)$ is the size of the dependency data required by the executing microservice $v_{n,m+1}$. χ_{enc} is the computational complexity (cycles/bit) of the generating encrypted files. ξ_g^i is the available amount of the computing resources of container c_g^i per second.

Then, a summary of the encrypted file is generated to create a concise representation of it. The delay of generating the data file summary is calculated by Eq. 7:

$$\tau_{n,m,g,i}^{enc,2}(t) = \frac{a_{n,m+1}^{dep}(t)\,\delta_{enc}\chi_{dig}}{\xi_g^i}, \tag{7}$$

where δ_{enc} is the encryption ratio that represents the ratio of the encrypted data size to the original data size, and χ_{dig} is the computational complexity of generating the data file summary.

Finally, a digital signature is generated to create a unique identifier for the file summary using a private key. The delay of generating the digital signature is calculated by Eq. 8:

$$\tau_{n,m,g,i}^{enc,3}(t) = \frac{a_{n,m+1}^{dep}(t)\,\delta_{enc}\chi_{sig}}{\xi_g^i}, \tag{8}$$

where χ_{sig} is the computational complexity of generating the digital signature.

Therefore, the total data encryption delay of the microservice $v_{n,m}$ on container c_g^i is the sum of the delay of generating encrypted files, the delay of generating data file summary, and the delay of generating a digital signature, which is given by Eq. 9:

$$\tau_{n,m,g,i}^{enc}(t) = \tau_{n,m,g,i}^{enc,1}(t) + \tau_{n,m,g,i}^{enc,2}(t) + \tau_{n,m,g,i}^{enc,3}(t). \tag{9}$$

After data encryption of $v_{n,m}$, the encrypted dependency data are migrated to the container where the subsequent microservice

$v_{n,m+1}$ is processed. The delay of the migrating dependency data is calculated as Eq. 10:

$$\tau^{tra}_{n,m,g,i}(t)=\begin{cases}0, \sum_{i=1}^{I} s^{g,i}_{n,m+1}(t)=1\\ s^{g,i}_{n,m}(t)\sum_{g'=1}^{G}\sum_{i'=1}^{I} s^{g',i'}_{n,m+1}(t)\left(\tau^{enc}_{n,m,g,i}(t)+\tau^{sen}_{n,m,g,g'}(t)\right),\\ \text{Otherwise}\end{cases} \tag{10}$$

where $\sum_{i=1}^{I} s^{g,i}_{n,m+1}(t)=1$ represents that $v_{n,m}$ and $v_{n,m+1}$ are processed at the same edge server. $\tau^{sen}_{n,m,g,g'}(t)$ is the delay of migrating the encrypted dependency data $v_{n,m}$ from the edge server w_g to the edge server w'_g. Since the size of the data file summary and digital signature is too small to impact $\tau^{sen}_{n,m,g,g'}(t)$, it can be calculated simply as Eq. 11:

$$\tau^{sen}_{n,m,g,g'}(t)=\frac{a^{dep}_{n,m+1}(t)\,\delta_{enc}}{R^{tra}_{g,g'}(t)}, \tag{11}$$

where $R^{tra}_{g,g'}(t)$ denotes the transmission rate from the edge server w_g to the edge server w'_g in the t-th slot, which is calculated as Eq. 12:

$$R^{tra}_{g,g'}(t)=B_{g,g'}\log_2\left(1+SINR_{g,g'}(t)\right), \tag{12}$$

where $B_{g,g'}$ and $SINR_{g,g'}(t)$ represent the transmission bandwidth and SINR between the edge server w_g and the edge server w'_g, respectively.

2.5 Microservice data queuing and computing delay

Since a container can only process one microservice at a time, a queue is maintained in the container to cache the microservices waiting to be processed. The microservices in the queue will be processed according to the first-in first-out principle. After obtaining the encrypted dependency data of $v_{n,m-1}$, the queuing delay experienced by microservice $v_{n,m}$ at container c^i_g can be iteratively calculated by the queuing delay and computing the delay of the microservice that ranks just before $v_{n,m}$ at c^i_g, which is given by Eq. 13:

$$\tau^{que}_{n,m,g,i}(t)=s^{g,i}_{n,m}(t)\left(\Delta\tau^{que}_{n,m,g,i}(t)+\Delta\tau^{com}_{n,m,g,i}(t)\right), \tag{13}$$

where $\Delta\tau^{que}_{n,m,g,i}(t)$ and $\Delta\tau^{com}_{n,m,g,i}(t)$ represent the queuing delay and computing delay of the microservice that ranks just before $v_{n,m}$ on container c^i_g, respectively.

The container c^i_g utilizes its available computing resources to process microservices in a serial manner, and the processing delay of microservice $v_{n,m}$ on the container c^i_g is given by Eq. 14:

$$\tau^{com}_{n,m,g,i}(t)=\begin{cases} s^{g,i}_{n,1}(t)\,\dfrac{a_{n,1}(t)\,\chi_{n,1}}{\xi^i_g}, & m=1\\ s^{g,i}_{n,m}(t)\,\dfrac{a^{dep}_{n,m}(t)\,\delta_{enc}\chi_{n,m}}{\xi^i_g}, & \text{Otherwise,}\end{cases} \tag{14}$$

where $\chi_{n,m}$ denotes the computational complexity of microservice $v_{n,m}$.

2.6 Total execution delay of microservices

Therefore, the total execution delay of the M microservices of the device b_n is denoted as the sum of the end–edge microservice offloading delay, the edge–edge microservice migration delay, and the microservice data queuing and computing delay, which is given by Eq. 15:

$$\tau^{sum}_n(t)=\tau^{off}_{n,g}(t)+\sum_{m=1}^{M-1}\sum_{g=1}^{G}\sum_{i=1}^{I}\tau^{tra}_{n,m,g,i}(t) +\sum_{g=1}^{G}\sum_{i=1}^{I}\left(\tau^{que}_{n,M,g,i}(t)+\tau^{com}_{n,M,g,i}(t)\right). \tag{15}$$

The total execution delay for all devices is given by Eq. 16:

$$\tau^{tot}(t)=\sum_{n=1}^{n=N}\tau^{sum}_n(t). \tag{16}$$

3 Optimization problem formulation

In this paper, we consider the optimization problem of microservice container selection for multi-flow aggregated energy dispatch service in a distribution grid. The optimization objective is to minimize the time-averaged total execution delay over T slots. The optimization problem is formulated as Eq. 17:

$$\begin{aligned}&\mathbf{P1}:\ \min_{\left\{s^{g,i}_{n,m}(t)\right\}}\ \frac{1}{T}\sum_{t=1}^{T}\frac{1}{N}\tau^{tot}(t)\\ &\text{s.t.}\quad C_1:\sum_{g=1}^{G}\sum_{i=1}^{I}s^{g,i}_{n,m}(t)=1,\forall v_{n,m}\in\mathcal{V}_n,\forall c^i_g\in\mathcal{C}_g,\forall t\in\mathcal{T},\\ &\qquad C_2: s^{g,i}_{n,m}(t)\le z^{g,i}_{n,m},\forall v_{n,m}\in\mathcal{V}_n,\forall c^i_g\in\mathcal{C}_g,\forall t\in\mathcal{T},\\ &\qquad C_3:\sum_{n=1}^{N}s^{g,i}_{n,m}(t)\le num^{\max}_{g,i},\forall v_{n,m}\in\mathcal{V}_n,\forall c^i_g\in\mathcal{C}_g,\forall t\in\mathcal{T},\end{aligned} \tag{17}$$

where C_1 and C_2 are the constraints for container selection, which mean that microservice $v_{n,m}$ can only select one container, and the selected container can process the type of microservice $v_{n,m}$. C_3 is the constraint of the maximum number of microservices that a container can handle, which means that container c^i_g can handle at most $num^{\max}_{g,i}$ microservices in each time slot.

