Future Generation Computer Systems 159 (2024) 567–579

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Replica fault-tolerant scheduling with time guarantee under energy constraint in fog computing

Ruihua Liu [a,1], Wufei Wu [b,*,1], Xiaochuan Guo [b,1], Gang Zeng [c], Keqin Li [d]

[a] *College of Mathematics and Computer Science, Nanchang University, Nanchang, Jiangxi, China*
[b] *College of Information Engineering, Nanchang University, Nanchang, Jiangxi, China*
[c] *College of Engineering, Nagoya University, Nagoya 464-8603, Japan*
[d] *Department of Computer Science, State University of New York, New Paltz, NY 12561, USA*

A R T I C L E I N F O

Keywords:
Energy constraint
Fog computing
Reliability
Replica
Scheduling length
Task scheduling

A B S T R A C T

Fog computing offers stronger local computing power and reduces data transmission load, making it an ideal solution to meet the energy-saving and efficient requirements of intelligent connected vehicle applications. As intelligent networked vehicles and vehicle–road collaboration technologies advance rapidly, optimizing scheduling under fog computing architecture has become a prominent research area. However, existing studies primarily concentrate on parallel task scheduling with low energy consumption or high real-time performance, failing to address the requirement for high reliability in intelligent networked vehicle scenarios. To achieve time and reliability optimization for vehicular applications in fog computing architecture, this paper proposes a fog computing task scheduling algorithm and explores its extension using replication techniques. Subsequently, the algorithm underwent evaluation utilizing randomly generated directed acyclic graph models as well as real-life automotive application instances. The experimental findings indicate that in comparison to existing methods, the proposed algorithm exhibits a notable improvement in reliability while ensuring time optimization, thereby demonstrating a distinct level of advancement and practicality.

1. Introduction

1.1. Background

Intelligent Connected Vehicles (ICV) refer to the integration of vehicle networking and intelligent technologies, aiming to revolutionize the transportation industry. Equipped with advanced onboard sensors, controllers, actuators, and communication technologies, ICVs enable intelligent information exchange and sharing between vehicles, people, roads, and back-end systems. The primary focus of ICVs is to address critical challenges related to safety, energy efficiency, environmental protection, and autonomous perception capabilities [1]. To achieve these goals, it is crucial to improve the safety of vehicles and optimize the real-time performance and reliability of the networked vehicle system through effective algorithms for task sequencing and execution location determination [2,3].

Against the backdrop of rapid development in information and communication technology, the demand for high processing resources, low latency, and high reliability in ICV applications continues to increase [4]. These applications typically require significant computational power and consume a large amount of energy. However, intelligent connected vehicle terminals often have limited computing capabilities and battery capacity, making it difficult to run complex applications on them [5,6]. In recent years, fog computing architecture has emerged as an effective solution to provide high-performance computing capabilities, especially in applications such as intelligent networked vehicles, where research on task scheduling algorithms in fog computing environments has attracted widespread attention [7]. Fog computing extends the concept of cloud computing by decentralizing data management, processing, and applications to network edge devices (such as sensors and edge devices) rather than relying solely on cloud data centers [8]. The core idea is to introduce an intermediate layer called the fog layer, situated between the end devices and the cloud data center. The fog layer processes and stores data locally, thereby alleviating the burden on cloud computing and storage.

Fig. 1 illustrates the conceptual distance between cloud and fog, as well as their relationship with the user side. In fog computing,

* Corresponding author.
E-mail addresses: liuruihua@email.ncu.edu.cn (R. Liu), wuwufei@ncu.edu.cn (W. Wu), 416100220228@email.ncu.edu.cn (X. Guo), zeng.gang@e.mbox.nagoya-u.ac.jp (G. Zeng), lik@newpaltz.edu (K. Li).
[1] Ruihua Liu, Wufei Wu and Xiaochuan Guo are co-first authors. Our code is available at https://github.com/silent309/FogComputing2024.git.

https://doi.org/10.1016/j.future.2024.05.014
Received 29 December 2023; Received in revised form 30 March 2024; Accepted 14 May 2024
Available online 17 May 2024

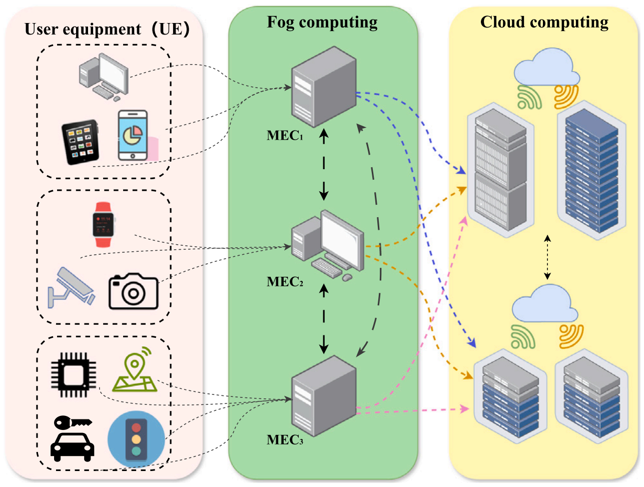

Fig. 1. A simple conceptual diagram of cloud–fog-terminal relationship.

offloading compute-intensive tasks from user equipment (UE) to mobile edge cloud (MEC) servers can effectively enhance the computing capabilities of clients, extend their battery life, and improve energy efficiency, enabling user terminals to save energy and accomplish more tasks in the same amount of time. Additionally, due to its proximity to the user side, the fog layer provides significant advantages in reducing transmission costs and latency [9].

1.2. Motivation

In general, the mobile application running on UE can be represented as a task list with different priority constraints, which can be quite complex [10–12]. There are several commonly used models for representing parallel applications with prioritized tasks, such as directed acyclic graphs (DAGs), hybrid DAGs (HDAGs), and task interaction graphs (TiGs) [13–15]. Furthermore, these tasks often have different execution computational tasks and communication data transmission volumes. Unlike common distributed system task scheduling problems, the fog server in fog computing does not need UE to bear energy loss when executing tasks, and more factors need to be comprehensively considered, which further aggravates the complexity of the problem [16–18].

Additionally, existing research on fog computing task scheduling optimization mainly focuses on reducing the overall scheduling length or improving energy usage efficiency, but insufficient attention has been given to enhancing application reliability [19]. Meanwhile, it should be noted that the maturity level of task scheduling strategies in fog computing is not as advanced as that in traditional heterogeneous distributed systems, leaving significant room for further development. Considering the importance of reliability metrics in the intelligent networked vehicles, there is an urgent need for supplementary contributions in this field.

Therefore, the motivation of this paper is to find a better optimization algorithm to improve the time and reliability of task offloading strategies for applications in fog computing environments.

1.3. Our contributions

The main contributions of this paper are summarized as follows.

- We attempt to add the transmission reliability calculation method based on block fading into the reliability calculation model of fog computing, replace the linear reliability [20], achieve a more accurate quantization of reliability index in fog computing environment.
- We propose a time-reliability dual-objective optimization scheduling algorithm, referred to as Energy-Constrained Level-by-Level Time-Reliability Optimization Scheduling (ECLLTROS), which taking into account the communication or computation consumption generated during replica execution in fog computing, calculates the number of replicas for each task, filters the execution environment, and ensures the reliability and response-time of the application.

The rest of this paper is organized as follow. Section 2 reviews related research. Section 3 introduces related models of the system. Section 4 introduces the problems to be solved. Section 5 proposes the ECLLTROS algorithm. Section 6 assesses the algorithms and Section 7 concludes this study.

2. Related work

There have been many related studies on achieving dual-objective optimization of time and reliability under energy constraints in a fog computing environment, which can be mainly divided into the following two aspects: (1) achievements in reliability optimization, (2) time optimization achievements. Next, we will introduce them separately.

(1) Achievements in optimizing reliability. Reliability refers to the probability that an application continuously provides the correct service, which is an important metric for assessing whether an application can stably and reliably deliver services. This has made the issue of maximizing reliability under energy constraints a research hots pot in recent years [21,22]. Over the past 50 years, significant advancements have been made in developing various methods to enhance reliability. These methods can be broadly categorized into four key areas [23,24]:

1. Fault prevention: These measures aim to prevent faults from occurring or being introduced in the first place.
2. Fault tolerance: This focuses on mitigating the impact of business failures when they do occur, ensuring continuity and resilience.
3. Troubleshooting: Strategies in this area aim to minimize the number and severity of failures, enabling prompt identification and resolution of issues.
4. Failure prediction: This involves estimating the current number of failures, forecasting their future occurrence, and assessing the potential consequences they may have.

In the research field of task scheduling, fault-tolerant methods are often used to improve the reliability. Liu et al. proposed a common reliability model, and achieved cost reduction by efficiently allocating tasks among heterogeneous embedded systems while meeting end-to-end distributed function response time targets [25]. Zhao et al. designed a dynamic replica execution algorithm which named MaxRe to maximize the reliability of heterogeneous distributed systems, but did not consider the energy used for replica offloading execution [26]. Xie et al. assumed that for the current task, the maximum reliability value would be assigned to the processor for each subsequent unassigned task to minimize the cost of resource consumption. This method was called MRCRG method [27], but the algorithm was not mature enough in the stage of reliability index division. On this basis, Yuan et al. invented the RGAGM algorithm to pre-allocate reliability values to unassigned tasks according to the geometric mean, thereby reducing resource consumption costs and effectively ensuring reliability targets [28].

On the other hand, the addition of replica can also effectively improve reliability. Xie et al. focused on the work related to quantitative fault-tolerant scheduling. In order to meet the reliability requirements of workflow on heterogeneous infrastructure, the execution cost was reduced by minimizing the number of replicas [29]. Wang et al. showed an improved algorithm to search for all optimal reliability communication paths for the current task [30]. This approach focuses on cloud or heterogeneous distributed system clusters and has not yet been applied to vehicular applications in fog computing architecture.

