frontiers | Frontiers in Energy Research

TYPE Original Research
PUBLISHED 18 January 2024
DOI 10.3389/fenrg.2024.1358310

OPEN ACCESS

EDITED BY
Yuqing Dong,
The University of Tennessee, Knoxville, United States

REVIEWED BY
Lu Guanpeng,
Guangdong Water Conservancy and Electric Power Survey and Design Institute Co., Ltd., China
Wei Qiu,
Hunan University, China

*CORRESPONDENCE
Jing Qu,
epquj27@mail.scut.edu.cn

RECEIVED 19 December 2023
ACCEPTED 05 January 2024
PUBLISHED 18 January 2024

CITATION
Hu K, Qu J, Cai Z, Li X, Liu Y and Zheng J (2024), Service scheduling strategy for microservice and heterogeneous multi-cores-based edge computing apparatus in smart girds with high renewable energy penetration.
Front. Energy Res. 12:1358310.
doi: 10.3389/fenrg.2024.1358310

Service scheduling strategy for microservice and heterogeneous multi-cores-based edge computing apparatus in smart girds with high renewable energy penetration

Kaiqiang Hu, Jing Qu*, Zexiang Cai, Xiaohua Li, Yuanyuan Liu and Junjie Zheng[1]

[1]School of Electric Power Engineering, South China University of Technology, Guangzhou, China

The microservice-based smart grid service (SGS) organization and the heterogeneous multi-cores-based computing resource supply are the development direction of edge computing in smart grid with high penetration of renewable energy sources and high market-oriented. However, their application also challenges the service schedule for edge computing apparatus (ECA), the physical carrier of edge computing. In the traditional scheduling strategy of SGS, an SGS usually corresponds to an independent application or component, and the heterogeneous multi-core computing environment is also not considered, making it difficult to cope with the above challenges. In this paper, we propose an SGS scheduling strategy for the ECA. Specifically, we first present an SGS scheduling framework of ECA and give the essential element of meeting SGS scheduling. Then, considering the deadline and importance attributes of the SGS, a microservice scheduling prioritizing module is proposed. On this basis, the inset-based method is used to allocate the microservice task to the heterogeneous multi-cores to utilize computing resources and reduce the service response time efficiently. Furthermore, we design the scheduling unit dividing module to balance the delay requirement between the service with early arrival time and the service with high importance in high concurrency scenarios. An emergency mechanism (EM) is also presented for the timely completion of urgent SGSs. Finally, the effectiveness of the proposed service scheduling strategy is verified in a typical SGS scenario in the smart distribution transformer area.

KEYWORDS

smart grid services, edge computing apparatus, service scheduling, microservice, heterogeneous multi-cores

1 Introduction

With the widespread integration of renewable energy sources and the increasing marketization of the smart grid, the number and variety of smart grid service (SGS) at the edge of the smart grid, such as the smart distribution transformer area—typically functioning as the smallest unit of power supply management (Cen et al., 2022; Xiao, H.

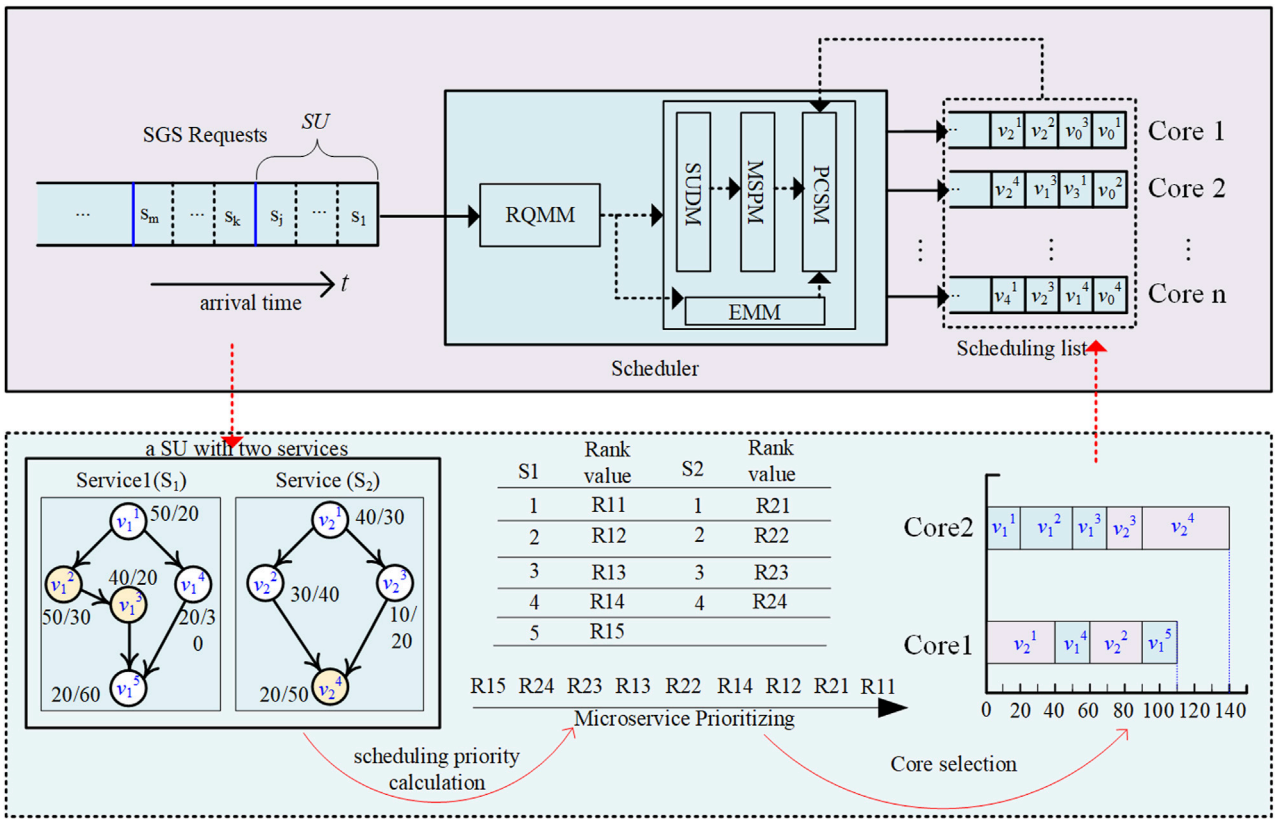

FIGURE 1
The microservice scheduling framework for ECA.

et al., 2023)—are expanding. These services encompass a diverse range, including Vehicle-to-Grid (V2G) (Chukwu and Mahajan, 2014; Chamola et al., 2020; Peng and Niu, 2023), micro-grid energy management and control (Mondal et al., 2022; Wu et al., 2022; Xiao et al., 2023), and demand response at the user side (Bachoumis et al., 2022; Jia et al., 2022). New demands are subsequently placed on edge computing apparatus (ECA), a key computing node at the edge side of the smart grid supporting the processing of the SGS (Li et al., 2021; 2022). On the one hand, these large numbers of SGSs require more efficient and powerful computing capabilities for the edge computing node (Li et al., 2022). On the other hand, the services deployed on ECA require a flexible service organization to meet their needs for flexibility and rapid iteration (Zhou et al., 2021). The heterogeneous multi-cores-based computing resource supply and microservice-based service organization can meet the abovementioned requirements, hence emerging as the development direction of ECA in smart grids (Zhang et al., 2019; Jiang et al., 2020).