4 The Microservice container selection algorithm based on the enhanced ant colony with empirical SINR and delay awareness

The formulated optimization problem of a microservice container selection is NP-hard due to the following twofold couplings. First, there exist complicated interdependencies between consecutive microservices generated by the same devices. The execution of the subsequent microservice requires the dependency data of the preceding microservice, which depends on both the SINR performance of edge–edge microservice migration and the computing capacity of the container. Therein, solving the container selection problem needs to consider the execution order of multiple microservices. Second, the microservices generated by different devices but processed in the same container are also intertwined. The queuing delay experienced by a microservice depends on not only container computing capacity but also the number of microservices that rank before it. The solution requires balancing these competitive relationships among the microservices generated by different devices. Moreover, the presence of uncertainties such as electromagnetic interference, noise, and container load incurs significant performance fluctuations. As a result, it becomes infeasible to directly apply conventional optimization techniques relying on convex modeling and certain global knowledge. It is intuitive to address this complexity by levering more intelligent and flexible heuristic algorithms.

The ant colony algorithm is an intelligent heuristic algorithm that simulates the foraging behavior of ant colonies in nature. It offers advantages such as distributed computing capacity, high parallelism, and adaptability, making it suitable for applications in microservices container selection. However, the traditional ant colony algorithm suffers from large randomness and relatively slow convergence speeds, which has a larger tendency to fall into a local optimum. Therefore, it is difficult to use the traditional ant colony algorithm to find the global optimal solution efficiently in complex microservice selection issues with interdependencies between consecutive microservices of the same device and coupling among microservices of different devices processed in the same container.

In response to the aforementioned issues, we propose a microservice container selection algorithm based on the enhanced ant colony with empirical SINR and delay performance awareness, as shown in Figure 2. Historical average metrics, including end-to-edge migration SINR and queuing and computing delays of containers, are amalgamated to form the empirical performance profile. This profile dynamically updates the heuristic information, improving both the search efficiency and convergence speed of the ant colony. Consequently, it avoids selecting containers with poor link conditions and high computational burdens. Moreover, the algorithm combines both local and global pheromone updating mechanisms. Locally, the incorporation of empirical SINR within the pheromone matrix diminishes the likelihood of selecting containers with inferior migration performance and mitigates the risk of falling into local optima. Globally, the insights derived from the empirically optimal solution are disseminated to all ants through global pheromone updating, thereby facilitating enhanced exploration capabilities in subsequent iterations.

4.1 Rules for path selection considering pheromone and heuristic information

Let $\mathcal{A} = \{Ant_1, \ldots, Ant_l, \ldots, Ant_L\}$ represent the set of L ants, in which Ant_l represents the l-th ant. In this paper, "path" refers to the edge container selected by the ant for microservice processing. $\phi_{g,i}^{n,m}(t)$ and $\eta_{g,i}^{n,m}(t)$ are defined as the pheromone and heuristic information on the path from $v_{n,m}$ to c_g^i, which are used to guide the ants to select the appropriate path. The influence factors of $\phi_{g,i}^{n,m}(t)$ and $\eta_{g,i}^{n,m}(t)$ are denoted as α and β, which reflect their relative importance degrees. A threshold $\sigma_0 \in (0,1)$ is set as a state transfer factor. Through the comparison of a random number $\sigma \in (0,1)$ with σ_0, we design two rules for path selection, which are as follows.

Rule 1: when $\sigma \leq \sigma_0$, select the path corresponding to the maximum product of $\phi_{g,i}^{n,m}(t)$ and $\eta_{g,i}^{n,m}(t)$ considering respective influence factors, which is given by Eq. 18:

$$\hat{s}_{n,m}^{g,i}(t) = \underset{\left\{s_{n,m}^{g,i}(t)\right\}}{\arg\max} \left\{ \left[\phi_{g,i}^{n,m}(t)\right]^{\alpha} \left[\eta_{g,i}^{n,m}(t)\right]^{\beta} \right\}, \tag{18}$$

where $\hat{s}_{n,m}^{g,i}(t)$ represents the optimal solution of this iteration.

Rule 2: when $\sigma > \sigma_0$, select the path according to the probability distribution, which is given by Eq. 19:

$$P\left[s_{n,m}^{g,i}(t) = 1\right] = \begin{cases} \dfrac{\left[\phi_{g,i}^{n,m}(t)\right]^{\alpha}\left[\eta_{g,i}^{n,m}(t)\right]^{\beta}}{\sum_{c_g^i \in \mathcal{C}_{n,m}^o}\left[\phi_{g,i}^{n,m}(t)\right]^{\alpha}\left[\eta_{g,i}^{n,m}(t)\right]^{\beta}}, & c_g^i \in \mathcal{C}_{n,m}^o, \\ 0, & \text{others}, \end{cases} \tag{19}$$

where $\mathcal{C}_{n,m}^o = \{c_g^i | z_{n,m}^{g,i} = 1\}$ is the set of optional paths available for microservice $v_{n,m}$, i.e., the set of containers that can process the type of microservice $v_{n,m}$.

4.2 Heuristic information updating with empirical performance awareness

We design a heuristic information updating scheme based on empirical performance awareness. We first introduce the calculation of empirical performance of edge–edge SINR and the empirical performance of queuing and computing delay. On this basis, a dynamic heuristic information updating scheme is proposed to dynamically update the empirical expectations of microservices assigned to containers and then guide the microservice to select the best containers according to the empirical expectations, which effectively reduces the number of iterations of the ant colony algorithm and prevents ants from falling into the local optimum.

4.2.1 Empirical performance of SINR and delay

Due to the variation in electromagnetic interference and container loads, it is difficult to accurately predict parameters such as SINR, queuing delay, and computing delay. $\widetilde{SINR}_{\hat{g},g}^{n,m}(t)$ is defined as the historical average edge–edge SINR from the edge server $w_{\hat{g}}$, where the preceding microservice $v_{n,m-1}$ is processed to w_g, where the subsequent microservice $v_{n,m}$ is processed. $\tilde{\tau}_{g,i}^{que}(t)$ and $\tilde{\tau}_{g,i}^{com}(t)$ are defined as the historical average queuing delay and computing delay of all the microservices processed in container c_g^i. $\widetilde{SINR}_{\hat{g},g}^{n,m}(t)$ and $\tilde{\tau}_{g,i}^{que}(t) + \tilde{\tau}_{g,i}^{com}(t)$ are given by the following equations:

$$\widetilde{SINR}_{\hat{g},g}^{n,m}(t) = \frac{1}{t-1}\sum_{r=1}^{t-1} SINR_{\hat{g},g}^{n,m}(r), \tag{20}$$

$$\tilde{\tau}_{g,i}^{que}(t) + \tilde{\tau}_{g,i}^{com}(t) = \sum_{r=1}^{t-1} \frac{\sum_{n=1}^{N}\sum_{m=1}^{M} s_{n,m}^{g,i}(r)\left(\tau_{n,m,g,i}^{que}(r) + \tau_{n,m,g,i}^{com}(r)\right)}{(t-1)\sum_{n=1}^{N}\sum_{m=1}^{M} s_{n,m}^{g,i}(r)}, \tag{21}$$