(2) Achievements in optimizing time. Clouds and fog differ in terms of task offloading possibilities and diversity. To optimize virtualized resources and deliver corresponding delay-sensitive services effectively, Minh et al. discovered a new context-aware service decentralization mechanism for experimental evaluation and comparative measurement of real intelligent transportation system services in fog computing [1]. Khan et al. utilized fog computing to address the task offloading issue in the 5G IoHT cloud computing environment and presented the MSCA algorithm [31]. However, when the fog is unable to independently perform complete task computations, some tasks must be offloaded from the fog to the cloud. Determining the optimal timing for transferring the task from the fog to the cloud represents a significant decision point. The MTFCT algorithm, proposed by Jindal et al. theoretically tackles this challenge, although it has yet to be validated through relevant experiments [8].

Li studied the problem of computational offloading under the background of traditional task scheduling from the perspectives of optimal computational offloading under energy constraints and optimal computational offloading under time constraints [19]. At the same time, Li applied this idea to fog computing, adopted the idea of step-by-step scheduling and greedy algorithm to eliminate priority constraints and maximize the utilization rate of fog server, while reducing the total scheduling length as much as possible [32]. Meanwhile, Li applied this idea to fog computing. The proposed ECLL algorithm adopts the concepts of step-by-step scheduling and greedy algorithms, simplifying the problem by eliminating priority constraints among tasks. This approach maximizes the utilization of fog servers while minimizing the total scheduling length as much as possible [32]. Alzailaa et al. proposed a task classification and scheduling scheme for unconstrained tasks in fog-cloud networking environments, which takes into account task characteristics, user profiles, environmental exposure, and network topology, to improve the overall latency of high-priority critical tasks [33]. Tsega et al. developed a deadline-aware data offloading model and algorithm using the design science approach, which has been implemented on fog nodes. The algorithm considers various factors such as task deadlines, computational capabilities of fog nodes, task characteristics, and transmission latency during the scheduling process, which could ensure efficient management of resources, optimizing task completion times while meeting deadlines [34].

In this study, our goal is to propose a scheduling algorithm in fog computing to achieve reliability objective optimization for energy-constrained applications, building upon existing research on time optimization.

3. Establishment of model

In this section, we introduce four models that are directly related to task optimization scheduling. The difference from previous research models is that we have incorporated a block fading-based transmission reliability calculation method into the reliability model, making the reliability calculation more comprehensive. We have compiled the symbols and their definitions as they appear in this article in Table 1.

3.1. The application model

Consider a user equipment UE has a mobile application $G = (L, \prec)$, where $L = (t_1, t_2, \cdots, t_m)$ represents the list of tasks to be performed. Each task t_i is denoted as $t_i = (r_i, d_i)$, where r_i represents the computational requirement of t_i (the computational workload, measured in billions of processor cycles or billions of instructions (BI)), d_i represents the communication requirement of t_i (the amount of data exchanged between the UE and MEC, measured in millions of bits (MB)). $\prec$ represents the existence of priority constraints within the task list, indicating the execution order that tasks must adhere to. Applications with priority constraints can be described using directed acyclic graphs (DAGs).

Table 1
Notations and definitions.

Notations	Definitions
G	A set of tasks with priority constraints
N	A set of heterogeneous processors
E	The total energy constraint value of G
S	The optimal scheduling strategy of G under energy constraint E
RE	The task offloading execution reliability of G
T	The task offloading execution time of G
r_i	The execution requirement of task t_i
d_i	The communication requirement of task t_i
$s_{i,j}$	The execution speed of task t_i on MEC$_j$
$c_{i,j}$	The communication speed of task t_i from UE to MEC$_j$
$ct_{i,j}$	The communication time of task t_i from UE to MEC$_j$
$et_{i,j}$	The execution time of task t_i on MEC$_j$
$T_{i,j}$	The offloading execution time of task t_i on MEC$_j$
TP_j	The cumulative task offloading execution time on MEC$_j$
$P_{i,j}$	The transmission power of task t_i from UE to MEC$_j$
w_j	The bandwidth of the channel between UE and MEC$_j$
β_j	A constant that accounts for background noise, adjacent channel interference and channel gain between UE and MEC$_j$
$\left\|h_{i,j}\right\|^2$	The power attenuation coefficient of task t_i from UE to MEC$_j$
$\lambda_{i,j}$	The constant failure rate per time unit of the processor MEC$_j$
$RE_{i,j}^{comm}$	The communication reliability of task t_i on MEC$_j$
$RE_{i,j}^{comp}$	The execution reliability of task t_i on MEC$_j$
SNR$_{ar}$	The actual received signal noise ratio
θ_j	The threshold signal noise ratio of MEC$_j$
$P_{i,j}^{out}$	The communication interruption rate of task t_i from UE to MEC$_j$
$proc(t_i)$	The subscript of MEC which actually allocated for task t_i

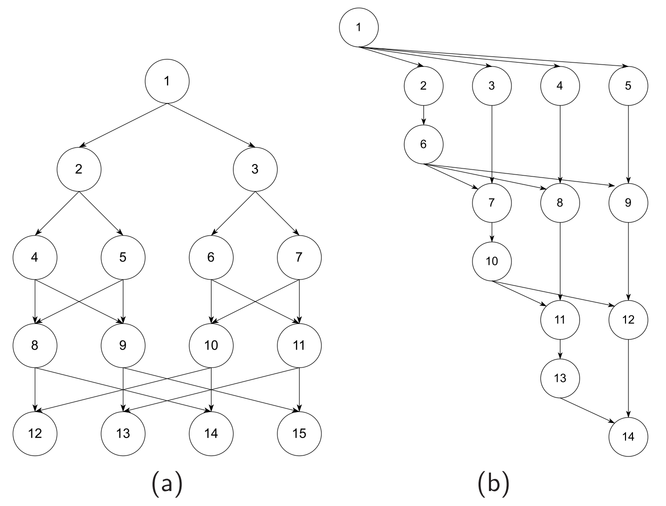

Fig. 2. (a) a Fast Fourier Transform DAG model with 5 layers architecture. (b) a Gaussian Elimination DAG model with 8 layers architecture.

Fig. 2 shows two DAGs with classical structures, based on fast Fourier transform (FFT) and Gaussian elimination (GE) respectively [27,28]. In the DAG, nodes represent the m tasks in the task list L, and arcs between nodes are defined such that there exists an arc from t_1 to t_2 only if $t_1 \prec t_2$, meaning that if $t_1 \prec t_2$, then t_1 is a predecessor of t_2 and t_2 can only start its execution after t_1 has finished.

3.2. The computation and communication models

Assume there is a fog server set N consisting of n heterogeneous server MECs, that is, $N = (\text{MEC}_1, \text{MEC}_2, \cdots, \text{MEC}_n)$. Each MEC$_j$ has its own processor execution speed s_j in GHz or billions of instructions per second $(1 \le j \le n)$, s_j is determined by the power available to the server itself, regardless of UE.

In terms of task offloading, the offloading execution time of task t_i includes the execution time on the server and the transmission time from the terminal to the fog server. UE can choose to execute each task locally or send it to an idle MEC in the fog. If it is decided to directly execute t_i on the UE, the computation speed $s_{0,i}$ of task t_i is determined by the UE. At the same time, local execution does not incur communication time, so the offloading execution time of task t_i on the UE is equal to its execution time (in seconds), which can be calculated as follows (where $1 \le i \le m$):

$$T_{i,0} = et_{i,j} = r_i/s_{i,0}. \tag{1}$$

However, if the decision is made to offload task t_i to an MEC for fog computing, the task's execution time on MEC$_j$ can be computed as r_i/s_j. In addition, the transmission time between the UE and MEC$_j$ for task t_i can be derived from $d_i/c_{i,j}$, where $c_{i,j}$ is the communication speed (data transfer rate in megabits per second) of task t_i between the UE and MEC$_j$. Therefore, the offloading execution time of task t_i on MEC$_j$ can be calculated as follows (where $1 \le i \le m, 1 \le j \le n$):

$$T_{i,j} = ct_{i,j} + et_{i,j} = r_i/s_{i,j} + d_i/c_{i,j}. \tag{2}$$

3.3. The power consumption model

In terms of energy consumption, the power consumption generated by UE during task calculations can be categorized into two parts [35]: dynamic power consumption and static power consumption. The static power consumption is mainly incurred by the current leakage, whereas the dynamic power consumption is mainly caused by the charging and discharging of capacitors, both of which are measured in watts. The dynamic power consumption, denoted as P_d, is typically expressed as

$$P_d = \xi {s_0}^{\alpha}, \tag{3}$$

where ξ and α are constants determined by the specific technique used [36]. The static component P_s is usually considered constant. Therefore, the total power consumption, denoted as P, is given by the sum of the dynamic and static components,

$$P = P_d + P_s = \xi {s_0}^{\alpha} + P_s. \tag{4}$$

For all cases where $1 \le i \le m$, when task t_i is not offloaded and is directly executed on the UE with the computation speed of $s_{i,0}$, its power consumption is $P_0 = \xi {s_{i,0}}^{\alpha} + P_s$. The energy consumption of task t_i on the UE, measured in joules, can be calculated as follows:

$$E^{comp}_{i,0} = P(r_i/s_{0,i}) = ((\xi {s_{i,0}}^{\alpha} + Ps)/s_{i,0})r_i. \tag{5}$$

Task offloading operations in the fog environment have more constraints to consider. This is because the calculation of energy consumption in different servers varies depending on the environment. If a task is executed on the terminal, only its computational energy consumption is considered (no task offloading occurs). However, if a task is offloaded and executed on an MEC, only its transmission energy consumption is considered (the energy required for the fog server to execute the task is provided by itself rather than the UE [19,32]).