In the ECA, an SGS consists of multiple independent microservices and can be realized by collaborating these microservices (Lyu et al., 2020; Yin et al., 2022). These microservices have different computational characteristics, such as matrix computation operations, digital signal processing operations, power message encapsulation, and encryption and parsing operations. During the processing, microservices with different computational characteristics have different computing speeds on different cores of the ECA (Lan et al., 2022). For example, the microservice task for message parsing is processed much faster on a specially customized FPGA core than on a general-purpose core such as a CPU. Besides, some microservices cannot execute in certain cores due to processing core instruction set dependencies. Subsequently, a key question arises regarding how the ECA can schedule the SGS task to different cores for better scheduling performance. However, very little literature in the smart grid area carries out a study of the above issues. The traditional smart grid apparatus with the function of edge computing, such as TTU, DTU, and other IED (Intelligent Electronic Device), carries a solidified and limited number of services, and the development and deployment of services is in the form of individual applications (Wojtowicz et al., 2018). The service scheduling is usually fixed during the development session for the apparatus. Thus, it is difficult to be applied to the ECA.

From the mathematical form, SGSs scheduling for ECA belongs to the problem of scheduling a set of microservices with dependencies in a heterogeneous system, which is an NP-C problem (Sahni et al., 2021; Roy et al., 2023). There are three general solutions: heuristic-based list scheduling algorithms, random search-based intelligent algorithms, and machine learning-based methods. Some effective heuristic-based list scheduling algorithms have been proposed, such as HEFT (Topcuoglu, Hariri, and Min-You Wu, 2002), PEFT (Arabnejad and Barbosa, 2014), CPOP (Kelefouras and Djemame, 2022). For the random search-based scheduling strategy, microservice scheduling is established as an optimization problem that is solved using intelligent algorithms such as the Genetic Algorithm (Rehman et al., 2019), Ant Colony Algorithm (Gao et al., 2019), and Particle Swarm Algorithm (Rodriguez and Buyya, 2014). In recent years, with the rapid development and application of AI

TABLE 1 Parameters of the SGSs.

Services (with its importance and deadline)	Microsevices	Execution time on each core (milliseconds)	DAG structure information
Plug-play service in the SDTA (importance = 5, deadline = 3,700 m)	Primary device registration packet analysis	[100,120,20,∞]	$\begin{bmatrix}0&0&1&0\\0&0&1&0\\0&0&0&1\\0&0&0&0\end{bmatrix}$
	Secondary device response packet analysis	[50,120,50,∞]	
	SDTA mapping	[300,400,∞,200]	
	Return the successful registration packet	[50,100,25,∞]	
Topology identification service in the SDTA (importance = 3, deadline = 4,600 m)	Voltage pulse packet analysis	[100,120,50,∞]	$\begin{bmatrix}0&1&1&0\\0&0&0&1\\0&0&0&1\\0&0&0&0\end{bmatrix}$
	Voltage fluctuation similarity calculation	[200,250,∞,100]	
	Voltage fluctuation association calculation	[300,250,∞,150]	
	Topology relationship analysis	[200,250,∞,∞]	
Loop resistance supervision service in the SDTA (importance = 4, deadline = 3,950 m)	Voltage packet analysis	[100,120,50,∞]	$\begin{bmatrix}0&0&1&0\\0&0&1&0\\0&0&0&1\\0&0&0&0\end{bmatrix}$
	Current packet analysis	[100,120,50,∞]	
	Loop resistance matrix calculation	[300,200,∞,100]	
	Operating condition evaluation	[200,250,∞,∞]	
Line loss analysis PIoT service in the SDTA (importance = 5 deadline = 6,700 m)	Line loss packet analysis	[100,120,50,∞]	$\begin{bmatrix}0&0&1&0&0\\0&0&1&0&0\\0&0&0&1&0\\0&0&0&0&1\\0&0&0&0&0\end{bmatrix}$
	Load packet analysis	[100,120,50,∞]	
	Stationary test	[300,400,∞,100]	
	Granger test	[300,400,∞,100]	
	Electricity-theft identification	[150,300,∞,∞]	
Grid-connection management of photovoltaic generation service in the SDTA (importance = 1, deadline = 11,150 m)	Photovoltaic generation capacity packet analysis	[100,120,∞,∞]	$\begin{bmatrix}0&1&1&0&0&0&0&0&0&0&0\\0&0&0&1&0&1&0&0&0&0&0\\0&0&0&1&0&1&0&0&0&0&0\\0&0&0&0&1&0&0&1&0&0&0\\0&0&0&0&0&0&0&0&0&0&1\\0&0&0&0&0&0&1&0&0&0&0\\0&0&0&0&0&0&0&0&0&0&1\\0&0&0&0&0&0&0&0&0&1&0\\0&0&0&0&0&0&0&0&0&0&1\\0&0&0&0&0&0&0&0&0&0&0\end{bmatrix}$
	Current packet analysis	[100,120,∞,∞]	
	Voltage packet analysis	[150,180,50,∞]	
	Current harmonic distortion calculation	[150,180,50,∞]	
	Harmonic elimination device control	[150,200,∞,∞]	
	Voltage deviation calculation	[250,200,∞,130]	
	Reactive power compensation control	[100,200,∞,∞]	
	Voltage fluctuation calculation	[250,200,∞,130]	
	Battery storage control	[150,200,∞,∞]	
	Photovoltaic generation access control	[200,300,∞,∞]	
Electric vehicle charging management service in SDTA (importance = 2, deadline = 6,900 m)	Electricity price packet analysis	[100,120,50,∞]	$\begin{bmatrix}0&0&0&1&0\\0&0&0&1&0\\0&0&0&1&0\\0&0&0&0&1\\0&0&0&0&0\end{bmatrix}$
	Electric vehicle charging packet analysis	[100,120,50,∞]	
	Load packet analysis	[100,120,50,∞]	
	Charging period optimization	[600,800,∞,300]	
	Electric vehicle charging control	[150,220,∞,∞]	

(Continued on following page)

TABLE 1 (*Continued*) Parameters of the SGSs.