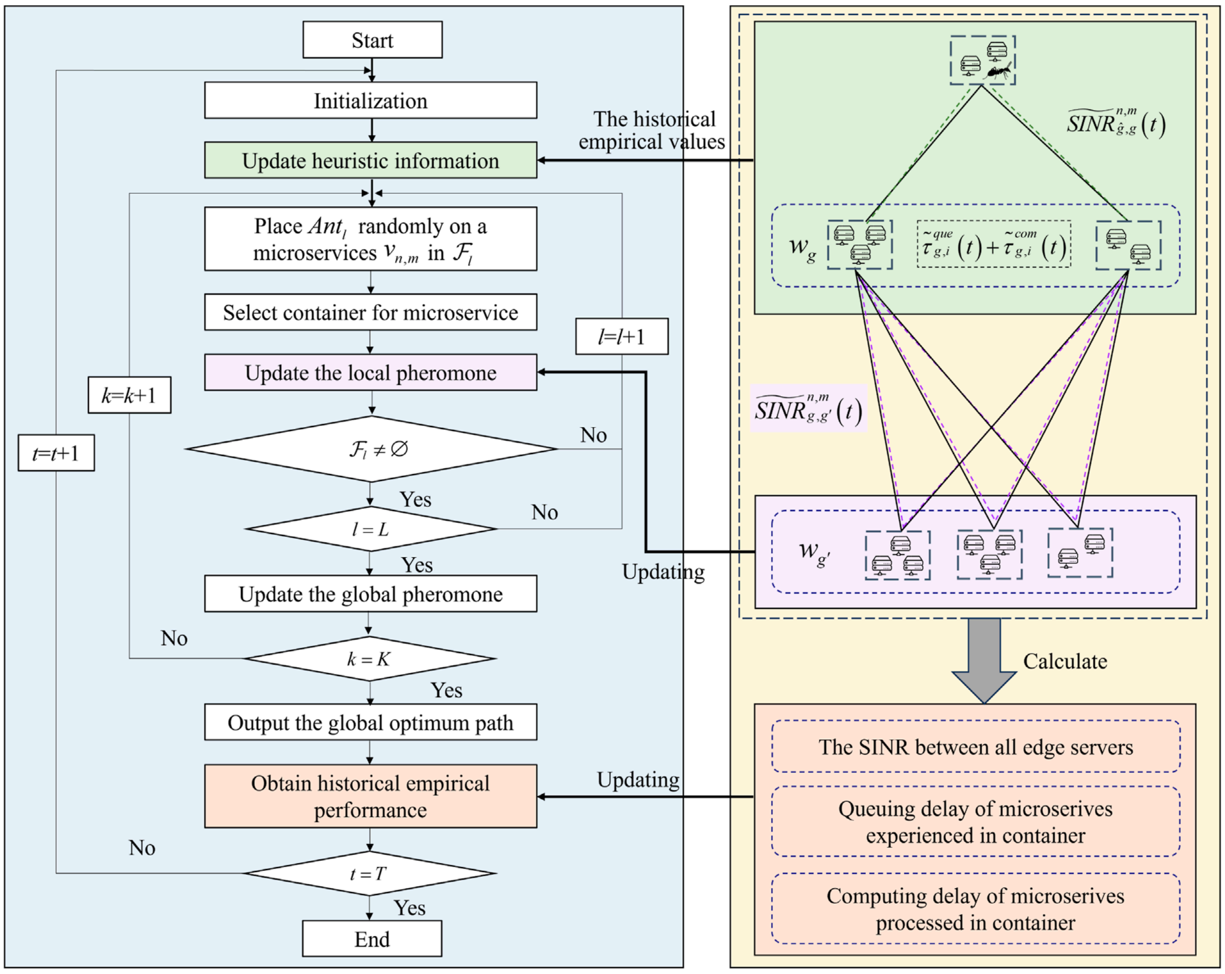
Start
Initialization
Update heuristic information
Place Ant_l randomly on a microservices $v_{n,m}$ in $\mathcal{F}_l$
Select container for microservice
Update the local pheromone
$\mathcal{F}_l \neq \varnothing$
$l = L$
Update the global pheromone
$k = K$
Output the global optimum path
Obtain historical empirical performance
$t = T$
End
Yes
No
l=l+1
k=k+1
t=t+1
The historical empirical values
Updating
Calculate
$\widetilde{SINR}_{\hat{g},g}^{n,m}(t)$
$\widetilde{SINR}_{g,g'}^{n,m}(t)$
w_g
$w_{g'}$
$\tilde{\tau}_{g,i}^{que}(t)+\tilde{\tau}_{g,i}^{com}(t)$
The SINR between all edge servers
Queuing delay of microserives experienced in container
Computing delay of microserives processed in container

FIGURE 2
Microservice container selection algorithm based on the enhanced ant colony with empirical SINR and delay performance awareness.

where $\sum_{n=1}^{N}\sum_{m=1}^{M} s_{n,m}^{g,i}(r)\left(\tau_{n,m,g,i}^{que}(r)+\tau_{n,m,g,i}^{com}(r)\right)$ represents the total queuing and computing delay of the microservices that select container c_g^i.

4.2.2 Dynamic heuristic information updating based on empirical performance

Heuristic information is an estimate that measures the merit of a path, which provides insights about the path quality that help ants make decisions along with the guidance of the pheromone. Once ants do not have enough pheromone, the heuristic factor becomes the main basis for decision making in path selection rules. It can help ants to search effectively in the local space and avoid falling into local optimal solutions. The heuristic information $\eta_{g,i}^{n,m}(t)$ signifies the expectation of ants regarding assigning the microservice $v_{n,m}$ to container c_g^i. Based on Eqs 18, 19, a larger $\eta_{g,i}^{n,m}(t)$ represents a more preferred path for ants. In order to achieve the minimization of the total execution delay of microservices in dynamic environments, the heuristic information needs to be adaptively updated based on the historical experience. It solves the effects of poor convergence and low stability of heuristic information updating on an iteration-by-iteration basis and also provides reliable empirical expectations for the microservice selection of containers, thus realizing the awareness of SINR and delay. The dynamic heuristic information $\eta_{g,i}^{n,m}(t)$ is updated as follows:

$$\eta_{g,i}^{n,m}(t) = \omega_{SINR}\cdot\widetilde{SINR}_{\hat{g}g}^{n,m}(t) + \frac{\omega_{que+com}}{\tilde{\tau}_{g,i}^{que}(t)+\tilde{\tau}_{g,i}^{com}(t)}, \tag{22}$$

where ω_{SINR} and $\omega_{que+com}$ denote the weights assigned with the empirical performance of SINR and the empirical performance of queuing and computing delay, respectively.

4.3 Local and global integrated pheromone updating

Pheromone is a chemical released by ants on a path to transmit information. It acts as a collective memory in the ant colony algorithm, which records the experience of the ant colony during the searching process. Ants tend to choose paths with higher pheromone concentrations. Thus, when an ant on a path discovers a high-quality solution, it releases more pheromone, leading other ants to be more likely to choose the same path. In the container selection problem, we define $\phi_{g,i}^{n,m}(t)$ as the pheromone on the path from $v_{n,m}$ to c_g^i, and a larger $\phi_{g,i}^{n,m}(t)$ means that ants prefer to choose the container c_g^i for the microservice $v_{n,m}$ based on Eqs 18, 19. In the process of iterative ant colony path searching, we introduce a method that involves both local and global integrated pheromone updating to balance local and global explorations and accelerate convergence speed.

4.3.1 Local pheromone updating

Local pheromone is updated after an ant has traveled a path, and the ant has the capability to induce the evaporation of pheromones along this path. Local pheromone is updated to reduce the probability of repeatedly selecting the same path, thereby mitigating the risk of entrapment in a local optimal solution. In order to adjust the evaporation amount of the local pheromone, we combine the updating with edge–edge SINR considering subsequent microservice migrations. $\mathcal{C}_{n,m+1}^o = \{c_g^i | z_{n,m+1}^{g,i} = 1\}$ is defined as the optional container set for microservice $v_{n,m+1}$, and $|\mathcal{C}_{n,m+1}^o|$ is defined as the number of optional containers. Since the subsequent microservice v_{m+1} may have a lower average SINR as it completes the migration process from w_g to $w_{g'} \in \mathcal{C}_{n,m+1}^o$, it will evaporate more pheromone $\phi_{g,i}^{n,m}(t)$ along the path. Thus, the detailed equation for localized pheromone updating is as follows:

$$\phi_{g,i}^{n,m}(t) = \left(1 - \frac{\rho_1 \cdot y|\mathcal{C}_{n,m+1}^o|}{\sum\limits_{c_{g'}^i \in \mathcal{C}_{n,m+1}^o} \widetilde{SINR}_{g,g'}^{n,m}(t)}\right) \cdot \phi_{g,i}^{n,m}(t) + \rho_1 \phi_0, \tag{23}$$

where $\widetilde{SINR}_{g,g'}^{n,m}(t)$ represents the historical average edge–edge SINR from w_g selected by the current microservice $v_{n,m}$ to $w_{g'}$ that can be selected by the subsequent microservice $v_{n,m+1}$. The calculation of $\widetilde{SINR}_{g,g'}^{n,m}(t)$ is similar to that in Eq. 20. $\rho_1 \in [0,1]$ is the local pheromone evaporation parameter, ϕ_0 is the initial pheromone content, and y is a scaling factor.