Let $P_{i,j}$ be the transmission power (in watts) of the transmission task t_i from UE to MEC$_j$ ($1 \le j \le n$). The wireless communication environment in which tasks are transmitted between UE and MEC is assumed to be a block fading environment, that is, the transmission action will be affected by channel fading, and the channel fading coefficient will remain unchanged during the transmission of a single task.

So the communication speed (i.e. data transfer speed, measured in millions of bits per second) from UE to MEC$_j$ is [19,32]:

$$c_{i,j} = w_j \log_2(1 + P_{t,j}\beta_j \left|h_{i,j}\right|^2), \tag{6}$$

where w_j represents the channel band width, β_j is a constant that accounts for background noise, adjacent channel interference, and channel gain between UE and MEC$_j$. $h_{i,j}$ is a random coefficient used to characterize Rayleigh fading (wireless channel fading), which takes into account factors such as multipath effect and shadow effect. Its probability density follows the Rayleigh distribution, and the power attenuation coefficient $\left|h_{i,j}\right|^2$ follows the exponential distribution with an average value of σ_j^2, namely:

$$f_h(x) = \frac{1}{\sigma_j^2}\exp(-x/\sigma_j^2). \tag{7}$$

Thus, the energy consumption for communication of task t_i from UE to MEC$_j$ is:

$$E^{comm}_{i,j} = P_{i,j}(d_i/c_{i,j}) = (2^{c_{i,j}/\omega_j} - 1)d_i/(\beta_j c_{i,j}\left|h_{i,j}\right|^2). \tag{8}$$

3.4. The reliability model

According to the ISO26262 standard, random hardware failures (i.e., instantaneous failures) are expected to follow a specific probability distribution. During the calculation process, the occurrence of transient faults in tasks based on a DAG function is assumed to follow the Poisson distribution [27,28].

Set λ_j to represent the constant failure rate per time unit of the processor, so the reliability $RE_{i,j}$ of task t_i executed on processor during its execution time can be expressed as :

$$RE^{comp}_{i,j} = e^{-\lambda_j et_{i,j}}. \tag{9}$$

In the aspect of task transmission reliability, communication interruption rate is often used to measure the communication reliability of the system in the relevant research of mobile cloud computing [37,38]. The interrupt rate is an important performance index in wireless communication. According to Shannon channel coding theory, when the signal transmission rate is lower than the channel capacity, a coding technology must be utilized to ensure accurate signal reception. Conversely, when the signal transmission rate exceeds the channel capacity, the signal cannot be correctly received, leading to a communication interruption. Therefore, given the target transmission rate TR and the channel capacity C, the communication interruption rate can be defined as:

$$P_{out} = \Pr(C < TR). \tag{10}$$

According to Shannon capacity formula, channel capacity and instantaneous signal noise ratio (SNR) show a monotonically increasing relationship, assuming that the actual received SNR and threshold SNR of the communication receiving end are SNR_{ar} and SNR_0 respectively, the communication interruption rate can also be expressed as:

$$P_{out} = \Pr(SNR_{ar} < SNR_0). \tag{11}$$

When task t_i is unloaded to MEC$_j$ for execution, according to Eq. (6), the actual received SNR of MEC$_j$ is as follows:

$$SNR_{ar} = P_{t,j}\beta_j\left|h_{i,j}\right|^2. \tag{12}$$

Assuming that the threshold SNR of MEC$_j$ is θ_j, then according to Eqs. (7) and (11), the communication interruption rate at this time can be expressed as:

$$\begin{aligned} P^{out}_{i,j} &= \Pr(SNR_{ar} < \theta_j) \\ &= \Pr(\left|h_{i,j}\right|^2 < \theta_j/P_{t,j}\beta_j) \\ &= \int_0^{\frac{\theta_j}{P_{t,j}\beta_j}} \frac{1}{\sigma_j^2}\exp(-x/\sigma_j^2)dx \\ &= 1 - \exp(-\theta_j/P_{t,j}\beta_j\sigma_j^2). \end{aligned} \tag{13}$$

Based on this, the communication reliability when task t_i is offloaded to MEC$_j$ can be obtained as:

$$RE^{comm}_{i,j} = 1 - P^{out}_{i,j} = \exp(-\theta_j/P_{t,j}\beta_j\sigma_j^2), \tag{14}$$

where $1 \le k \le m$. Obviously, when task t_i is executed on UE, $RE^{comm}_{i,j} = 1$. To sum up, we can get the total reliability when task t_i is assigned to MEC_j:

$$RE_{i,j} = RE^{comp}_{i,j} \cdot RE^{comm}_{i,j}. \tag{15}$$

The reliability value of the application G is determined by taking the product of the reliability values of all tasks in the list L, namely:

$$RE(G) = \prod_{t_i \in L} (RE(t_i, \text{MEC}_{proc(t_i)})), \tag{16}$$

where $\text{MEC}_{proc(t_i)}$ represents the MEC actually allocated for task t_i.

4. Problem definition

In this section, we provide a formal definition of the optimization problem addressed in this study.

The DAG-based application $G = (L, \prec)$ has a task list $L = (t_1, t_2, \dots, t_m)$, which is generated on terminal UE $= (s_0, \xi, P_s)$, and $t_i = (r_i, d_i)$ $(1 \le i \le m)$. Suppose there are n edge servers MEC: MEC_1, MEC_2, $\cdots$, MEC_n in fog computing environment, $\text{MEC}_j = (s_j,\ c_j,\ W_j)$ for all $j \in [1, n]$. The task scheduling strategy S for the mobile application G aims to determine the execution starting time and location (either on the UE or MEC) for each task t_i in the task list L $(1 \le i \le m)$.

R_j is defined as the total execution requirement of all tasks t_i transmitted to MEC_j, and D_j is the total data of tasks transmitted to MEC_j, L_j is defined as the set of tasks to be executed on MEC_j $(0 \le j \le n)$ (for the convenience of calculation and statistics, UE is defined as MEC_0). In terms of time, the execution time of L_0 is $T_0 = R_0/s_0$ (there is no transmission time for local execution), the execution time of L_j is $T_j = R_j/s_j + D_j/r_j$, and the total time of L is:

$$T = \frac{R_0}{s_0} + \sum_{j=1}^{n} \left(\frac{R_j}{s_j} + \frac{D_j}{c_j} \right). \tag{17}$$

In relation of energy consumption, it should be noted that local task execution does not incur transmission energy consumption. If the task is offloaded to the fog-end server, its computing energy consumption is not taken into account (as the server bears it), and can be obtained from Eqs. (5) and (8). The total energy consumption of the application amounts to:

$$E(G) = \left(\frac{\xi {s_0}^{\alpha} + P_s}{s_0} \right) R_0 + \sum_{j=1}^{n} \left(\frac{2^{c_j/w_j} - 1}{\beta_j c_j \left| h_{i,j} \right|^2} \right) D_j. \tag{18}$$

In regards to task reliability, the total reliability of application G is determined by the product of the individual reliabilities of all MECs, as calculated using Eq. (9), (14), and (15). The calculation entails the multiplication of the execution reliability with the communication reliability for each MEC, as demonstrated below:

$$RE(G) = \exp\left(\sum_{j=0}^{n} -\lambda(R_j/s_j) - \frac{\theta_j}{P_{t,j} \beta_j \sigma_j^2} \right). \tag{19}$$

Now we can define dual objective optimization problem to optimize the task offloading scheduling scheme in fog computing, taking into account execution time, reliability, and energy constraint.

Problem(Time-Reliability dual objective optimization scheduling problem under energy constraint in fog computing): Given a DAG-based task list denoted as $L = (t_1,\ t_2,\ \cdots,\ t_m)$, where each task $t_i = (r_i,\ d_i)$ $(1 \le i \le m)$, and a UE $= (\xi,\ \alpha,\ P_s)$, as well as a set of n MEC: MEC_1, MEC_2, $\cdots$, MEC_n, where each $\text{MEC}_j = (s_j,\ c_j,\ w_j)$ $(1 \le j \le n)$, and energy constraints are represented by $\widetilde{E}$. The objective is to find a task scheduling strategy, denoted as $S = (L_0,\ L_1,\ \cdots,\ L_n)$, and determine the computation speed s_0 and communication speed c_j $(1 \le j \le n)$, in such a way that minimize time T and maximize reliability RE while ensuring that the total energy consumption E does not exceed $\widetilde{E}$.