Services (with its importance and deadline)	Microsevices	Execution time on each core (milliseconds)	DAG structure information
Price-based demand response aggregated service (importance = 1, deadline = 4,150 m)	Load packet analysis	[50,70,25,∞]	$\begin{bmatrix} 0&0&1&1&0&0&0\\ 0&0&1&1&0&0&0\\ 0&0&0&0&1&0&0\\ 0&0&0&0&0&1&0\\ 0&0&0&0&0&1&0\\ 0&0&0&0&0&0&1\\ 0&0&0&0&0&0&0 \end{bmatrix}$
	Real-time electricity price packet analysis	[50,70,25,∞]	
	Time-of-use electricity price packet analysis	[100,140,∞,∞]	
	Load forecast	[100,40,∞,∞]	
	Elastic matrix calculation	[150,50,∞,40]	
	Demand response calculation	[150,200,∞,100]	
	Load control	[50,120,∞,∞]	

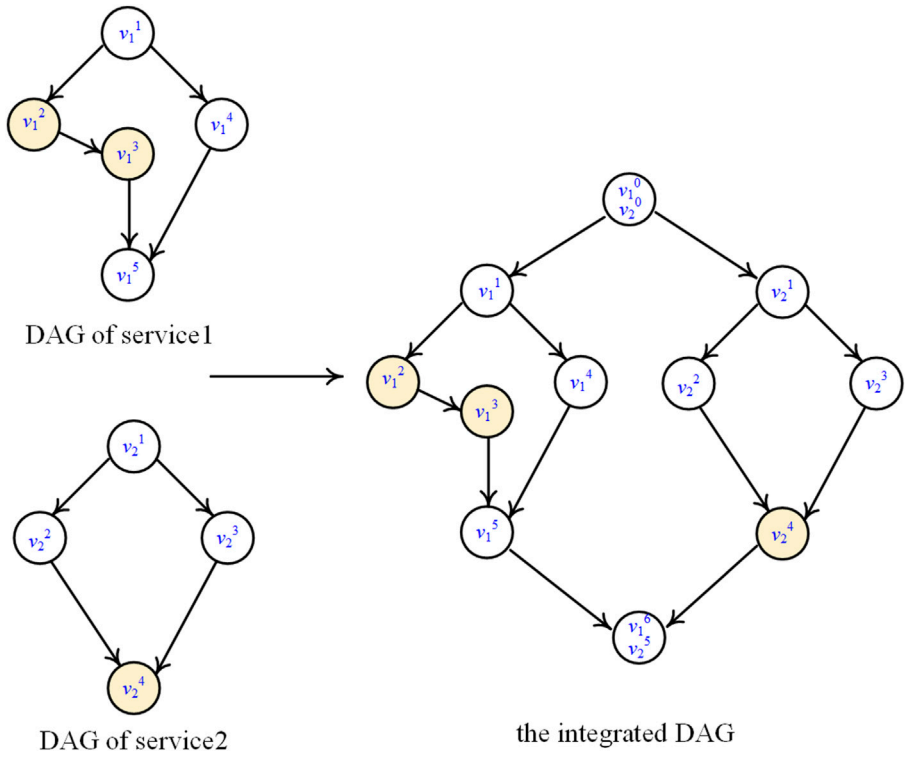

FIGURE 2
The illustration for multi-services integration.

technology, machine learning-based methods have also been applied to the study of this problem, such as deep reinforcement learning (Gao and Feng, 2022) and deep Q-learning (Kaur et al., 2022). The above literature mainly addresses the service scheduling problem for scientific workflow in distributed computing system environments, such as grid and cloud computing environments. In these scenarios, the service workflow is typically non-real-time tasks, and the type and number of services to be scheduled are fixed. However, in the SGS scheduling problem for the ECA, service requests arrive dynamically and concurrently in real-time, and different SGSs have different importance and quality of service requirements. In addition, the state of service request may change with the operational state of the smart grid and some needs to be finished before the deadline in emergencies (Li et al., 2018).

In this paper, we propose an ECA-oriented microservice scheduling strategy based on the inspiration of list scheduling to address the issues above and fill the gap in ECA-related research fields. The main contributions of this paper are summarized as follows:

(1) A services scheduling framework is proposed for the ECA in the smart grid. The problem of service scheduling is divided into two sub-problems: *microservice prioritizing* and *core*

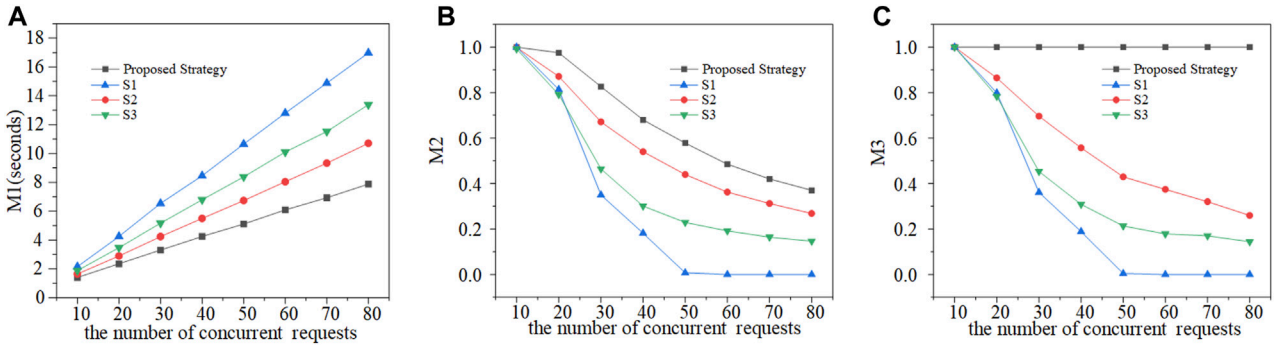

FIGURE 3
Effect of changing the number of concurrent requests **(A)** on M1, **(B)** on M2 for different strategy, and **(C)** on M3.

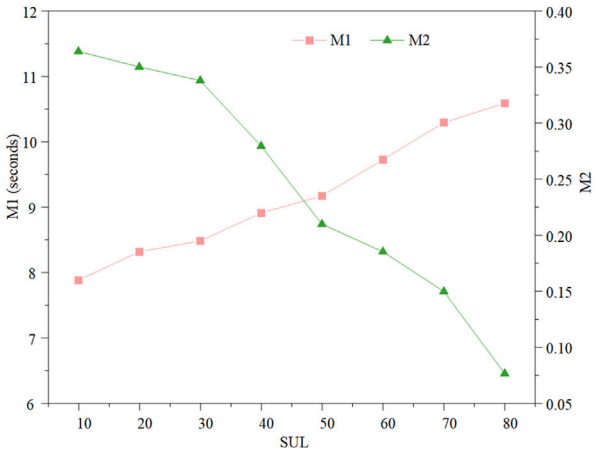

FIGURE 4
Effect of changing *SUL* on M1 and M2.

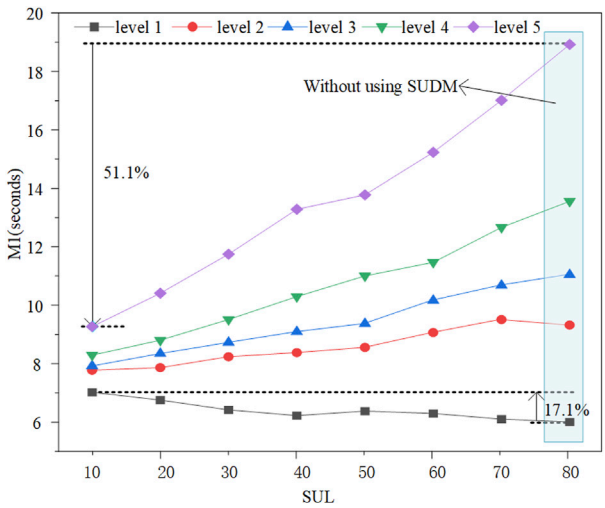

FIGURE 5
Effect of changing *SUL* on M1 for the SGSs with different importance.

selection. *Microservice prioritizing* can determine the scheduling order of microservices for meeting the execution order constraints of microservices in a service. Then, the *core selection* can allocate the microservice to a proper core for execution. Two novel mechanisms, namely, *scheduling unit dividing mechanism* and *emergency mechanism*, are also integrated into the framework, where the former is used to balance the delay requirement between the service with early arrival time and the service with high importance in high concurrency scenarios; and the latter is responsible for the timely completion of urgent SGSs.