4.3.2 Global pheromone updating

Global pheromone is updated after all ants in $\mathcal{A}$ have traveled and selected their own paths, i.e., completing an iteration of training. The global pheromone is updated. Global pheromone acts as a synergistic guide between ant colonies. The ants are able to interact and act synergistically in the searching space by sharing the global pheromone. The updating of the global pheromone helps accelerate the convergence of the ants to the global optimal solution. Upon completing an iteration of training, a global optimal path planning strategy with the smallest total microservice execution delay is chosen, and the global pheromone on this optimal path is increased, which is given by the following equation:

$$\phi_{g,i}^{n,m}(t) = (1-\rho_2)\phi_{g,i}^{n,m}(t) + \rho_2 \Delta\phi_{g,i,\text{best}}^{n,m}(t), \tag{24}$$

where ρ_2 is the global pheromone updating parameter. $\Delta\phi_{g,i}^{n,m}(t)$ is the increment of the pheromone on the global optimal path, which is given by the following equation:

$$\Delta\phi_{g,i,\text{best}}^{n,m} = \frac{\frac{1}{N}\tau_{\text{best}}^{tot}(t)}{NM}, \tag{25}$$

where $\frac{1}{N}\tau_{\text{best}}^{tot}(t)$ represents the minimum total microservices execution delay for all ants in set $\mathcal{A}$.

4.4 Algorithm process

The implementation process of the proposed microservice container selection algorithm based on the enhanced ant colony with empirical SINR and delay performance awareness is shown in Algorithm 1. We assume that there are a total of K training iterations, and for one iteration, each ant in $\mathcal{A}$ should select paths for all microservices in $\mathcal{V}_n$. The allowed list for Ant_l is defined as $\mathcal{F}_l$, which indicates that microservices in $\mathcal{F}_l$ are waiting for Ant_l to select paths for them. The tabu list for Ant_l is defined as $\widehat{\mathcal{F}}_l$, which indicates that microservices in $\widehat{\mathcal{F}}_l$ have finished path selection. The procedures of the proposed algorithm are detailed below.

1) *Initialization:* initialize $\mathcal{B}$, $\mathcal{V}_n$, $\mathcal{W}$, $\mathcal{C}_g$, and $\{z_{n,m}^{g,i}\}$, and set $s_{n,m}^{g,i}(t) = 0$.

2) *Global optimal path execution over slots:* at the beginning of slot$t$, calculate the historical average SINR $\widetilde{SINR}_{\tilde{g},g}^{n,m}(t)$ between all edge servers, the historical average queuing delay $\tilde{\tau}_{g,i}^{que}(t)$, and computing delay $\tilde{\tau}_{g,i}^{com}(t)$ of all microservices processed in container c_g^i based on Eqs 20, 21. Through these calculations of empirical performance, heuristic information $\eta_{g,i}^{n,m}(t)$ is updated based on Eq. 22. Then, we start the iterative training of the global optimal solution searching in procedure 3. Finally, the devices and edge servers execute the global optimal path selection strategy $\{s_{n,m}^{g,i}\}$ obtained in the K-th iteration and obtain the SINR $SINR_{\tilde{g},g}^{n,m}(t)$ between all edge servers as well as the queuing delay and computing delay.

3) *Global optimal path searching over iterations:* at the beginning of the k-th iteration, we initialize the allowed list $\mathcal{F}_l = \emptyset$ and tabu list $\widehat{\mathcal{F}}_l = \emptyset$ for all ants in $\mathcal{A}$. Then, we perform the ant colony optimization in procedure 4. Through the comparison of all the ants' total microservice execution delay, we obtain the best path with the minimum delay $\frac{1}{N}\tau_{\text{best}}^{tot}(t)$ and calculate the increment of the pheromone on the global optimal path $\Delta\phi_{g,i}^{n,m}(t)$ based on Eq. 25. On this basis, the global optimal path of the k-th iteration is derived, and the global pheromone is updated based on Eq. 24.

4) *Path selection through ant colony optimization:* ants in $\mathcal{A}$ sequentially perform path selection for all microservices. For ant Ant_l, it adds the first microservice of all devices waiting to be offloaded into $\mathcal{F}_l$. Ant_l is placed randomly on a microservice, e.g., $v_{n,m}$ in $\mathcal{F}_l$ as the starting point, and then a container c_g^i is selected for the microservice $v_{n,m}$ based on the rules of Eqs 18, 19. Once path selection is completed, the local pheromone on the selected path is updated based on Eq. 23. Then, the current microservice $v_{n,m}$ is taken out of $\mathcal{F}_l$ and put into the tabu list $\widehat{\mathcal{F}}_l$. In addition, the subsequent microservice $v_{n,m+1}$ is added into the allowed list $\mathcal{F}_l$. When all the microservices have completed their path selection, i.e., $\mathcal{F}_l = \emptyset$, the total microservice execution delay of Ant_l is calculated.

4.5 Convergence analysis

From the perspective of training iterations, ants search for optimal paths based on the local and global integrated pheromone updating method. On one hand, the update of the local pheromone serves to alleviate the risk of being trapped in the local optimality. On the other hand, the global pheromone provides positive incentives for exploration. Through their combination, the proposed algorithm exhibits a stronger ability to search for the global optimum in a large

```
1: Initialization: initialize $\mathcal{B}$, $\mathcal{V}_n$, $\mathcal{W}$, $\mathcal{C}_g$, and $\{z_{n,m}^{g,i}\}$, and set $s_{n,m}^{g,i}(t)=0$.
2: For $t=1,\ldots,T$ do
    3: Calculate the historical average SINR between all edge servers based on Eq. 20.
    4: Calculate the historical average queuing delay and computing delay of all the microservices processed in container $c_g^i$ based on Eq. 21.
    5: Update heuristic information $\eta_{g,i}^{n,m}(t)$ based on Eq. 22.
    6: For $k=1,\ldots,K$ do
        7: Initialize $\mathcal{F}_l=\emptyset$, $\widehat{\mathcal{F}}_l=\emptyset$ for all ants in $\mathcal{A}$.
        8: For $l=1,\ldots,L$ do
            9: Add the first microservice of all devices waiting to be offloaded into $\mathcal{F}_l$.
            10: While $\mathcal{F}_l\neq\emptyset$
                11: Place $Ant_l$ randomly on a microservice $v_{n,m}$ in $\mathcal{F}_l$ as a starting point.
                12: Select a container for the current microservice $v_{n,m}$ based on Eqs 18, 19.
                13: Update the local pheromone based on Eq. 23.
                14: Take the current microservice $v_{n,m}$ out of $\mathcal{F}_l$ and put it into the tabu list $\widehat{\mathcal{F}}_l$.
                15: Add the subsequent microservice $v_{n,m+1}$ into the allowed list $\mathcal{F}_l$.
            16: End while
            17: Calculate the total microservice execution delay of $Ant_l$.
        18: End for
        19: Compare the total microservice execution delay of ants in set $\mathcal{A}$, and derive the global optimal path.
        20: Update the global pheromone based on Eqs 24, 25.
    21: End for
    22: Execute the global optimal path selection strategy obtained in the $K$-th iteration.
    23: Obtain the SINR between all edge servers as well as the queuing delay and computing delay of all microservices processed in container $c_g^i$ in slot $t$.
24: End for
```

Algorithm 1. Microservice container selection algorithm based on the enhanced ant colony with empirical SINR and delay awareness.

solution space and avoid falling into a local optimum. From the perspective of slots, empirical SINR, queuing delay, and computing delay are calculated at the beginning of each slot based on the information obtained by executing the container selection strategy at the end of the last time slot. Then, the heuristic information used to perform path selection in this slot is updated. Dynamic iterative heuristic information updating provides better direction for optimal path searching. In addition, compared with conventional ant colony algorithm, both pheromone updating and heuristic information updating leverages empirical performance as a key basis, which effectively accelerates the convergence speed and avoids the performance decay due to the variation of the environment.