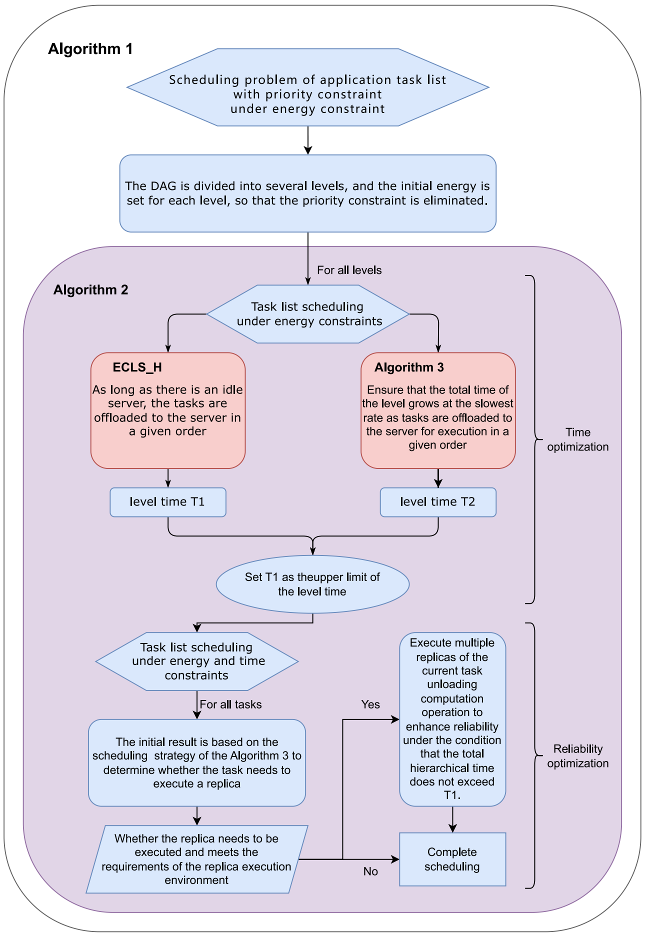
Algorithm 1
Scheduling problem of application task list with priority constraint under energy constraint
The DAG is divided into several levels, and the initial energy is set for each level, so that the priority constraint is eliminated.
For all levels
Algorithm 2
Task list scheduling under energy constraints
ECLS_H
As long as there is an idle server, the tasks are offloaded to the server in a given order
Algorithm 3
Ensure that the total time of the level grows at the slowest rate as tasks are offloaded to the server for execution in a given order
level time T1
level time T2
Set T1 as theupper limit of the level time
Time optimization
Task list scheduling under energy and time constraints
For all tasks
The initial result is based on the scheduling strategy of the Algorithm 3 to determine whether the task needs to execute a replica
Whether the replica needs to be executed and meets the requirements of the replica execution environment
Yes
No
Execute multiple replicas of the current task unloading computation operation to enhance reliability under the condition that the total hierarchical time does not exceed T1.
Complete scheduling
Reliability optimization

Fig. 3. The flow chart of the proposed three-algorithm solution.

5. Proposed algorithms

An overview of the solution is shown in Fig. 3 to illustrate the logic and invocation.

In our research, we mainly consider the scheduling optimization perspective of single-user and multi-server, so the proposed algorithms run on UE. For the problem to be solved, we will follow the following three steps to simplify and deal with it:

1. Level-by-level scheduling. We attempt to divide the tasks of the same execution priority in the task list into the same class, that is, the same layer of task list in the DAG, which can eliminate the priority constraint in the application. So that the task list optimization scheduling problem with priority constraints under energy constraints is simplified to the task list optimization scheduling problem without priority constraints under energy constraints, and an initial energy value is set for each level, and the remaining energy is distributed many times in the form of slices.
2. Time optimization. After the simplification of the problem in the first step, the scheduling operations at each level will be conducted in a structure without priority constraints, and the calculation of total hierarchical time is based on the maximum execution time of each server. Therefore, during the allocation process, we only need to ensure that the maximum execution time of each task on different servers is minimized to achieve the slowest hierarchical time growth. This allows sufficient time to be freed up for the subsequent reliability enhancements, ensuring that there is ample sacrificial space available.

3. Reliability optimization. There are various methods to enhance reliability, but in this study, only fault-tolerant approaches are employed for this purpose. The obtained level time in step 2 is recorded as T_2, and the level time T_1 calculated by the ECLS_H algorithm [32] is defined as the upper limit of the level time $T_{deadLine}$. The consumable value of the time can be obtained from $T_{deadLine} - T_2$. Use the consumption value as the reliability improvement space. A task scheduling policy can be obtained that keeps the level time below $T_{deadLine}$, but has a great improvement in reliability index.

5.1. Analysis

In order to accurately determine the initial values of energy levels, this section presents two theorems for establishing lower bounds on task execution and transmission energy, while also considering the transmission reliability in the presence of channel fading. The contents of the theorems and their proof are as follows:

Theorem 1. *When the computing speed of the UE s_0 is set to $s_0^* = (P_s/\xi(\alpha-1))^{1/\alpha}$, the required energy consumption E_0^* is the lowest, with a value of $R_0 P_s^{1-1/\alpha}\xi^{1/\alpha}\alpha/(\alpha-1)^{1-1/\alpha}$, and the corresponding time T_0^* is $R_0(\xi(\alpha-1)/P_s)^{1/\alpha}$.*

Proof. According to the relationship between energy and power and time, there is the following formula:

$$\begin{aligned} E_0 &= P_0 T_0 = (P_d + P_s)(r/s_0) \\ &= (\xi {s_0}^{\alpha-1} + P_s)(r/s_0) \\ &= (\xi {s_0}^{\alpha-1} + P_s/s_0) r. \end{aligned} \tag{20}$$

Taking the partial derivative with respect to s_0 on both sides of the equation, we obtain that

$$\frac{\partial E_0}{\partial s_0} = r\left(\xi(\alpha-1){s_0}^{\alpha-2} - \frac{P_s}{{s_0}^2}\right), \tag{21}$$

when the left side of the equation is 0, so $\xi(\alpha-1){s_0}^{\alpha-2} = P_s/{s_0}^2$, which is actually

$$s_0 = s_0^* = (P_s/\xi(\alpha-1))^{1/\alpha}. \tag{22}$$

From this, we can conclude that

$$E_0^* = (\xi(s_0^*)^{\alpha-1} + P_s/s_0^*)r, \tag{23}$$

$$T_0^* = r/s_0^*. \tag{24}$$

Based on this, we observe that the function of s_0 with respect to E_0 takes on a U-shaped curve. As s_0 increases from 0 to s_0^*, E_0 decreases, but once s_0 is greater than or equal to s_0^*, E_0 increases as s_0 continues to increase. Therefore, with s_0^* as the dividing point, we can always find an s_0 on both sides such that the calculated value of E_0 is the same. In other words, s_0^* represents the lowest point, corresponding to the minimum value of E_0^*. Thus, the proof is complete. ■

Theorem 2. *For the MEC in fog computing, E_j is an increasing function of c_j, and as c_j approaches 0, E_j approaches $E_j^* = \ln 2/\left(w_j\beta_j\left|h_{i,j}\right|^2\right)D_j$, and T_j approaches $T_j^* = R_j/s_j$, for all $1 \le j \le n$.*

Proof. According to the relationship between energy and power and time, there is the following formula:

$$E_j = P_j T_j = P_j(d/c_j) = (2^{c_j/\omega_j} - 1)d/(\beta_j c_j \left|h_j\right|^2), \tag{25}$$

set $y = (2^{c_j/\omega_j} - 1)/c_j$, this also makes it so that

$$y = \frac{e^{\ln 2(c_j/\omega_j)} - 1}{c_j} = \frac{\ln 2}{\omega_j}\cdot\frac{e^{\ln 2(c_j/\omega_j)} - 1}{\ln 2 c_j \omega_j}, \tag{26}$$

set $x = \ln 2(c_j/\omega_j)$, so the equation can be rewritten as:

$$y = \frac{\ln 2}{\omega_j}\cdot\frac{e^x - 1}{x}. \tag{27}$$

Meanwhile, by expanding e^x using Taylor series, we have

$$e^x = \sum_{k=0}^{\infty} x^k/k!. \tag{28}$$

According Eqs. (27) and (28), we can get the functional relationship between y and x:

$$y = \frac{\ln 2}{\omega_j}\cdot\sum_{k=0}^{\infty}\frac{x^{k-1}}{k!}. \tag{29}$$

From Eq. (29), we can see that y is an increasing function of x. At the same time, when c_j approaches 0, x also approaches 0, which leads to $y = \ln 2/\omega_j$, similarly, it will also cause E_j to approach E_j^*, expressed as:

$$E_j^* = \left(\frac{\ln 2}{\omega_j\beta_j\left|h_{i,j}\right|^2}\right)d. \tag{30}$$

Meanwhile, we can conclude that E_j is an increasing function of c_j. The claim for T_j is obvious. ■

5.2. Task scheduling for time-reliability with priority constraints

In the optimization of task offloading scheduling in heterogeneous distributed systems, to ensure that task offloading does not violate the priority constraints of tasks themselves, a common approach is the $rank_u$ method [27,28], which is calculated using the following formula:

$$rank_u(t_i) = \overline{T_i} + \max_{t_j \in \mathrm{succ}(t_i)}\{c_{i,j} + rank_u(t_j)\}, \tag{31}$$

where $\overline{T_i}$ represents the average worst case execution time of task t_i, and all tasks are sorted in descending order based on their ranks for execution. However, in the fog computing, the calculation method of energy has changed, the value of $\overline{T_i}$ cannot be known in advance before the actual offloading, rendering the $rank_u$ method not fully adaptable to task offloading optimization in fog environments.

To address this issue, we partition the DAG into v levels, denoted as $L = (H_1, H_2, \ldots, H_v)$, where each level represents a group of tasks. Level H_1 consists of the initial tasks with no predecessor tasks. To comply with task priority constraints during the scheduling process, we adopt a hierarchical task scheduling approach. This means that tasks in level H_2 can only start execution once all tasks in level H_1 are completed. In other words, each level of the DAG is independently and separately scheduled for task offloading, effectively eliminating the impact of priority constraints, although sacrificing some flexibility due to the coordinated action of levels. This facilitates the optimization of subsequent algorithms.

For data integration and result visualization purposes, we summarize the offloading scheduling strategy as $S = (L_0, L_1, \ldots, L_v)$, where $L_x = (LT_x, LRE_x)$ (for all $1 \le x \le v$), storing the total offloading execution time and total reliability of the tasks in each level.

In Algorithm 1, an energy constrained scheduling algorithm with priority constraints in fog computing is proposed, which is called Energy Constrained Level-by-Level Time-Reliability Optimization scheduling (ECLLTROS).