(2) A novel SGS scheduling strategy is proposed. The SGS scheduling models are built, including the service and ECA models. The SGS is represented by a directed acyclic graph (DAG), and the ECA is modeled as a heterogeneous system with multiple cores. We design a microservice scheduling prioritizing module considering the SGS attributes of importance and deadline to determine the microservices scheduling sequence. Then, the insert policy is introduced to allocate the microservices to heterogeneous cores of ECA. In addition, both scheduling unit dividing and emergency solutions are developed and integrated into the SGS scheduling algorithm.

(3) Extensive simulations-based performance evaluation is conducted. Based on the ideas of solutions in existing works, several benchmark solutions are developed for performance comparison. Three metrics are used to evaluate the scheduling performance, and the performance comparisons are conducted in different levels of service concurrency. The simulation results demonstrate that the proposed strategy is effective and superior for SGS scheduling of ECA. Furthermore, the influence analysis of the algorithm parameter *scheduling unit length* (*SUL*) is also performed for the parameter selection of the algorithm in practical applications.

The remainder of this article is organized as follows. Section 2 introduces the SGSs scheduling framework for ECA and outlines its basic operation principle. Section 2 introduces the SGS scheduling model for ECA, including the ECA and SGS models. Section 3 describes the proposed microservice scheduling strategy and gives the algorithm's pseudo-code. Section 4 evaluates and analyzes the performance of the proposed strategy. Section 5 concludes this article.

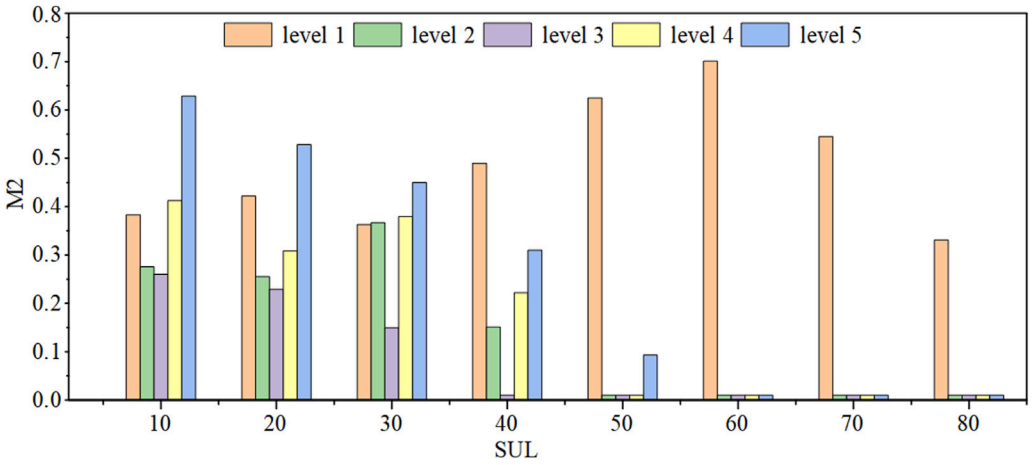

FIGURE 6
Effect of changing *SUL* on M2 for the SGSs with different importance.

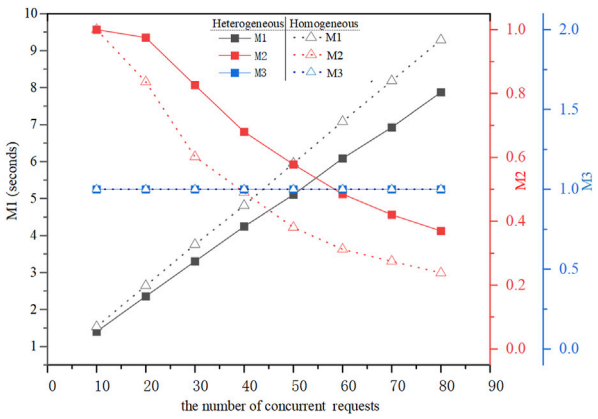

FIGURE 7
Performance comparison result.

2 The SGS scheduling framework for ECA

Figure 1 shows the proposed SGS scheduling framework for the ECA. As shown in Figure 1, the diagram is divided into two layers by boxes. The upper layer shows the main components of the scheduling framework, while the lower layer uses examples to illustrate the basic workflow of the scheduling framework. The upper layer has three parts: the service request queue, scheduler, and scheduling list. The services request queue storages the service request information. The scheduling list records the microservice sequence that a processing core needs to execute. The scheduler allocates services to the processing cores for execution according to a certain service strategy. There are several modules in the scheduler box. The request queue monitoring module (RQMM) collects request queue information, including queue length and service arrival time, and passes it to the scheduling unit dividing module (SUDM). The SUDM is responsible for dividing the service requests into multiple fixed-length request subgroups based on the arrival time of service, and a service request group is briefly denoted as a *scheduling unit* (*SU*). Each service request within an *SU* is sorted according to its importance, which is one of the attributes of the SGS. The microservice scheduling prioritizing module (MSPM) calculates the scheduling priority of each microservice in an *SU* through a microservice prioritizing algorithm (such as Algorithm 1 mentioned in Section 4.1), forming a microservice scheduling sequence. Then, based on a core section algorithm (such as Algorithm 1 mentioned in Section 4.2), the processing core selection module (PCSM) allocates each microservice to a processing core in turn. The algorithm needs to obtain scheduling queue information through module 1. When the RQMM finds an urgent request, it activates the emergency mechanism module (EMM). The ongoing normal service will be logged and paused at the time, and then the urgent service will be executed based on a predefined service scheduling scheme.

The lower layer illustrates the SGS scheduling process under normal conditions on an ECA with two processing cores, and the length of the SU is set to 2. The SGS is represented by DAG. The service is successfully completed when all the microservices are executed according to directed edge constraints.

3 Service scheduling model for ECA

In this section, we introduce the service scheduling model for ECA, including the ECA and SGS models. In addition, an evaluation metric model is also presented for service scheduling according to ECA's actual performance demand.

3.1 The ECA model

The ECA belongs to a heterogeneous multi-core system, which integrates different types and numbers of processing cores and can be modeled as a set $P = \{ p_1^{\#} \ p_2^{\#} \ \dots \ p_r^x \ \dots \}$, where the element p_r^x denotes the number x of core type r (x, $r \in Z+$). The cores with the same type have the same workload for the same microservice task. Conversely, the cores with different types have different workloads for the same microservices task. Additionally, some cores may only be

able to handle one or more specific microservices. Thus, the execution time of the microservice on the core p_r can be calculated by:

$$w_{i,r}^{j} = \frac{L_{ij,r}}{C_r} \tag{1}$$

where $L_{ij,r}$ denotes the workload of microservice j of service i on the core r. C_r represents the computation speed of core p_r.