4.6 Complexity analysis

The complexity of the proposed algorithm is mainly related to the procedures of path selection, pheromone updating, and heuristic information updating. For path selection, the complexity for an ant selecting one path for all devices' microservices is $O(NM)$. Since there are total K training iterations in one slot and L ants in each iteration, the complexity of path selection in one slot is $O(NMLK)$. For pheromone updating, the complexity consists of two parts, i.e., local pheromone updating along with each path selection and global pheromone updating at the end of one iteration. Global pheromone updating relies on the calculation of the total microservices execution delay of an ant and their comparison result. Thus, the complexity of pheromone updating in one slot is $O(NMLK+NML^2K)$. Heuristic information is updated at the beginning of one slot based on the calculation of empirical edge–edge SINR, queuing delay, and computing delay. Therefore, the complexity of heuristic information updating is $O(G(G-1)/2+2GI)$. Based on the above analysis, the complexity of the proposed algorithm is $O(NMLK(2+L)+G(G+4I-1)/2)$. Compared with the conventional ant colony algorithm with the complexity $O(2NMLK)$, the proposed algorithm sacrifices complexity slightly for better convergence performance and stronger learning adaptability.

5 Simulation results

In this section, we first theoretically analyze the proposed scheme from the perspectives of privacy, fairness, and security. Then, we validate the proposed scheme by simulation in a specific scenario.

5.1 Privacy, fairness, and security analysis

5.1.1 Privacy protection

The devices and edge servers generate key pairs, with the public and private keys used for encryption and decryption purposes. This encryption ensures that intercepted data are incomprehensible to attackers without access to the corresponding private key. Furthermore, communication between devices and edge servers is conducted through blinded signatures, maintaining their anonymity within the network.

5.1.2 Fairness

According to the devised smart contract, any edge server engaging in the "free-ride" attack will face penalties, and instances of microservice offloading and migration failure are recorded in the blockchain. Furthermore, entities of the "double-claim" and repudiation attacks will likewise be penalized. Consequently, only honest entities are eligible for rewards, with successful microservice offloading and migration events being recorded.

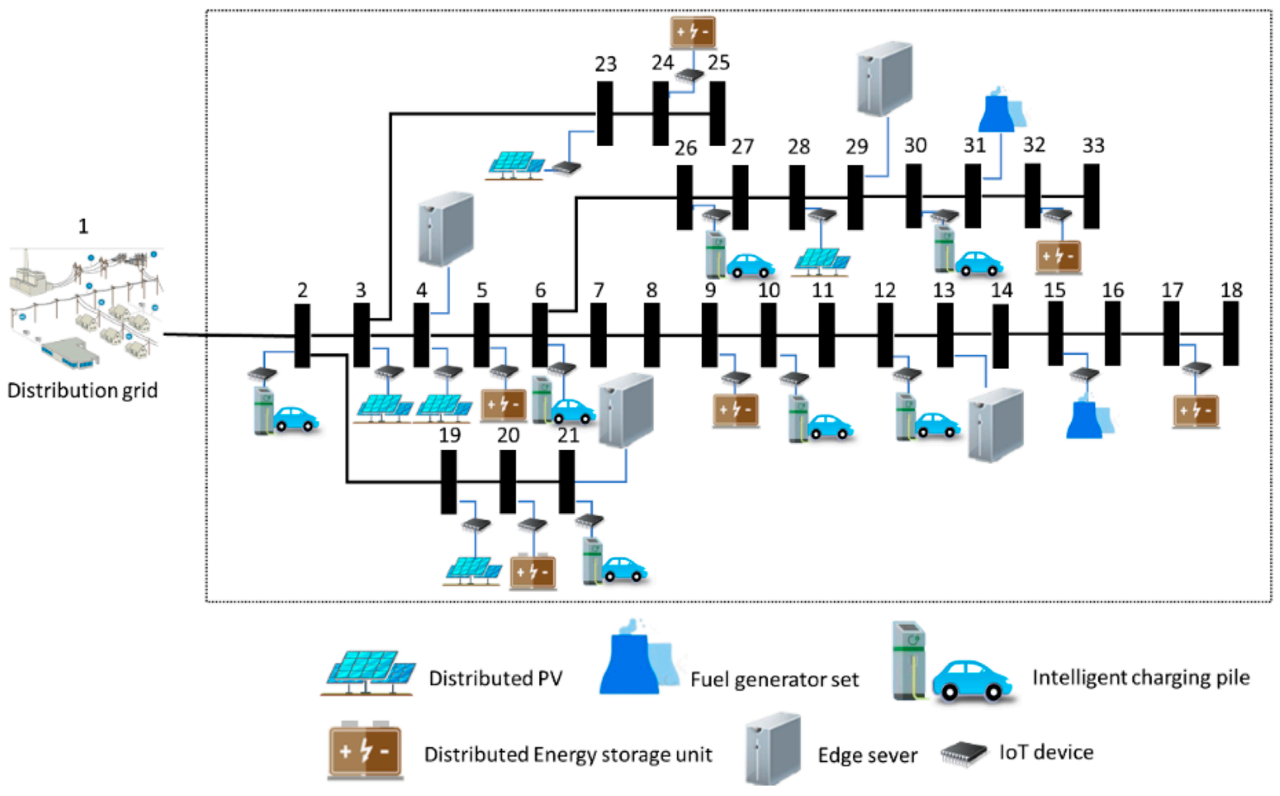

FIGURE 3
IEEE-33-node power distribution system.

5.1.3 Security

In this section, we will conduct a security analysis of three common blockchain attack threats: the “double-claim” attack, the “free-ride” attack, and the “repudiation” attack. These attacks are chosen due to their relevance to the microservices computing process based on the blockchain framework proposed in this paper.

1) *Security against “double-claim” attack:* upon the receipt of $ROOT(m_3)$, the smart contract automatically triggers transaction settlement, preventing any malicious device or edge server from claiming rewards multiple times.
2) *Security against “free-ride” attack:* in the event that a malicious edge server fails to allocate adequate computational resources, it will be unable to provide the Merkle hash root value $ROOT(m_2)$ to the computation component within the stipulated timeframe. Consequently, the transaction is automatically terminated, and the edge server faces penalties for the failure in microservice offloading or migration. Alternatively, if the malicious edge server generates $ROOT(m_2)$ without performing actual data computation, resulting in equality with $ROOT(m_1)$, it will be penalized for malicious behavior, with the failure event being recorded in the blockchain.
3) *Security against “repudiation” attack:* at the outset of the microservice offloading and migration process, all three entities must contribute to a currency deposit pool. The smart contract design ensures that these currencies cannot be returned to a malicious base station’s wallet until the transaction settlement is complete. Additionally, if a device attempts to deny the contributions of an edge server, it must submit a $ROOT(m_3)$ distinct from $ROOT(m_2)$ to the computation component. In such cases, the device is ineligible for rewards, contravening the rationality assumption of devices.