Algorithm 1 Energy Constrained Level-by-Level Time-Reliability Optimization Scheduling

Input: An application $G = (L, \prec)$ with $L = (t_1, t_2, \cdots, t_m)$, where $t_i = (r_i, d_i)$, for every $1 \le i \le m$, UE $= (\alpha, \xi, P_s)$, MEC$_j = (s_j, w_j, c_j)$, for every $1 \le j \le n$, and $\widetilde{E}$.

```
Output: An optimal computation offloading and power allocation
    strategy which can ensure the total energy consumption $E$ does not
    exceed the threshold $\widetilde{E}$, while achieving dual objective optimization
    of time ($T$) and reliability ($RE$).
 1: for ($x = 1; x \le v; x++$) do
 2:     $LRE_x, LT_x \leftarrow$ ECTROS($E_x, H$);
 3: end for;
 4: $E_{residual} \leftarrow \widetilde{E} - \sum E_x$, for all $1 \le x \le v$;
 5: $E' \leftarrow E_{residual} / K$;
 6: while ($E_{residual} > 0$) do
 7:     if ($E_{residual} \le E'$) then
 8:         $\Delta E \leftarrow E_{residual}$;
 9:     else
10:         $\Delta E \leftarrow \gamma E'$, where $\gamma \in [0.5, 1.0]$;
11:     end if;
12:     $x' \leftarrow$ indexmax $1 \le x \le v$ ($LT_x$ - ECTROS($E_x + \Delta E$, $H$));
13:     $E'_x \leftarrow E'_x + \Delta E$;
14:     $LRE'_x \leftarrow$ ECTROS($E'_x, H$);
15:     $E_{residual} \leftarrow E_{residual} - \Delta E$;
16: end while;
17: $T \leftarrow LT_1 + LT_2 + \ldots + LT_v$;
18: $RE \leftarrow LRE_1 \cdot LRE_2 \cdot \ldots \cdot LRE_v$;
19: return $S, T$ and $RE$.
```

Notation: In this paper, we set

indexmax$(LT_1, LT_2, \ldots, LT_v)$

to be the index x such that $LT_x = \max(LT_1,\ LT_2,\ \cdots,\ LT_v)$.

At this point, tasks within the same level do not have priority constraints, allowing the use of a scheduling algorithm without priority constraints to make decisions on task offloading. This decision-making process will be provided by Algorithm 2. Meanwhile, the initial energy constraint E_x of a single level is determined as follows:

Let $R_x = \sum_{t_i \in L_x} r_i$ and $D_x = \sum_{t_i \in L_x} d_i$, for all $1 \le l \le v$. Then, the following formula can be obtained through Eqs. (23) and (30):

$$E_x = R_x {P_s}^{1-1/\alpha} \xi^{1/\alpha} \frac{\alpha}{(\alpha-1)^{1-1/\alpha}} + \left(\frac{\ln 2}{\min_{1 \le j \le n}(\omega_j \beta_j) \left| h_{min} \right|^2} \right) D_x. \tag{32}$$

Algorithm 1 mainly focuses on determining the optimal allocation of a given energy threshold among the levels v. It calculates the optimal total reliability and total response time values for applying the no-priority-constrained task scheduling algorithm to different levels H_x under the energy constraint E_x (where $1 \le x \le v$).

At the beginning of the algorithm, each level H_x schedules and allocates tasks using the initial energy E_x calculated by Eq. (32) (lines 1–3), resulting in a preliminary strategic result. Next, the remaining energy is calculated by subtracting the sum of energy consumption from all level ($E_1 + E_2 + \cdots + E_v$) from the total energy constraint $\widetilde{E}$, as shown in line 4. The obtained value is then divided by K to determine the upper limit of energy E' for each iteration (line 5). Subsequently, a while loop is executed (lines 6–16), with the loop executing no fewer than K times. In each iteration, the following operations are performed:

(1) Determine ΔE, which is a random value generated by multiplying γ with E', where γ is uniformly distributed in the range [0.5, 1.0] (line 10).

(2) Obtain the level with the maximum optimization magnitude of energy allocation increment ΔE (line 12).

(3) Allocate ΔE based on the results from the previous step. The above while loop terminates after allocating all remaining energy. Determine the overall reliability and response time of the application under the current allocation scenario (lines 17–18).

The time complexity of ECLLTROS is $O((v+K)\times(mn(\log(m)+2)+m^2))$, where scheduling all tasks of all levels can be done in $O(v\times(mn(\log(m)+2)+m^2))$ time, the distribution of the remaining energy costs $O(K \times (mn(\log(m)+2)+m^2))$ time.

5.3. Task-replica scheduling for time-reliability without priority constraints

With Algorithm 1, we have eliminated the priority constraints inherent in the DAG, simplifying the problem to a task offloading scheduling problem within the same level, that is, a task scheduling without priority constraints.

In the past years, many methods have been proposed in the research of pre-power allocation. In constant velocity method, all tasks are assumed to have the same computation speed. However, this approach is not feasible in fog computing because the UE cannot control the computation speed s_j of MEC$_j$, leading to an inability for the UE to regulate the task execution time of MEC, making it unattainable for all tasks to have the same execution time. Therefore, in this paper, we utilize the equal energy method that each task consumes an equal amount of energy, denoted as: $\widetilde{E}/m$.

It should be noted that equal energy method is not an optimal energy distribution method, the energy allocation of tasks within the same level is not optimized, resulting in lower levels of optimization for some tasks. There is still considerable room for improvement. Therefore, we set a reliability threshold η for the final reliability of tasks, if the reliability value of a task does not meet this threshold, the execution of replicas is introduced, and the task is executed simultaneously on multiple servers, which could improve the overall reliability of the application in a fault-tolerant manner.

Meanwhile, from recent related studies, we have observed that in heterogeneous distributed systems, the execution phase of replica addition does not need to consider task offloading and energy. However, the expansion of fog computing environment means that when the UE decide to offload tasks to a fog server, there must be an offloading transmission operation, which implies that the energy for transmission needs to be borne by the UE. Additionally, wireless transmission between servers in the fog computing environment is also feasible, subject to various influencing factors. Therefore, in this paper, we set the energy used by a single replica is the same as that of a task, and when a task decides to execute the replica, only one replica transmission behavior is performed, and the replica data is regarded as being transmitted to the fog, and the fog server can automatically extract it without UE sending it multiple times. Furthermore, considering the significant time sacrifice caused by excessive execution of replicas, we only allow a portion of tasks within the level to have the permission to execute replicas, denoted as the task-replica ratio y. In other words, the number of tasks that can execute replicas within the current level is m/y.

In Algorithm 2, an energy constrained scheduling algorithm of equal energy method without priority constraints is presented, namely, Energy Constrained Time-Reliability Optimization Scheduling (ECTROS).

Algorithm 2 Energy Constrained Time-Reliability Optimization Scheduling

```
Input: A given task list $L = (t_1, t_2, \cdots, t_m)$, where $t_i = (r_i, d_i)$, for all
    $1 \le i \le m$, UE = $(\xi, \alpha, P_s)$, MEC$_j = (s_j, c_j, w_j)$, for all $1 \le j \le n$, and
    $\widetilde{E}$.
Output: An computation offloading and power allocation strategy are
    proposed to ensure that the energy consumption ($E$) does not
    exceed the given threshold $\widetilde{E}$, while achieving optimization of
    reliability ($RE$) under the premise that the time does not exceed
    $T_{deadline}$.
1: Initialize the list $L$ using heuristic $H$;
2: $E_{task} \leftarrow \widetilde{E}/(m + (m/y))$, $E_{replica} \leftarrow (m/y) \cdot E_{task}$;
3: $T_{deadLine} \leftarrow$ ECLS_H($\widetilde{E}, H$);
4: $S_{initial}, T_{minimum} \leftarrow$ ECTM($\widetilde{E} - E_{replica}, H$);
5: for (all $t_i$ in $S_{initial}$) do
6:     Retrieve the reliability list $RE_{list}$ of task $t_i$ execution on all MEC.

7:     if (the replica execution condition is met) then
8:         while (the replica execution condition is met) do
```

```
9:          Add a replica to task $t_i$;
10:         Assign the MEC with the highest reliability in $RE_{list}$, excluding the original execution MEC of $t_i$, as the execution MEC for this replica.
11:       end while;
12:       Refresh $RE_i^{final}$ with Equation (33);
13:       Refresh the time $T_j$ of all servers which executing replicas;
14:       $E_{replica} \leftarrow E_{replica} - E_{task}$;
15:     else
16:       Maintain the original allocation offloading strategy;
17:     end if;
18: end for;
19: $T \leftarrow \max(T_1, T_2, T_3, \cdots, T_n)$;
20: $RE \leftarrow RE_1^{final} \cdot RE_2^{final} \cdot RE_3^{final} \cdot \ldots \cdot RE_m^{final}$;
21: return $S$, $T$ and $RE$.
```

The key of Algorithm 2 is make full use of the interval between the level time and the upper limit time to improve the reliability of the level.

After initializing the task list with heuristic H (line 1), the equal energy method is used to allocate the single task energy to the total energy used by the copy (line 2). The ECLS_H and ECTM algorithms are called respectively to calculate the total time, and the total time obtained by ECLS_H algorithm is set as the time upper limit $T_{deadLine}$, and set the total time obtained by the ECTM algorithm as the initial value of the scheduling strategy (lines 3–4).

Set a reliability threshold η, if $RE_{i,j}{}'$ (for all $i \in [1, m]$) is less than the threshold, add replicas to the current task until the duplicate execution condition is no longer met, and refresh the time data of the task and the replica execution server and the final reliability of the task (lines 7–10). The replica execution conditions are as follows:

(1) The reliability does not exceed the threshold η.

(2) The available server queue is not empty.

(3) The available energy for replica is greater than 0.

(4) The total time of the MEC executing the replica does not exceed $T_{deadLine}$.

The replica calculation is dynamic, so the number of replicas $\varphi_{\text{num}}(t_i)$ required for each task can be different. After the algorithm is calculated, the final reliability value of the task t_i is :

$$RE_i^{final} = 1 - \prod_{MEC_j \in \varphi_{mec}(t_i)} (1 - RE_{i,j}), \tag{33}$$

where $\varphi_{mec}(t_i)$ represents the list of servers where the replica is assigned to execute. If the reliability of the current task is greater than the threshold, it is considered that it does not need replica (line 12).