The processing cores are interconnected via an on-chip high-speed bus. It can be assumed that the communication bandwidth among the cores is the same, and any two processing cores can communicate in both directions without contention (Roy et al., 2023). The communication bandwidth between the cores can be described by an adjacency matrix, and for an ECA with z cores:

$$A_z = \begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1z} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{z1} & a_{z2} & \ldots & a_{zz} \end{bmatrix} \tag{2}$$

where A_z is a symmetric matrix whose diagonal elements are infinite, and the non-diagonal elements a_{gh} represent the communication rate between the core g and the core p_h. Thus, the communication time between microservice j executed in core p_g and microservice k executed in core p_h is:

$$c_{jg,\mathrm{kh}} = \frac{d_{j,\mathrm{k}}}{a_{gh}} \tag{3}$$

where $d_{j,\mathrm{k}}$ denotes the data volume transmitted from microservice j to microservice k.

3.2 The SGS model

A set of SGSs in the SGSs request queue is denoted as $\mathrm{S} = \{\mathrm{S}_1,\mathrm{S}_2,\mathrm{S}_\mathrm{n}\}$, and n is the concurrent number of SGSs. An SGS $S_i = \{A_i, D_i, T_i, I_i, G_i\}$ contains several basic information. A_i and D_i represent the arrival time and deadline of the S_i; T_i represents the request state of service, which is a binary value, where 0 and 1 represent normal and urgent states, respectively; I_i denotes the importance of S_i, which is given based on expert experience. G_i is a DAG, $G_i = (V_i, E_i)$, where V is the set of v microservices, and E is the set of e edges between the microservices. The edge represents the data dependency between two microservices. The non-entry node microservice may have one or more inputs and is triggered to execute when all input data from the directly connected nodes are available. The node in a DAG with zero in-degree denotes the *entry microservice*, and the node with zero out-degree denotes the *exit microservice*. Suppose there are multiple exit microservices or entry microservices for a service DAG. In that case, they can be connected with zero time-weight edges to a single pseudo-exit task or a single entry task with zero time-weight. The microservice j of the S_i denotes v_i^j. In addition, there are several scheduling attributes for each microservice.

(1) The average execution time. It is the average value of the execution times required on different processor cores for a microservice. The average execution time of microservice v_i^j can be calculated by:

$$w_i^j = \frac{\sum_{r=1}^{p} w_{i,r}^{j}}{p} \tag{4}$$

where $w_{i,r}^{j}$ is the execution time of the microservice v_i^j on the core p_r, and p is the number of processing cores of ECA.

(2) The average communication time. It is the average value of the communication time between microservices on any two cores. The average communication time of microservice v_i^j can be defined by:

$$c_{i,k}^{j} = \frac{d_{i,k}^{j}}{\sum_{r=1}^{p}\sum_{s=r+1}^{p-1} A_{r,s} / \left((p^2 - p)/2\right)} \tag{5}$$

(3) The earliest execution start time (EST). The EST of microservice v_i^j in core p_r is defined by:

$$EST\left(v_i^j, p_r\right) = \max\left(\begin{array}{c} available\left(v_i^j, p_r\right), \\ \max\limits_{v_i^k \in pred\left(v_i^j\right)} \left(\mathrm{EFT}\left(v_i^k, p_l\right) + \mathrm{c}_{k,j,l,r}\right) \end{array} \right) \tag{6}$$

where $available\left(v_i^j, p_r\right)$ represents the earliest time that the core p_r can execute v_i^j. $pred\left(v_i^j\right)$ denotes the set of predecessor microservices for v_i^j.

(4) The earliest execution finish time (EFT). The EFT of microservice v_i^j in core p_r is defined by:

$$EFT\left(v_i^j, p_r\right) = w_{i,r}^{j} + EFT\left(\mathrm{v}_{i,r}^{j}\right) \tag{7}$$

3.3 Evaluation metric model

Generally, the average response time of services is a basic metric for service scheduling, and it can evaluate the overall scheduling performance. Aiming to the ECA in the smart grid, if these services can be completed before the deadline, they can get a better effect on the smart grid operation. Especially when the smart grid is in an emergency or unhealthy state, some services must be completed before the deadline to offer help for restoring normal operation as soon as possible. Otherwise, it will lead to serious safety accidents and economic losses. Thus, this study introduces the service meeting deadlines rate (SMDA) for the performance evaluation, including the SMDA of normal service and the SMDA of urgent service. The above three metrics are abbreviated as M_1, M_2, and M_3, and their definitions are formulated as follows.

(1) M_1 represents the average response time of services and can be calculated by:

$$M_1 = \sum_{i=1}^{n}\left(t_i^{finish} - t_i^{arrival}\right)/n \tag{8}$$

where n is the concurrent number of SGSs. t_i^{finish} and $t_i^{arrival}$ represent the finished time and the arrival time of the SGS s_i, respectively.

(2) M_2 represents the SMDA of normal service and can be calculated by:

$$\begin{aligned} M_2 &= \left(\sum_{s_i \in S\_n} s_i\right) \Big/ |S\_n| \\ T_i &= \begin{cases} 0, t_i^{finish} < D_i \\ 1, t_i^{finish} i \geq D_i \end{cases} \end{aligned} \tag{9}$$

where S_n is the set of normal service requests,$|S\_n|$ is the number the normal services.

(3) M_3 represents the SMDA of urgent service and can be calculated by:

$$M_3 = \left(\sum_{s_i \in S\_urgent} s_i \right) \Big/ |S\_urgent|$$
$$s_i = \begin{cases} 0, t_i^{finish} < D_i \\ 1, t_i^{finish} \geq D_i \end{cases} \tag{10}$$

where S_u is the set of urgent services requests, and $|S\_u|$ is the number the urgent services.

4 SGS scheduling strategy for ECA

In this section, the specific scheduling algorithms are presented based on the proposed framework.

4.1 Microservice prioritizing

Microservice prioritizing generates the priority of each microservice using a rank value, which is used to determine the execution order of microservices. The pseudo-code of the proposed microservice prioritizing method is shown in Algorithm 1, which mainly includes the following three steps.

Step 1: Calculate the average execution time and average communication time of a microservice v_i^j according to Eq. 1 and Eq. 2, respectively.

Step 2: Calculate the critical path length for the microservice. The critical path length for microservice is computed by:

$$cp\left(v_i^j\right) = w_j + \max_{v_k \in succ\left(v_i^j\right)} \left(c_{i,k} + cp\left(v_k\right)\right) \tag{11}$$

Step 3: Calculate the rank value of each microservice. The rank for a microservice v_i^j is computed by:

$$rank\left(v_j^i\right) = \frac{cp\left(v_j^i\right)}{cp\left(v_{\text{entry}}^i\right)} \Big/ \left(\tilde{I}^i + \tilde{Q}^i\right) \tag{12}$$

where the v_{entry}^i is the entry node of the service i, and it is defined as the critical path of the service i; $\tilde{I}^i$ *and* $\tilde{Q}^i$ represents the normalized value of importance of service i, *and* the normalized value *of* deadline of the service i.

$$\tilde{I}_i = \frac{I_i - \min\limits_{i \in I}\{I_i\}}{\max\limits_{i \in I}\{I_i\} - \min\limits_{i \in I}\{I_i\}} \tag{13}$$

$$\tilde{Q}_i = \frac{Q_i - \min\limits_{i \in I}\{Q_i\}}{\max\limits_{i \in I}\{Q_i\} - \min\limits_{i \in I}\{Q_i\}} \tag{14}$$

1: **For** each SGS deployed in ECA **do**
2: calculate the *CP* of the DAG of the SGS
3: **For** each microservice in the DAG **do**
4: calculate its average execution time according to Eq. (4)
5: calculate its average communication time according to Eq. (5)
6: calculate the *CP* of the microservice according to Eq. (11)
7: calculate the rank value of the microservice according to Eq. (12)
8: **End for**
9: **End for**
10: Output the rank value for each microservice

Algorithm 1. Smart grid microservices prioritizing.