5.2 Simulations

The IEEE-33-based node distribution grid, as shown in Figure 3, is selected for simulations, which contains distributed PV, fuel generator sets, intelligent charging piles, distributed energy storage units, IoT devices, and edge servers. The simulation parameters are shown in Table 1 (Khapre et al., 2020, Tariq et al., 2020, Zhu et al., 2024).

Two state-of-art algorithms are employed for comparison. The first algorithm is the containerized microservices deployment approach based on the ant colony (CMAC), which adopts conventional ant colony algorithm (Lee et al., 2023). The second algorithm is the upper confidence bound-based microservices deployment approach (UCBM), which achieves more rational exploration and improves resource utilization by considering microservice priorities and resolving conflicts between differentiated microservices (Deng et al., 2023). Neither CMAC nor UCBM considers the empirical performance of SINR between the edge servers as well as

TABLE 1 Simulation parameters.

Parameter	Value	Parameter	Value
N	20	M	10
G	4	I	6
T	200	$B_{n,g}$	1 MHz
δ	−114 dBm	$\alpha_{n,g}$	1.4
$\beta_{n,g}$	−0.791	$\xi_{n,g}$	10^{-4}
$\mu_{n,g}$	[1.2, 2.6]	χ_{enc}	3×10^6 cycles/Mbits
χ_{dig}	2×10^6 cycles/Mbits	χ_{sig}	3×10^6 cycles/Mbits
δ_{enc}	0.8	$\chi_{n,m}$	[2, 10] $\times10^6$ cycles/Mbits
ξ_g^i	$[1\times10^8, 3\times10^8]$ CPU cycles/s	$P_n^{tran}(t)$	$[0.1, 0.4]$ W
$B_{g,g'}$	1.2 MHz	$num_{g,i}^{\max}$	8
ρ_1	0.8	ρ_2	0.1
K	200	L	200

queuing and computing delay of microservices processed in containers.

Figure 4 illustrates the average service execution delay versus iteration under different algorithms. The average execution delay refers to $\frac{1}{N}\tau^{tot}(t)$. When algorithms iterate 2,000 times, the proposed algorithm reduces the average service execution delay by 23.64% and 41.93% compared with CMAC and UCBM, respectively. The proposed algorithm takes into account the empirical performance of SINR, queuing delay, and computing delay in heuristic information updating to minimize total delay, resulting in the best convergence performance. Additionally, the proposed algorithm updates local pheromones differently based on the empirical performance of SINR, which significantly speeds up the convergence by avoiding low-quality links.

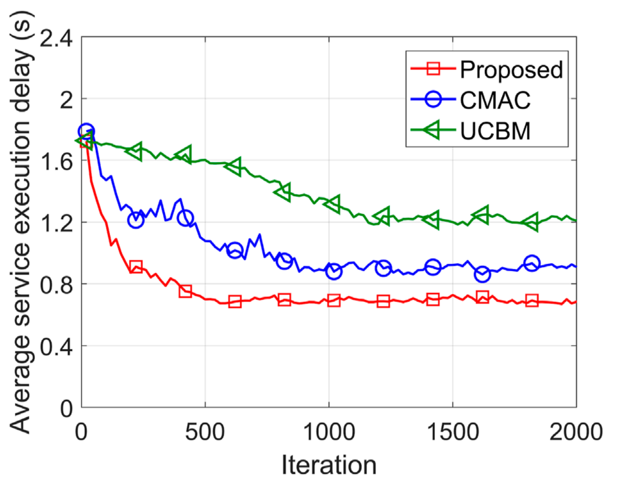

FIGURE 4
Average service execution delay under different algorithms.

Figure 5 shows the edge server computing workload with the edge server under different algorithms. The computing load of the edge server of the proposed algorithm is the most balanced. The reason is that the proposed algorithm can make full use of containers with relatively poor computing resources but better link conditions and less computing workload to reduce migration and queuing delay. In contrast, CMAC and UCBM focus solely on computing and aggressively select containers with rich computing resources to reduce computing delay, which results in the unbalanced computing workload of edge servers.

Figures 6, 7 illustrate the average service execution delay versus the number of devices and the microservice computational complexity, respectively. With the increase of the number of devices and computational complexity, it is evident that the average service execution delays of the three algorithms are significantly enhanced. On one hand, as the number of devices increases, the total number of microservices to be processed also increases, resulting in higher queuing and migration delay. On the other hand, the rise in computational complexity leads to an increase in computing delay. When the number of devices is 26, the proposed algorithm reduces the average service execution delay by 31.18% and 52.33% compared to CMAC and UCBM, respectively. When the computational complexity is 10×10^6 cycles/Mbits, the proposed algorithm reduces the average service execution delay by 34.34% and 50.1%, respectively. This is attributed to the proposed algorithm's consideration of the empirical performance of queuing delay and SINR of container selection, effectively mitigating the increase in queuing delay and migration delay. Furthermore, due to the balanced computing workload of the edge server, the increase in

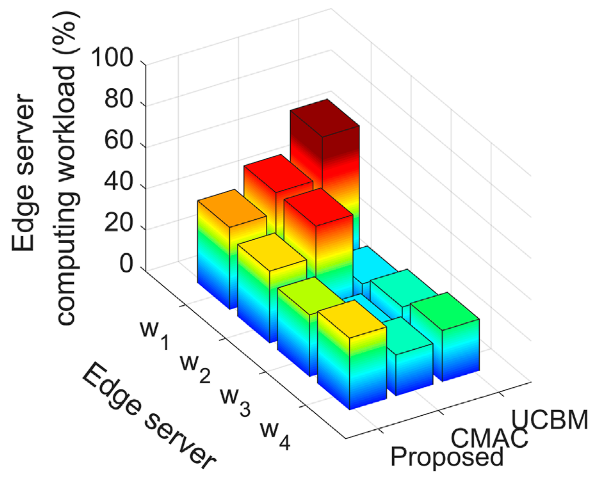

FIGURE 5
Edge server computing workload under different algorithms.

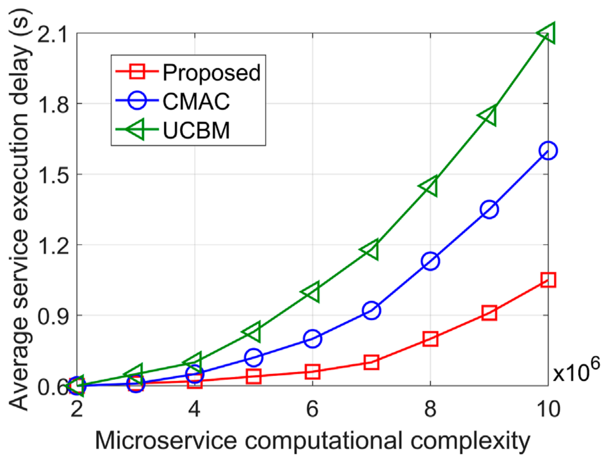

FIGURE 7
Average service execution delay versus microservice computational complexity.

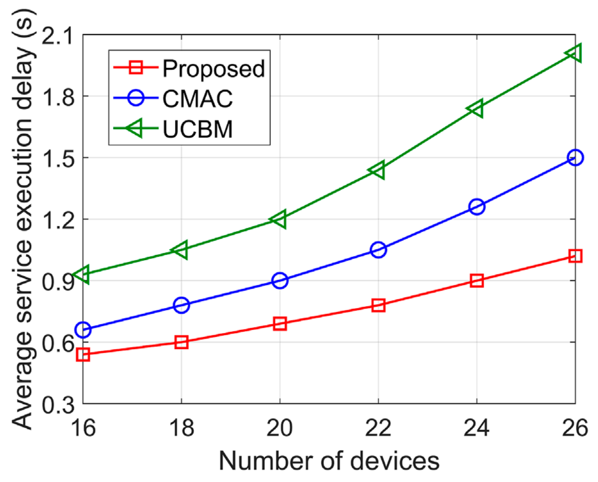

FIGURE 6
Average service execution delay versus the number of devices.