The time complexity of ECTROS algorithm is $O(mn(\log(m) + 2) + m^2)$, algorithm ECLS takes $O(m(n + m))$ time in line 3 [32], perform replica computation operations in $O(mn)$ time and call Algorithm 3 in $O(mn \log(m))$ time.

5.4. *Task scheduling for time without priority constraints*

The main objective of Algorithm 3 is to maximize the utilization of server queues in the fog computing while keeping the total scheduling length as minimal as possible, which is called Energy Constrained Time Minimization algorithm (ECTM).

Algorithm 3 Energy Constrained Time Minimization Algorithm

```
Input: A given task list $L = (t_1, t_2, \cdots, t_m)$, where $t_i = (r_i, d_i)$, for all $1 \le i \le m$, UE $= (\xi, \alpha, P_s)$, MEC$_j = (s_j, c_j, w_j)$, for all $1 \le j \le n$, and $\widetilde{E}$.
Output: An computation offloading strategy and power allocation strategy are proposed to ensure that the energy consumption ($E$) does not exceed the given threshold $\widetilde{E}$, while minimizing the total time ($T$).
1: Initialize the list $L$ using heuristic $H$;
2: for ($j = 0; j \le n; j++$) do
3:   $T_j \leftarrow 0$;
4: end for;
5: for ($i = 1; i \le m; i++$) do
6:   $RE_i^{final} \leftarrow 1$;
7:   for (all MEC$_j$) do
8:     Offload $t_i$ on MEC$_j$ at time $T_j$;
9:     $RE_{i,j} \leftarrow$ the overall reliability of $t_i$;
10:    $T_{i,j} \leftarrow$ the response time of $t_i$;
11:  end for;
12:  $j' \leftarrow \text{indexmin}(T_{i,0} + T_0, T_{i,1} + T_1, \cdots, T_{i,n} + T_n)$;
13:  Designate MEC$_{j'}$ as the actual execution MEC for task $t_i$;
14:  $RE_i^{final} \leftarrow RE_{i,j}{}'$;
15:  $T_j \leftarrow T_j + T_{i,j}{}'$;
16: end for;
17: $T = \max(T_1, T_2, T_3, \cdots, T_n)$;
18: $RE = RE_1^{final} \cdot RE_2^{final} \cdot RE_3^{final} \cdot \ldots \cdot RE_m^{final}$;
19: return $S$, $T$ and $RE$.
```

The key of Algorithm 3 is to maximize the utilization of the fog servers while allowing the total time to grow at the slowest rate.

After initializing the server time and task data (lines 1–4), set RE_i^{final} to represent the actual reliability of each task (line 6). Using a For loop (lines 7–11), unload a task to all MECs and record the calculated reliability as $RE_{i,j}$, which contains the calculated reliability and communication reliability of the task.

By comparing the minimum value of $T_{i,j} + T_j$, the corresponding index of the server j' is obtained (line 12), so the MEC$_{j'}$ is the actual allocated server for the current task. Meanwhile, the offloading execution time and reliability of the current task when offloaded to MEC$_{j'}$ are obtained (lines 13–14). Therefore, after the loop is completed, the total reliability of the task list, the total offloading execution time and the corresponding scheduling policy can be obtained (lines 16–17).

The time complexity of ECTM is $O(nm\log(m))$, within the time $O(m \times n)$ all tasks scheduling, in $O(\log(m))$ time calculating minimum time consumption value of each task.

6. Experiments and discussion

In order to evaluate the task optimization effect of the proposed ECLLTROS algorithm in fog computing, corresponding experiments are designed in this section. The program is realized with JAVA, and the results have been carried out on a workstation with an Intel Core i5-11400 processor running at 2.60 GHz and 8 GB RAM.

6.1. *Parameter setting*

In this study, we consider a fog computing environment consisting of one UE and $n = 7$ MEC servers. The UE is configured with the following parameters: $\alpha = 2.0$, $\xi = 0.1$, $P_s = 0.05$ Watts, $\lambda = 0.0015$. Each MEC$_j$ is configured with the following parameters: $s_j = 3.1 - 0.1j$ BI/second, $\beta_j = 5.25 - 0.25j$ W s^{-1}, $w_j = 2.9 + 0.1j$ MB/second, $\lambda_j = 0.0015 + 0.0002j$, where j ranges from 1 to n (These parameters are set in a way similar to existing studies [19,32]).

In addition, based on the Rayleigh random fading model described above, the channel power fading coefficient is considered as a continuous random variable greater than zero. For simplicity in problem analysis and experiments, this paper assumes that the value of $\left|h_{i,j}\right|^2$ during the actual scheduling process is a fixed discrete value within the range of $\{0.4, 0.6, 0.8, 1.0\}$. Therefore, the value of $\left|h_{min}\right|^2$ is set to 0.4. Meanwhile, the fading coefficient σ_j^2 follows an independent uniform distribution within the range of [0.5, 0.9]. The SNR threshold $\theta_{0,j} = 0.5 + 0.1j$ db, where $0 \le j \le n$, and $0 \le i \le m$.

To represent and configure mobile application G, this study adopts a random DAG. In the past, the widely used random DAG generation methods in the task scheduling field were *Fan-in/Fan-out* [39] and

Table 2
Scheduling length for energy constrained level-by-level scheduling (95% C.I. = ±1.32241%).

	20	40	60	80	100	120	140	160	180	200
ORG	6.730777	11.243410	14.777211	17.844110	20.683817	22.874211	25.673958	27.238373	29.761461	31.508853
SRF	7.032300	11.562170	15.218487	18.762513	22.193556	24.492971	27.525085	29.523959	31.995812	34.113047
LRF	**6.240248**	**10.072959**	**12.699155**	**15.479339**	**18.388477**	**20.128623**	**22.506501**	**24.112373**	**26.608650**	**28.631861**
SDF	6.803812	11.331525	15.064040	18.297758	21.036884	23.405188	26.075848	28.097571	30.064148	32.032772
LDF	6.640983	11.068259	14.462160	17.513598	20.255224	22.280815	25.072499	26.906293	29.405372	31.232114
SRD	6.805786	11.327465	14.837259	18.154730	20.974092	23.186709	26.035815	28.223866	30.670089	32.457092
LRD	6.631069	10.929694	14.261595	17.361410	20.072169	21.948904	24.668660	26.307002	28.636455	30.346656

Table 3
Scheduling length for energy constrained level-by-level time-reliability optimization scheduling (95% C.I. = ±1.78491%).

	20	40	60	80	100	120	140	160	180	200
ORG	5.529705	9.353080	12.281459	14.809853	16.552394	17.819718	18.214560	19.881652	20.333724	21.567056
SRF	5.635894	9.667544	13.083405	15.450959	18.007469	19.422108	21.490574	22.831961	23.880844	24.680749
LRF	**5.146271**	**8.908753**	**11.943826**	**13.509538**	**14.923893**	**15.977928**	**16.985604**	**18.193393**	**19.208595**	**20.056841**
SDF	5.645221	9.429969	12.493057	15.044648	17.077330	18.283802	19.766866	20.337494	21.082818	22.017111
LDF	5.525499	9.145814	12.111658	14.224195	16.160288	17.137794	18.441901	19.123425	20.120409	21.092012
SRD	5.444014	9.390965	12.659561	15.084257	17.152453	18.443375	19.955077	21.182584	22.107998	23.358586
LRD	5.208784	9.096155	12.047926	13.742382	15.994254	16.887666	17.729355	18.570028	19.424517	20.374670

Table 4
Reliability result for energy constrained level-by-level scheduling (95% C.I. = ±2.34511%).

	20	40	60	80	100	120	140	160	180	200
ORG	0.793931	0.434511	0.323639	0.315200	0.304612	0.303267	0.303074	0.290280	0.286506	0.285850
SRF	0.801347	0.476853	0.340827	0.295861	0.260420	0.254653	0.250528	0.248936	0.248470	0.238907
LRF	**0.824353**	**0.555171**	**0.495774**	**0.465800**	**0.434344**	**0.428839**	**0.426450**	**0.419150**	**0.417194**	**0.409947**
SDF	0.789203	0.440024	0.302287	0.293987	0.283431	0.282724	0.280508	0.278620	0.277662	0.274052
LDF	0.816261	0.458141	0.348229	0.338806	0.322448	0.318400	0.308888	0.306952	0.290992	0.288907
SRD	0.802725	0.452946	0.332205	0.307504	0.296477	0.292595	0.277432	0.264784	0.263000	0.262044
LRD	0.790219	0.471309	0.368541	0.348565	0.346837	0.340185	0.340182	0.334702	0.328007	0.325249

Table 5
Reliability result for energy constrained level-by-level time-reliability optimization scheduling (95% C.I. = ±2.56793%).