4.2 Insertion-based core selection

The insertion-based core selection method can insert a microservice into the earliest idle time slot between two microservices already scheduled on the same processing core. The execution time of the being scheduled microservice is less than or equal to the idle time slot, and its earliest execution finish time is less than or equal to the end time of the idle time slot.

The process for inserting microservice v_i^j into an idle time slot on proper processing core is shown in Algorithm 2, where the $EST\left(v_{bottom,r}\right)$ represents the last microservice in the scheduling list of core p_r, the $EFT\left(v_{bottom-1,r}\right)$ represents the second last microservice in the scheduling list of core p_r.

1: **for** each processing core of ECA **do**
2: **while** PQr (the scheduling list of processing core r) $\neq \varnothing$ and $EST\left(v_{bottom,r}\right) > \left(\max\limits_{v_i^k \epsilon pred\left(v_i^j\right)} \left(\text{EFT}\left(v_1^k, p_1\right)\right) + c_{k,j,1,r}\right)$ **do**
3: **if** $w_{j,r}^i \leq EST\left(v_{bottom,r}\right) - EFT\left(v_{bottom-1,r}\right)$ and $EFT\left(v_{bottom-1,r}\right) \geq \max_{v_k^i \epsilon pred\left(v_j^i\right)} \left(\text{EFT}\left(v_k^i, p_1\right) + c_{k,j,1,r}\right)$ **then**
4: $available\left(v_j^i, p_r\right) = EFT\left(v_{bottom-1,r}\right)$
5: **else if** $EFT\left(v_{bottom-1,r}\right) < \max\limits_{v_k^i \epsilon pred\left(v_j^i\right)} \left(\text{EFT}\left(v_k^i, p_1\right) + c_{k,j,1,r}\right)$ and $w_{j,r}^i \leq EST\left(v_{bottom,r}\right) - \max\limits_{v_k^i \epsilon pred\left(v_j^i\right)} \left(\text{EFT}\left(v_k^i, p_1\right) + c_{k,j,1,r}\right)$ **then**
6: $available\left(v_j^i, p_r\right) = \max\limits_{v_k^i \epsilon pred\left(v_j^i\right)} \left(\text{EFT}\left(v_k^i, p_1\right) + c_{k,j,1,r}\right)$
7: remove the last microservice from the PQ_r.
8: **end if**
9: **end while**
10: $EST\left(v_j^i, p_r\right) = \max\left(available\left(v_j^i, p_r\right), \max\limits_{v_k^i \epsilon pred\left(v_j^i\right)} \left(\text{EFT}\left(v_k^i, p_1\right) + c_{k,j,1,r}\right)\right)$
11: $EFT\left(v_1^j, p_r\right) = w_{1,r}^j + EFT\left(v_{1,r}^j\right)$
12: **end for**
13: **if** there are same *EFT* on different processing cores for microservice v_j^i
14: insert it into the core which has the minimal execution time
15: **else**
16: insert it into the core which has the minimal EFT
17: **end if**

Algorithm 2. Insertion policy-based processing core selection algorithm.

4.3 SGS scheduling strategy considering the scheduling unit dividing and emergency mechanisms

There are still two scenarios that we have to consider. One is in the high concurrency scenario, how to balance the processing delay demand between the service, which has an earlier arrival time and low importance, and the service, which has a later arrival time and high importance. The other is how to meet the urgent services' response time demand, which must be completed before the deadline in the emergency state of the smart grid. In response to the above issues, we design the scheduling unit dividing and emergency mechanism, respectively, which have been introduced in the above-proposed microservice scheduling framework for ECA in Section 1, and integrated them into the scheduling strategy. The pseudo-code of the SGSs scheduling algorithm is shown in Algorithm 3. The emergency mechanism program has preset the processing core allocation results of the microservices for the services in an emergency state, and its processing core allocation results can be determined by Algorithm 2.

```
While true do
  if the MQMM detects an urgent service request then
  stop the currently executing microservice, and execute the established scheduling plan for the urgent services.
  continue
  end if
  While SGS request queue Q is not empty
  calculate the length of the SGS request queue |Q|.
  if |Q| ≥ ε then//where ε is the threshold for scheduling unit dividing, and it is equal to the SUL then
  divide the service requests into multiple scheduling units
  sort the service microservices within each scheduling unit in order of rank and move them to the microservice unscheduled list
  else
  sort the service microservices within services request queue in order of rank and move them to the microservice unscheduled list
  end if
  end while
  While unscheduled list is not empty do
  remove the first microservice at the unscheduled list and schedule the microservice to a processing core by the Algorithm 2
  end while
end while
```

Algorithm 3. SGSs scheduling strategy for ECA

5 Simulation

This section reports the numerical simulation result of the proposed strategy. All programs are implemented by Matlab and executed on an HP workstation.

5.1 Simulation background

Some typical SGSs in a smart distribution transformer area (Cen et al., 2022) are chosen to deploy in the ECA with four heterogeneous processing cores. The parameters of the SGSs are shown in Table 1. The service DAG structure information is represented by an adjacency matrix. The execution times on the four processing cores are represented by a vector, where the symbol '∞' indicates the microservice cannot execute in the core. SGS importance is given based on expert experience and divided into five levels from 1 (high) to 5 (low). Because the workload of microservice is difficult obtained accurately, the execution time of microservices on the different processing cores can be obtained by using an application profiler (Sahni et al., 2021) or using the statistics from multiple runs (Bochenina et al., 2016). Due to the low data transfer volume between microservices and the very high transmission rate between processing cores, the communication time is much shorter than the computation time, so communication time is ignored in the simulation. We assume all the kinds of SGSs have the same arrival probability. The number of concurrent service requests is set within the range [10–80], where the proportion of urgent service requests is set to 10%. Considering the service requests arrive at random, we repeat the experiment 100 times for each concurrency situation and take the average value as the simulation result to ensure the effectiveness of the simulation results.

5.2 Performance evaluation

Some benchmark solutions based on the ideas of solutions in existing works are developed to compare the performance.

(1) *Improved Heterogeneous Earliest Finish Time (HEFT)-based strategy(S1)*: The strategy first integrates all the services DAG into a big DAG by adding a virtual common zero entry node and a virtual common zero exit node, and then the integrated DAG uses the HEFT algorithm, which selects the microservice with the highest upward or lowest downward rank and then assign the tasks to the core, which can minimize its earliest finish time. Figure 2 illustrates the integration of multi DAGs.
(2) *Importance-based scheduling strategy (S2)*: The strategy first sorts service requests based on arrival time and then executes each service in sequence based on the HEFT algorithm.
(3) *First Come First Serve-based scheduling strategy (S3)*: this strategy is similar to S2, but it first sorts the service requests based on importance.