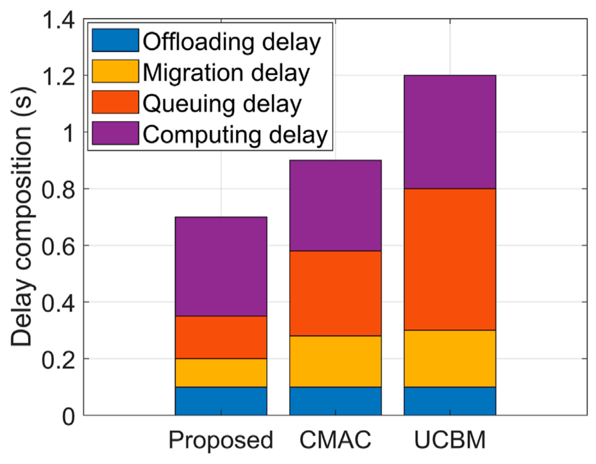

FIGURE 8
Delay composition of service execution delay under different algorithms.

computing complexity has less influence on the computing delay of the proposed algorithm.

Figure 8 compares the composition of service execution delay under different algorithms. It can be seen that the offloading delays of the three algorithms are the same because the offloading delay is mainly affected by the first microservice. Although the proposed algorithm exhibits a higher computing delay compared to CMAC and UCBM, it effectively reduces the queuing delay by 41.23% and 67.45% and migration delay by 44.63% and 61.32%, respectively. CMAC and UCBM only account for the computing delay in container selection, leading to a large migration delay. Furthermore, aggressive selection of containers with more computing resources causes microservices to accumulate, resulting in increased queuing delay. The proposed algorithm's consideration of workload balancing may result in the selection of a container with subpar computing resources, leading to a calculation delay that is not the lowest. However, it achieves optimal service execution delay performance by significantly reducing the migration and queuing delays.

Figures 9, 10 represent the average service execution delay and migration delay under sudden electromagnetic interference, respectively. Particularly, in the 80th time slot, the electromagnetic interference on a specific edge–edge microservice migration link is suddenly increased. Initially, the average service execution delay and migration delay of the three algorithms sharply decreased and stabilized after five time slots. However, after 80 slots, the sudden increase in electromagnetic interference led to a notable rise in the migration delay and average service execution delay for all three algorithms. In the 200th time slot, compared to CMAC and UCBM, the proposed algorithm reduces the average service execution delay by 25.44% and 46.34%, respectively, and reduces the migration delay by 52.96% and 72.01%, respectively. Notably, the proposed

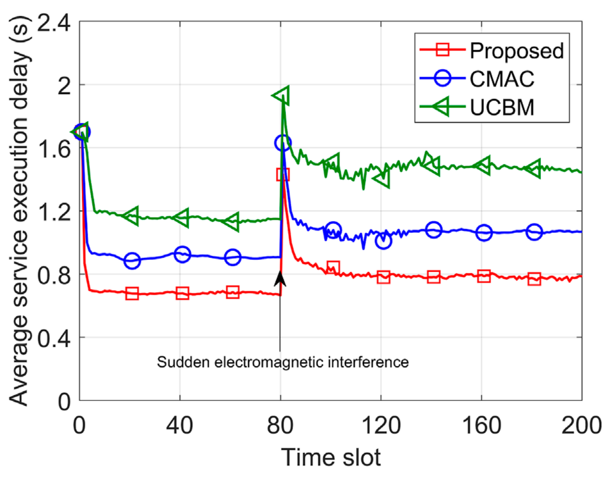

FIGURE 9
Average service execution delay under sudden electromagnetic interference.

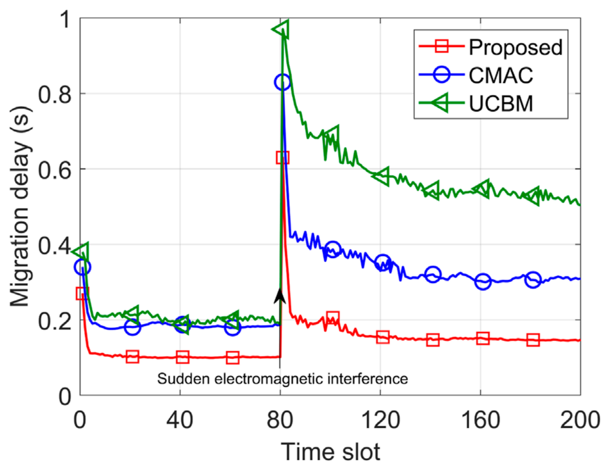

FIGURE 10
Migration delay under sudden electromagnetic interference.

algorithm considers the empirical performance of SINR and can avoid choosing the edge–edge microservice migration channel with large electromagnetic interference, resulting in the smallest increase in the average service execution delay and migration delay and the fastest convergence speed. UCBM is sensitive to the initial estimate, and it takes a longer time to converge when the electromagnetic interference changes suddenly, performing the worst among the three algorithms.

6 Conclusion

In this paper, we studied the edge–end collaborative secure and rapid response method for multi-flow aggregated energy dispatch service in the distribution grid. In response to the optimization problem of microservice container selection, we proposed a microservice container selection algorithm based on the enhanced ant colony with empirical SINR and delay performance awareness to minimize the time-averaged total execution delay. This proposed algorithm, building upon the traditional ant colony algorithm, integrates the historical average performance of edge–edge migration SINR and delay of queuing and computing to obtain more accurate heuristic information updating. It also combines local and global integrated pheromone updating, enhancing the searching efficiency and convergence speed. The simulation results demonstrate that compared to CMAC and UCBM algorithms, the proposed algorithm reduces the average service execution delay by 23.64% and 41.93%, respectively, and shows faster convergence and more balanced workloads. **Q2–2:** In forthcoming research, we will explore integrated sensing, transmission, and computing services aligned with comprehensive environmental sensing, cloud-edge–end management, and intelligent data processing, while also examining the security implications of IoT devices connecting to edge servers via 5G, with the goal of optimizing container selection and enabling automated energy dispatch.

Data availability statement

The raw data supporting the conclusion of this article will be made available by the authors, without undue reservation.

Author contributions

ZS: writing–original draft and writing–review and editing.

Funding

The author declares that financial support was received for the research, authorship, and/or publication of this article. This work was supported by the China Southern Power Grid technology project under Grant number 03600KK52220019 (GDKJXM20220253).

Conflict of interest

Author ZS was employed by the Power Dispatching and Controlling Center of Guangdong Power Grid Company Limited.

The authors declare that this study received funding from China Southern Power Grid technology project. The funder was involved in the study design, collection, analysis, interpretation of data, the writing of this article, and the decision to submit it for publication.

Publisher's note

References

Al-Debagy, O., and Martinek, P. (2018). "A comparative review of microservices and monolithic architectures," in 2018 IEEE 18th International Symposium on Computational Intelligence and Informatics (CINTI), Budapest, Hungary, 21-22 November 2018, 149–154.

Buzato, F. H. L., Goldman, A., and Batista, D. (2018). "Efficient resources utilization by different microservices deployment models," in 2018 IEEE 17th International Symposium on Network Computing and Applications (NCA), Cambridge, MA, USA, 1-3 November 2018, 1–4.

Cabrera, C., Svorobej, S., Palade, A., Kazmi, A., and Clarke, S. (2023). Maaco: a dynamic service placement model for smart cities. *IEEE Trans. Serv. Comput.* 16, 424–437. doi:10.1109/TSC.2022.3143029

Chhikara, Tekchandani, R., Kumar, N., and Obaidat, M. S. (2021). An efficient container management scheme for resource-constrained intelligent IoT devices. *IEEE Internet Things J.* 8 (16), 12597–12609. doi:10.1109/jiot.2020.3037181

Deng, X., Li, B., Li, X., Wu, Z., and Yang, Z. (2023). "Container and microservice-based resource management for distribution station area," in 2023 5th International Conference on Intelligent Control, Measurement and Signal Processing (ICMSP), Chengdu, China, 19th to 21st May 2023, 578–581.