	20	40	60	80	100	120	140	160	180	200
ORG	0.650516	0.565825	0.520844	0.508587	0.484851	0.460499	0.438398	0.421204	0.405598	0.392064
SRF	0.651472	0.597322	0.563288	0.536058	0.502490	0.460073	0.423993	0.396239	0.379458	0.376508
LRF	**0.711868**	**0.642316**	**0.602133**	**0.583635**	**0.568184**	**0.535915**	**0.521045**	**0.505131**	**0.488021**	**0.469651**
SDF	0.619731	0.537685	0.508914	0.474814	0.459834	0.449426	0.431538	0.405341	0.386135	0.379069
LDF	0.667283	0.571029	0.548172	0.537431	0.512628	0.498531	0.475211	0.450058	0.434046	0.408924
SRD	0.660859	0.548623	0.511194	0.488914	0.465561	0.455907	0.434716	0.413748	0.387684	0.379311
LRD	0.681893	0.616023	0.574343	0.554297	0.531476	0.519880	0.492781	0.476107	0.452560	0.435786

G(m,p) [40]. While the Fan-in/Fan-out method could employ the *TGFF v3.0* [41] tool to generate random DAGs, it could not guarantee the generation of DAGs with a fixed number of tasks. Therefore, we adopted the *G(m, p)* method in that study. In the *G(m, p)* method, the value of m was determined based on specific experiments. Specifically, a random DAG with m nodes and an arc probability p is generated using the following method:

For any pair of tasks t_i and t_j in G, where $1 \le i \le j \le m$, the probability of having a priority constraint is p. In other words, there is a directed arc from t_i to t_j between the corresponding nodes in the random DAG with a probability of p. Following the generation method mentioned above, for task t_i, the expected number of subsequent tasks is $(m-i)p$, where $1 \le i \le m$. By setting $p = b/m$, where b is set to 2 in this paper, we can observe that this number falls within the range of $[0, b)$.

The task computation and communication requirements are generated randomly. The r_i values are independently and uniformly distributed in the range [3.5, 5.0]. Similarly, the d_i values are also independently and uniformly distributed in the range [2.0, 4.0].

The following heuristics for the initial order of the task list $L = (t_1, t_2, \dots, t_m)$ are considered in this article:

• ORG (*Original Order*) – Tasks are arranged in their original order.
• SRF (*Smallest Requirement First*) – Tasks are ordered in ascending order of their computation requirements: $r_1 \le r_2 \le \cdots \le r_m$.
• LRF (*Largest Requirement First*) – Tasks are ordered in descending order of their computation requirements: $r_1 \ge r_2 \ge \cdots \ge r_m$.
• SDF (*Smallest Data First*) –Tasks are ordered in ascending order of their communication requirements: $d_1 \le d_2 \le \cdots \le d_m$.
• LDF (*Largest Data First*) –Tasks are ordered in descending order of their communication requirements: $d_1 \ge d_2 \ge \cdots \ge d_m$.
• SRD (*Smallest Requirement Data Ratio First*) – Tasks are ordered in ascending order of their computation-to-communication ratio: $r_1/d_1 \le r_2/d_2 \le \cdots \le r_m/d_m$.
• LRD (*Largest Requirement Data Ratio First*) – Tasks are ordered in descending order of their computation-to-communication ratio: $r_1/d_1 \ge r_2/d_2 \ge \cdots \ge r_m/d_m$.

6.2. Comparative experiment with different task number

The purpose of this experiment is to observe the optimization effect of the algorithm on the overall reliability and scheduling length under the randomly generated DAG structure model of different tasks. We set the number of tasks $m = 20, 40, 60, 80, \cdots, 200$, energy consumption constraint $E = 24$ m − 72 J, and the task-replica ratio of y is 4:1, the remaining parameters are all parameters in Section 6.1. A total of 200 random structure DAGs were generated in the simulation experiment, and the average experimental results are shown in Tables 2–5. The maximum 95% confidence interval (C.I.) of this experiment is ±2.56793%.

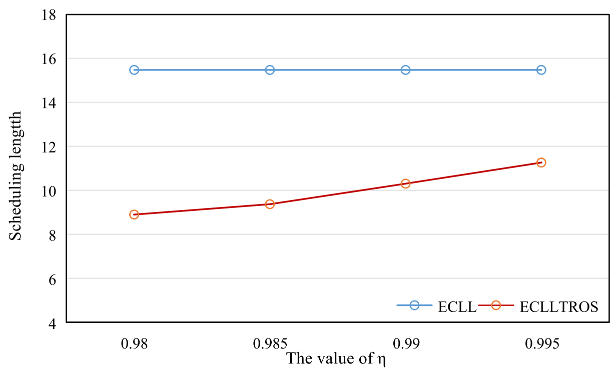

Fig. 4. Scheduling length of algorithms (LRF) with different reliability thresholds (95% C.I. = ±1.729251%).

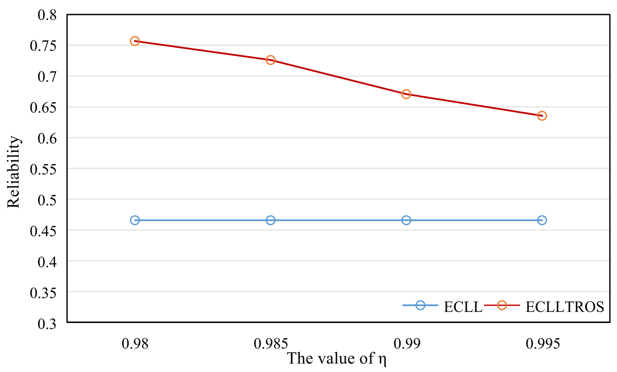

Fig. 5. Reliability of algorithms (LRF) with different reliability thresholds (95% C.I. = ±2.93989%).

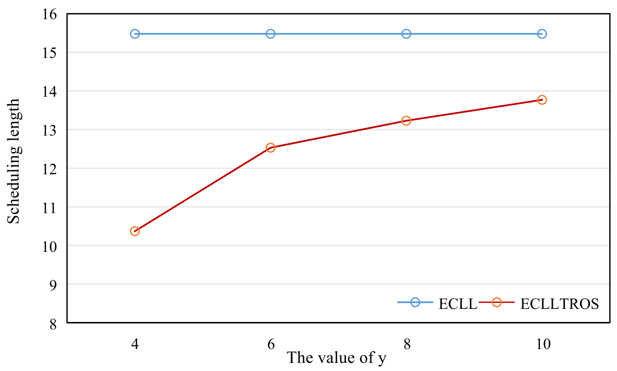

Fig. 6. Scheduling length of algorithms (LRF) with different ratio of replica (95% C.I. = ±1.88425%).

As can be seen in Tables 2 and 3, the scheduling length increases with the increase of m. Additionally, ECLLTROS implements control over the scheduling length, ensuring that the overall optimization results after adding replicas do not exceed the scheduling length of ECLL. This achieves optimization in terms of the time metric. Under the same number of tasks, the best result of ECLLTROS in terms of time index is LRF, while compared with the result of ECLL, the largest optimization improvement is LRD, which has a maximum time improvement of 29.41%.

The results from Tables 4 and 5 indicate a decrease in reliability with an increasing value of m. Notably, the ECLLTROS algorithm demonstrates significant advantages in enhancing reliability compared to the ECLL algorithm, with the influence of replica, the reliability of the ECLLTROS algorithm decreases in a more gradual manner. This advantage stems from the ECLLTROS algorithm's trade-off of time for higher reliability, as evidenced by the scheduling length data. The maximum reliability improvement under the same number of tasks is 92.95%. However, it should be noted that when the number of tasks is too small (m = 20), the difference between the upper and lower limits of time is small due to the small base number, and the reliability improvement space is not large, resulting in the final reliability optimization effect of ECLLTROS is not as good as that of the ECLL algorithm. Therefore, the algorithm is more practical when the number of tasks has a certain scale.

Therefore, based on the experiment, it can be concluded that, at the same task level, LRF is the most effective heuristic sequence for both scheduling length and reliability. Moreover, the optimization effect of the algorithm improves as the number of tasks involved in the offloading allocation increases.

6.3. Comparative experiment with different reliability thresholds

The purpose of the experiment in this section is to observe the reliability and scheduling length optimization effects of randomly generated DAG structure models under different reliability thresholds. The number of tasks set in this section is $m = 80$, and the reliability threshold is $\eta = 0.980, 0.985, 0.990$, and 0.995, the task-replica energy ratio of y is 4. Other parameters are set in reference to Section 6.1, and the simulation experiment results (using the LRF heuristic order as an example) are shown in Figs. 4 and 5. For all the data in this experiment, the maximum 95% C.I. is ±2.93989%.

From Figs. 4 and 5, it can be seen that both the reliability and scheduling length metrics in the experimental results outperform the ECLL scheduling results. As η gradually increases, the scheduling length exhibits an increasing trend. At the same time, due to the high reliability threshold of duplicate execution, some tasks that are close to the threshold preoccupy the qualification of duplicate execution, and some tasks that can be improved by a higher margin cannot be optimized, resulting in a decreasing reliability.

6.4. Comparative experiment with different task-replica ratio

The purpose of the experiment in this section is to observe the reliability and scheduling length optimization effect of randomly generated DAG structure models under different task-replica energy ratios. The number of tasks set in this section is $m = 80$, the reliability threshold is $\eta = 0.990$, and the task-replica ratio y is 2, 4, 6, ⋯, 10 respectively for experiments, other parameters are set in Section 6.1. The simulation experiment results (using the LRF heuristic order as an example) are shown in Figs. 6 and 7. For all the data in this experiment, the maximum 95% C.I. is ±2.94371%.

From Figs. 6 and 7, it can be observed that the fewer the executable copies in the task list, the worse the reliability improvement effect. Even when $y = 10$, the reliability optimization result of ECLLTROS algorithm is lower than that of ECLL, which also leads to a significant time increase.

6.5. Comparative experiment under a real-life industrial example

To evaluate the performance improvement of the algorithm in real-life vehicular networking applications, we utilized a real automotive function example shown in Fig. 8 adopted from [17,28] for experiment.