For the proposed scheduling strategy, the trigger threshold of the SUDM is set to 20, and the scheduling unit length (SUL) is set to 10. The performance comparison result of these scheduling strategies under different concurrent service requests is shown in Figure 3.

Figure 3A shows the calculation results of M1 under different numbers of concurrent requests n for these scheduling strategies. It can be seen that the proposed strategy obtains better average response time in all situations. As the number of service requests increases, the advantages of the proposed strategy become more

apparent. Compared with S1, S2, and S3, the proposed strategy reduces the average response time by approximately 14.5%, 34.8%, and 25.4% at n = 10, respectively, and the decrease reached 26.5%, 53.6%, and 41.2% at n = 80. For the S2, all microservices of different services are mixed to schedule. Thus, the microservice at the exit node always requires more time to be scheduled. Consequently, it has the worst performance on M1. The services are scheduled one by one in the strategies S1 and S2. In strategy S1, the services are scheduled according to their arrival time. Thus, the early arrival service can be scheduled and finished in time. However, in strategy S3, the services are scheduled according to their importance value. Thus, it will lead to a larger response time for the early arrival service when services arrive later but have higher importance. Especially when the execution time of the services is long, the performance will worsen.

Figure 3B shows the calculation results of M2 under different concurrent service requests for these scheduling strategies. The performance of the proposed strategy is better than other strategies in all concurrent situations. When the number of concurrent requests is low, i.e., n = 10, all the strategies can complete the services before their deadlines. As the concurrent requests increase, the M2 gradually decreases in all the strategies. The S1 has the worst performance on M2 because of its bad performance on average response time, and M2 has already decreased to zero at n = 50. Thus, it is not easy to apply to high-concurrency scenarios in a smart grid.

Figure 3C shows the calculation results of M3 under different concurrent service requests for these scheduling strategies. The proposed scheduling strategy can complete the urgent services before their deadlines in all concurrent situations due to the emergency mechanism, while the other strategies achieved similar performance as M2. Considering the demand for urgent service requests and high concurrency scenarios in the smart grid, emergency mechanisms in the SGS scheduling strategy are necessary.

5.3 Influence analysis of the parameter *SUL*

This section conducts a simulation analysis for the effect of the *SUL* and assesses the effectiveness of SUDM. We set the maximum number of concurrent service requests to 80 and set eight experiment groups at intervals of 10 within [10, 80], where the case of *SUL* = 80 represents the case without considering SUDM. Regardless of the value of *SUL*, urgent services are always completed before the deadline due to the emergency mechanism, so the urgent service were ignored to avoid interference with the simulation results.

Figure 4 shows the performance evaluation results under different *SUL*. It can be seen that the proposed method also improves the performance to some extent. Overall, better M1 and M2 will be achieved as the *SUL* decreases. Compared with the case without considering the SMDA, M1 decreases by approximately 31.4% under the case of *SUL* = 10. The change in M2 is relatively small before the length of the *SUL* is less than 40, and the degree of change in M2 increases sharply and nonlinearly. M2 increases by approximately 76.1% under the case of *SUL* = 10.

Figures 5, 6 show the performance results for services with different importance. It can be seen that the SUDM can effectively balance the performance of services with different levels of importance. Compared with the case without considering the SMDA, the services with low importance can reduce their average response time, as shown in Figure 5, and increase the rate of meeting deadlines, as shown in Figure 6. For example, the services with the lowest importance have a 51.1% reduction in M1 compared to without using SMDA. However, the services with the highest importance have a 17.1% reduction in M1 simultaneously. As for M2, all services achieved varying degrees of improvement compared to not using SMDA. It should be noted that a lower average response time only sometimes means a higher on-time completion rate because some overtime services are completed close to the deadline.

5.4 Performance comparison with the homogeneous multi-cores ECA

This section conducts a simulation analysis for the performance comparison with the homogeneous multi-cores ECA. In the simulation, the homogeneous multi-cores ECA have the same number of cores as the heterogeneous multi-cores ECA, but all the processing cores are set to type 1, whose execution time corresponds to the first element in the vector in the third column of Table 1. The performance comparison result is shown in Figure 7. At the low concurrent requests situation, such as n = 10, the two kinds of ECAs have similar performance in M1 and M2. As the concurrent requests increase, the heterogeneous multi-cores ECA has achieved better performance due to its dedicated core's advantages in differentiated computation ability for different SGSs, and it leads up to 1 and 2 improvements in M1 and M2, respectively, at n = 80. In addition, the performance result of M3 shows that both can complete the urgent services before the deadline under all the concurrent situations due to the emergency mechanism.

6 Conclusion

In this article, we have proposed an SGS scheduling strategy for ECA in smart grids. A microservice scheduling framework was presented to meet the demand for microservice-based SGS processing in the smart grid. Considering the SGS scheduling attributes of deadline and importance, a microservice prioritizing method was designed, and then the insert-based policy was utilized to schedule the microservice to the cores for efficient utilization of ECA's computing resources. Two novel mechanisms, SUDM and EPM, were presented to deal with urgent services under abnormal smart grid conditions and balance the performance of SGS with different importance, respectively. Extensive simulation experiments have demonstrated that the proposed strategy can effectively solve the SGS scheduling problem for the ECA.

Compared with other benchmark solutions, the proposed strategy can effectively reduce the average response time of services, improve the on-time completion rate, and guarantee the completion of urgent services before the deadline.

The work of this article aims to fill the related research gap in smart grids and promote the development of the ECA in the smart grid. In future work, we will further study the SGS offloading strategy to meet the quality of service demand for resource-constrained ECA by offloading microservices to the cloud center or other ECAs in smart grids.

Data availability statement

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

Author contributions

KH: Conceptualization, Data curation, Formal Analysis, Methodology, Validation, Writing–original draft, Writing–review and editing. JQ: Data curation, Formal Analysis, Investigation, Methodology, Validation, Writing–original draft, Writing–review and editing. ZC: Funding acquisition, Methodology, Project administration, Supervision, Writing–review and editing. XL: Conceptualization, Funding acquisition, Software, Supervision, Writing–review and editing. YL: Formal Analysis, Software, Validation, Writing–review and editing. JZ: Formal Analysis, Software, Validation, Writing–review and editing.

Funding

The author(s) declare financial support was received for the research, authorship, and/or publication of this article. This work is supported by the Key-Area Research and Development Program of Guangdong Province under grant No. 2019B111109002.