Dong, M., Ota, K., and Liu, A. (2016). RMER: reliable and energy-efficient data collection for large-scale wireless sensor networks. *IEEE Internet Things J.* 3 (4), 511–519. doi:10.1109/jiot.2016.2517405

Gao, Chen, M., Liu, A., Ip, W. H., and Yung, K. L. (2020). Optimization of microservice composition based on artificial immune algorithm considering fuzziness and user preference. *IEEE Access* 8, 26385–26404. doi:10.1109/access.2020.2971379

Han, P., Liu, Y., and Guo, L. (2021). Interference-aware online multicomponent service placement in edge cloud networks and its ai application. *IEEE Internet Things J.* 8, 10557–10572. doi:10.1109/jiot.2020.3048832

Khapre, S. P., Chopra, S., Khan, A., Sharma, P., and Shankar, A. (2020). "Optimized routing method for wireless sensor networks based on improved ant colony algorithm," in 2020 10th International Conference on Cloud Computing, Data Science and Engineering (Confluence), Noida, India, 29-31 January 2020, 455–458.

Lee, W.-T., Yang, Z.-Y., Liu, Z.-W., and Lee, S.-J. (2023). "Containerized microservices deployment approach based on ant colony optimization," in 2023 10th International Conference on Dependable Systems and Their Applications (DSA), Tokyo, Japan, August 10-11, 2023, 472–473.

Li, Q., Li, S., Song, X., Shen, Z., Xue, C., Xing, Y., et al. (2022). "Research on key technologies of multi-objective coordinated optimal operation of source, network and load of new power distribution system," in 2022 IEEE 6th Conference on Energy Internet and Energy System Integration (EI2), Chengdu, China, Oct. 28th to 30th, 2022, 3099–3104.

Li, W., Li, X., and Ruiz, R. (2021). "Scheduling microservice-based workflows to containers in on-demand cloud resources," in 2021 IEEE 24th International Conference on Computer Supported Cooperative Work in Design (CSCWD), Pune, India, 5-7 May 2021, 61–66.

Lyu, Z., Wei, H., Bai, X., and Lian, C. (2020). Microservice-based architecture for an energy management system. *IEEE Syst. J.* 14, 5061–5072. doi:10.1109/jsyst.2020.2981095

Meshram, D. K., Goel, N., and Chacko, S. (2022). "Integration of battery energy storage system with solar power generation system along with load management system," in 2022 International Conference for Advancement in Technology (ICONAT), Goa, India, 21-22 January 2022, 1–8.

Mota, A. V., Azam, S., Shanmugam, B., Yeo, K. C., and Kannoorpatti, K. (2017). "Comparative analysis of different techniques of encryption for secured data transmission," in 2017 IEEE International Conference on Power, Control, Signals and Instrumentation Engineering (ICPCSI), Chennai, India, 21-22 Sepember 2017, 231–237.

Naik, A., Choudhari, J., Pawar, V., and Shitole, S. (2021). "Building an edtech platform using microservices and docker," in 2021 IEEE Pune Section International Conference (PuneCon), Pune, India, 16-19 December 2021, 1–6.

Shen, Z., Ding, F., Yao, Y., Bhardwaj, A., Guo, Z., and Yu, K. (2023). A privacy-preserving social computing framework for health management using federated learning. *IEEE Trans. Comput. Soc. Syst.* 10 (4), 1666–1678. doi:10.1109/tcss.2022.3222682

Shunxin, L., Min, Z., Wenhai, N., Yinan, Z., Xin, L., and Shaoqiao, D. (2022). "Research on the smart grid development needs of power distribution network under and comprehensive electric energy use system," in 2022 IEEE 2nd International Conference on Power, Electronics and Computer Applications (ICPECA), Shenyang, China, 21-23 January 2022, 849–851.

Tan, A., Ma, H., Mei, Y., and Zhang, M. (2022). A cooperative coevolution genetic programming hyper-heuristics approach for on-line resource allocation in container-based clouds. *IEEE TCC* 10 (3), 1500–1514. doi:10.1109/tcc.2020.3026338

Tang, Zhou, X., Zhang, F., Jia, W., and Zhao, W. (2019). Migration modeling and learning algorithms for containers in fog computing. *IEEE T Serv. Comput.* 12 (5), 712–725. doi:10.1109/tsc.2018.2827070

Tariq, M., Adnan, M., Srivastava, G., and Poor, H. V. (2020). Instability detection and prevention in smart grids under asymmetric faults. *IEEE Trans. Ind. Appl.* 56 (4), 1–4520. doi:10.1109/tia.2020.2964594

Tariq, M., Naeem, F., Ali, M., and Poor, H. V. (2021). Vulnerability assessment of 6G enabled smart grid cyber-physical systems. *IEEE Internet Things J.* 8 (7), 5468–5475. doi:10.1109/jiot.2020.3042090

Tariq, M., and Poor, H. V. (2018). Electricity theft detection and localization in microgrids. *IEEE Trans. Smart Grid* 9, 1920–1929. doi:10.1109/TSG.2016.2602660

Wang, Y., Ding, P., Shen, Y., Shi, X., and Zhou, H. (2022). "Fine-grained object tracking system infrastructure based on cloud-edge collaboration," in 2022 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB), Bilbao, Spain, June 15 -17, 2022, 1–5.

Wu, J., Dong, M., Ota, K., Li, J., and Guan, Z. (2019). FCSS: fog-computing-based content-aware filtering for security services in information-centric social networks. *IEEE Trans. Emerg. Top. Comput.* 7 (4), 553–564. doi:10.1109/tetc.2017.2747158

Wu, J., Dong, M., Ota, K., Li, J., and Guan, Z. (2018). Big data analysis-based secure cluster management for optimized control plane in software-defined networks. *IEEE Trans. Netw. Serv. Manage.* 15 (1), 27–38. doi:10.1109/tnsm.2018.2799000

Xue, H., Chen, D., Zhang, N., Dai, H., and Yu, K. (2023). Integration of blockchain and edge computing in internet of things: a survey. *Future Gener. comput. Syst.* 144, 307–326. doi:10.1016/j.future.2022.10.029

Yang, J., Lin, G., Lv, R., Gao, C., and Chen, T. (2020). "Research on construction and dispatching of virtual power plant based on reserve energy storage of communication base station," in 2020 IEEE 4th Conference on Energy Internet and Energy System Integration (EI2), Wuhan, China, 30 October 2020 - 01 November 2020, 398–403.

Zhou, H., Wang, Z., Zheng, H., He, S., and Dong, M. (2023a). Cost minimization-oriented computation offloading and service caching in mobile cloud-edge computing: an A3C-based approach. *IEEE Trans. Netw. Sci. Eng.* 10 (3), 1326–1338. doi:10.1109/tnse.2023.3255544

Zhou, H., Zhang, Z., Wu, Y., Dong, M., and Leung, V. C. M. (2023b). Energy efficient joint computation offloading and service caching for mobile edge computing: a deep reinforcement learning approach. *IEEE Trans. Green Commun. Netw.* 7 (2), 950–961. doi:10.1109/tgcn.2022.3186403

Zhou, Z., Yang, X., and Xu, C. (2016). "Performance evaluation of multi-antenna based m2m communications for substation monitoring," in 2016 International Conference on Information and Communication Technology Convergence (ICTC), Jeju, Korea (South), 19-21 October 2016, 97–102.

Zhu, X., Ma, F., Ding, F., Guo, Z., Yang, J., and Yu, K. (2024). A low-latency edge computation offloading scheme for trust evaluation in finance-level artificial intelligence of things. *IEEE Internet Things J.* 11 (1), 114–124. doi:10.1109/jiot.2023.3297834