Conflict of interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's note

Supplementary material

The Supplementary Material for this article can be found online at: https://www.frontiersin.org/articles/10.3389/fenrg.2024.1358310/full#supplementary-material

References

Arabnejad, H., and Barbosa, J. G. (2014). List scheduling algorithm for heterogeneous systems by an optimistic cost table. *IEEE Trans. Parallel Distrib. Syst.* 25 (3), 682–694. doi:10.1109/TPDS.2013.57

Bachoumis, A., Andriopoulos, N., Plakas, K., Magklaras, A., Alefragis, P., Goulas, G., et al. (2022). Cloud-edge interoperability for demand response-enabled fast frequency response service provision. *IEEE Trans. Cloud Comput.* 10 (1), 123–133. doi:10.1109/TCC.2021.3117717

Bochenina, K., Butakov, N., and Boukhanovsky, A. (2016). Static scheduling of multiple workflows with soft deadlines in non-dedicated heterogeneous environments. *Future Gener. comput. Syst.* 55, 51–61. doi:10.1016/j.future.2015.08.009

Cen, B., Hu, C., Cai, Z., Wu, Z., Zhang, Y., Liu, J., et al. (2022). A configuration method of computing resources for microservice-based edge computing apparatus in smart distribution transformer area. *Int. J. Electr. Power Energy Syst.* 138, 107935. doi:10.1016/j.ijepes.2021.107935

Chamola, V., Sancheti, A., Chakravarty, S., Kumar, N., and Guizani, M. (2020). An IoT and edge computing based framework for charge scheduling and EV selection in V2G systems. *IEEE Trans. Veh. Technol.* 69 (10), 10569–10580. doi:10.1109/TVT.2020.3013198

Chukwu, U. C., and Mahajan, S. M. (2014). Real-time management of power systems with V2G facility for smart-grid applications. *IEEE Trans. Sustain. Energy* 5 (2), 558–566. doi:10.1109/TSTE.2013.2273314

Gao, Y., and Feng, K. (2022). "A deep reinforcement learning-based approach to the scheduling of multiple workflows on non-dedicated edge servers," in *Parallel distrib. Comput. Appl. Technol.* Editor H. Shen Cham, Switzerland: Springer International Publishing, 261–272. doi:10.1007/978-3-030-96772-7_24

Gao, Y., Zhang, S., and Zhou, J. (2019). A hybrid algorithm for multi-objective scientific workflow scheduling in IaaS cloud. *IEEE Access* 7, 125783–125795. doi:10.1109/ACCESS.2019.2939294

Jia, Q., Chen, S., Yan, Z., and Li, Y. (2022). Optimal incentive strategy in cloud-edge integrated demand response framework for residential air conditioning loads. *IEEE Trans. Cloud Comput.* 10 (1), 31–42. doi:10.1109/TCC.2021.3118597

Jiang, Y., Hu, S., Niu, Z., and Wu, L. (2020). Software architecture analysis of intelligent distribution and transformation terminal based on container technology. *J. Phys. Conf. Ser.* 1646 (1), 012085. doi:10.1088/1742-6596/1646/1/012085

Kaur, A., Singh, P., Singh Batth, R., and Peng Lim, C. (2022). Deep-Q learning-based heterogeneous earliest finish time scheduling algorithm for scientific workflows in cloud. *Softw. Pract. Exp.* 52 (3), 689–709. doi:10.1002/spe.2802

Kelefouras, V., and Djemame, K. (2022). Workflow simulation and multi-threading aware task scheduling for heterogeneous computing. *J. Parallel Distrib. Comput.* 168, 17–32. doi:10.1016/j.jpdc.2022.05.011

Lan, D., Taherkordi, A., Eliassen, F., Liu, L., Delbruel, S., Dustdar, S., et al. (2022). Task partitioning and orchestration on heterogeneous edge platforms: the case of vision applications. *IEEE Internet Things J.* 9 (10), 7418–7432. doi:10.1109/JIOT.2022.3153970

Li, B., (2018). Application prospect of edge computing in power demand response business. *Power Syst. Technol.* 42 (1), 79–87. doi:10.13335/j.1000-3673.pst.2017.1548

Li, J., Gu, C., Xiang, Y., and Li, F. (2022). Edge-cloud computing systems for smart grid: state-of-the-art, architecture, and applications. *J. Mod. Power Syst. Clean. Energy* 10 (4), 805–817. doi:10.35833/MPCE.2021.000161

Li, X., Chen, T., Cheng, Q., and Ma, J. (2021). Smart applications in edge computing: overview on authentication and data security. *IEEE Internet Things J.* 8 (6), 4063–4080. doi:10.1109/JIOT.2020.3019297

Lyu, Z., Wei, H., Bai, X., and Lian, C. (2020). Microservice-based architecture for an energy management system. *IEEE Syst. J.* 14 (4), 5061–5072. doi:10.1109/JSYST.2020.2981095

Mondal, A., Misra, S., and Chakraborty, A. (2022). Dynamic price-enabled strategic energy management scheme in cloud-enabled smart grid. *IEEE Trans. Cloud Comput.* 10 (1), 111–122. doi:10.1109/TCC.2021.3118637

Peng, C., and Niu, Y. (2023). Optimal serving strategy for vehicle-to-grid business: service agreement, energy reserve estimation, and profit maximization. *Front. Energy Res.* 11, 1199442. doi:10.3389/fenrg.2023.1199442

Rehman, A., Hussain, S. S., ur Rehman, Z., Zia, S., and Shamshirband, S. (2019). Multi-objective approach of energy efficient workflow scheduling in cloud environments. *Concurr. Comput. Pract. Exp.* 31 (8), e4949. doi:10.1002/cpe.4949

Rodriguez, M. A., and Buyya, R. (2014). Deadline based resource provisioningand scheduling algorithm for scientific workflows on clouds. *IEEE Trans. Cloud Comput.* 2 (2), 222–235. doi:10.1109/TCC.2014.2314655

Roy, S. K., Devaraj, R., and Sarkar, A. (2023). SAFLA: scheduling multiple real-time periodic task graphs on heterogeneous systems. *IEEE Trans. Comput.* 72 (4), 1067–1080. doi:10.1109/TC.2022.3191970

Sahni, Y., Cao, J., Yang, L., and Ji, Y. (2021). Multihop offloading of multiple DAG tasks in collaborative edge computing. *IEEE Internet Things J.* 8 (6), 4893–4905. doi:10.1109/JIOT.2020.3030926

Topcuoglu, H., Hariri, S., and Wu, M.-Y.(2002). Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Trans. Parallel Distrib. Syst.* 13 (3), 260–274. doi:10.1109/71.993206

Wojtowicz, R., Kowalik, R., and Rasolomampionona, D. D. (2018). Next generation of power system protection automation—virtualization of protection systems. *IEEE Trans. Power Deliv.* 33 (4), 2002–2010. doi:10.1109/TPWRD.2017.2786339

Wu, J., Qiu, R., Wang, M., Han, R., Huang, W., and Guo, Z. (2022). Control strategy of distributed energy micro-grid involving distribution system resilience. *Front. Energy Res.* 10, 841269. doi:10.3389/fenrg.2022.841269

Xiao, H., He, H., Zhang, L., and Liu, T. (2023). Adaptive grid-synchronization based grid-forming control for voltage source converters. *IEEE Trans. Power Syst.*, 1–4. 2023, doi:10.1109/TPWRS.2023.3338967

Yin, X., Zhu, Y., and Hu, J. (2022). A subgrid-oriented privacy-preserving microservice framework based on deep neural network for false data injection attack detection in smart grids. *IEEE Trans. Ind. Inf.* 18 (3), 1957–1967. doi:10.1109/TII.2021.3102332

Zhang, J., (2019). Conception and application of smart terminal for distribution internet of things. *High. Volt. Eng.* 45 (6), 1729–1736. doi:10.13336/j.1003-6520.hve.20190604007

Zhou, J., Cen, B., Cai, Z., Chen, Y., Sun, Y., Xue, H., et al. (2021). Workload modeling for microservice-based edge computing in power internet of things. *IEEE Access* 9, 76205–76212. doi:10.1109/ACCESS.2021.3081705