This application comprises six modules: an engine controller (t_1–t_7), an automatic gearbox ($t_8 - t_{11}$), an anti-lock braking system ($t_{12} - t_{17}$), a wheel angle sensor($t_{18} - t_{19}$), a suspension controller ($t_{20} - t_{24}$), and bodywork tasks ($t_{25} - t_{31}$). In this experiment, the application was deployed on 8 processors, including UE. We set the energy constraint $E = 24\ \text{m} - 72$ J, task replicas ratio y to 4:1, reliability threshold η to 0.999, and kept other parameters consistent with the settings outlined in Section 6.1. A total of one hundred simulation experiments were

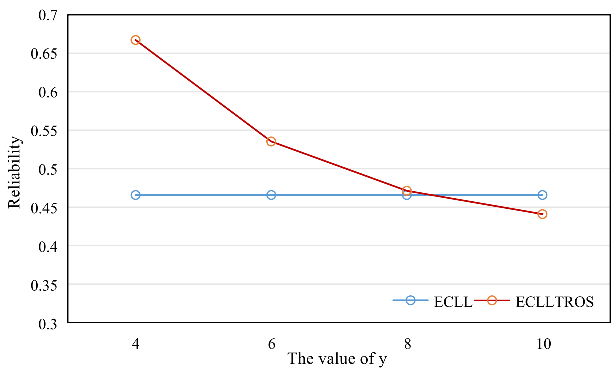

Fig. 7. Reliability of algorithms (LRF) with different ratio of replica (95% C.I. = ±2.94371%).

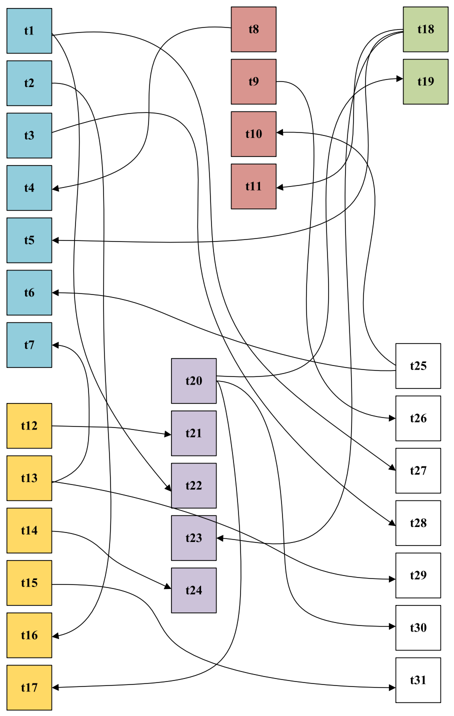

Fig. 8. A real-life automotive application.

Table 6
Experimental results of real vehicular networking application instances (95% C.I. = ±2.78398%).

Heuristics	ECLL		ECLLTROS	
	Reliability	Scheduling length	Reliability	Scheduling length
ORG	0.454682	5.809641	0.692454	4.216541
SRF	0.487353	5.937174	0.666322	4.302656
LRF	**0.540498**	**5.466762**	**0.792340**	**3.878570**
SDF	0.446692	5.928987	0.654195	4.354953
LDF	0.458675	5.744615	0.744626	4.046113
SRD	0.479681	5.838692	0.716017	4.164256
LRD	0.458414	5.850090	0.709262	4.186099

conducted, with the mean results presented in Table 6. The maximum 95% C.I. is ±2.78398%.

Based on the data in Table 6, it is evident that in real vehicular networking application instances, the ECLLTROS algorithm demonstrates superior performance in both time and reliability metrics compared to the ECLL algorithm. Furthermore, the experimental results further validate the pattern described in Section 6.2, namely, the significant optimization effect of the LRF heuristic sequence. This finding further emphasizes the practicality and advanced nature of the ECLLTROS algorithm in real-life applications.

7. Conclusion

This study aims to optimize the time and reliability metrics of fog computing applications under energy constraints. A dual-objective optimization algorithm ECLLTROS is proposed in this paper, targeting time and reliability with task replica energy constraints. Experimental results demonstrate that when the number of tasks reaches a certain scale, compared to the ECLL algorithm, the ECLLTROS algorithm can further enhance the overall reliability of applications while optimizing the time metric, making the algorithm more practical and advanced. This conclusion is further confirmed in experiments on a real-life automotive application. In future work, we will investigate the impact of changing the number of intermediate layers in scheduling optimization for randomly generated DAGs. Additionally, exploring different energy allocation strategies will be another research direction for us.

CRediT authorship contribution statement

Ruihua Liu: Writing – original draft, Methodology. **Wufei Wu:** Writing – review & editing, Writing – original draft, Resources, Investigation, Formal analysis. **Xiaochuan Guo:** Data curation. **Gang Zeng:** Writing – review & editing. **Keqin Li:** Writing – review & editing, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China under Grant No. 62002147, and the China Postdoctoral Science Foundation under Grant No. 2020TQ0134.

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.future.2024.05.014.

References

[1] Q.T. Minh, E. Kamioka, S. Yamada, CFC-ITS: Context-aware fog computing for intelligent transportation systems, IT Prof. 20 (6) (2018) 35–45.
[2] D. Xue, Task offload optimization management of networked vehicles in edge computing environment, in: International Signal Processing, Communications and Engineering Management Conference, Montreal, ON, Canada, 2022, pp. 38–42.
[3] G. Xie, Y. Chen, R. Li, K. Li, Hardware cost design optimization for functional safety-critical parallel applications on heterogeneous distributed embedded systems, IEEE Trans. Ind. Inform. 14 (6) (2018) 2418–2431.
[4] J. Kwak, Y. Kim, J. Lee, S. Chong, DREAM: Dynamic resource and task allocation for energy minimization in mobile cloud systems, IEEE J. Sel. Areas Commun. 33 (12) (2015) 2510–2523.
[5] E. Cuervo, et al., MAUI: Making smartphones last longer with code offload, in: Proc. ACM 8th Int. Conf. Mobile Syst. Appl. Services, San Francisco, CA, USA, 2010, pp. 49–62.
[6] D. Contini, L.F.S. de Castro, E. Madeira, S. Rigo, L.F. Bittencourt, Simulating smart campus applications in edge and fog computing, in: 2020 IEEE International Conference on Smart Computing, Bologna, Italy, 2020, pp. 326–331.

[7] H.-J. Hong, From cloud computing to fog computing: Unleash the power of edge and end devices, in: IEEE International Conference on Cloud Computing Technology and Science, Hong Kong, China, 2017, pp. 331–334.
[8] R. Jindal, N. Kumar, H. Nirwan, MTFCT: A task offloading approach for fog computing and cloud computing, in: 10th International Conference on Cloud Computing, Data Science & Engineering, Noida, India, 2020, pp. 145–149.
[9] J. Garcia, E. Simó, X. Masip-Bruin, E. Marín-Tordera, S. Sànchez-López, Do we really need cloud? Estimating the fog computing capacities in the City of Barcelona, in: IEEE/ACM International Conference on Utility and Cloud Computing Companion, Zurich, Switzerland, 2018, pp. 290–295.
[10] M.-R. Ra, A. Sheth, L. Mummert, P. Pillai, D. Wetherall, R. Govindan, Odessa: enabling interactive perception applications on mobile devices, in: Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services, Bethesda, Maryland, 2011, pp. 43–56.
[11] G. Xie, J. Jiang, Y. Liu, R. Li, K. Li, Minimizing energy consumption of real-time parallel applications using downward and upward approaches on heterogeneous systems, IEEE Trans. Ind. Inform. 13 (3) (2017) 1068–1078.
[12] Y. Liu, J. Liu, Z. Zhu, C. Deng, Z. Ren, X. Xu, Adaptive fault-tolerant scheduling in heterogeneous real-time systems, in: 14th IEEE Conference on Industrial Electronics and Applications, ICIEA, Xi'an, China, 2019, pp. 982–987.
[13] X. Xiao, G. Xie, R. Li, K. Li, Minimizing schedule length ofenergy consumption constrained parallel applications on heterogeneous distributed systems, in: Trustcom/BigDataSE/ISPA, IEEE, IEEE, 2016, pp. 1471–1476.
[14] J. Niu, C. Liu, Y. Gao, M. Qiu, Energy efficient task assignment with guaranteed probability satisfying timing constraints for embedded systems, IEEE Trans. Parallel Distrib. Syst. 25 (8) (2014) 2043–2052.
[15] M. Naghibzadeh, Modeling and scheduling hybrid workflows of tasks and task interaction graphs on the cloud, Future Gener. Comput. Syst. 65 (2016) 33–45.
[16] Z. Tang, L. Qi, Z. Cheng, K. Li, S.U. Khan, K. Li, An energy-efficient task scheduling algorithm in dvfs-enabled cloud environment, J. Grid Comput. 14 (1) (2016) 55–74.
[17] G. Xie, J. Jiang, Y. Liu, R. Li, K. Li, Minimizing energy consumption of real-time parallel applications using downward and upward approaches on heterogeneous systems, IEEE Trans. Ind. Inform. (2017).
[18] G. Xie, G. Zeng, R. Li, K. Li, Energy-aware processor merging algorithms for deadline constrained parallel applications in heterogeneous cloud computing, IEEE Trans. Sustain. Comput. (2017).
[19] Li K., Heuristic computation offloading algorithms for mobile users in fog computing, ACM Trans. Embedd. Comput. Syst. 20 (2) (2021) 11, 28pp..
[20] R. Liu, H. Huang, Y. He, X. Guo, C. Yan, J. Dai, W. Wu, Reliability Optimization Scheduling and Energy Balancing for Real-Time Application in Fog Computing Environment, in: Advanced Parallel Processing Technologies, Springer Science and Business Media LLC, Singapore, 2023.
[21] J. Liu, K. Li, D. Zhu, J. Han, K. Li, Minimizing cost of scheduling tasks on heterogeneous multicore embedded systems, ACM Trans. Embedd. Comput. Syst. 16 (2) (2016) 36.
[22] J. Liu, Q. Zhuge, S. Gu, J. Hu, G. Zhu, E.H.-M. Sha, Minimizing system cost with efficient task assignment on heterogeneous multicore processors considering time constraint, IEEE Trans. Parallel Distrib. Syst. 25 (8) (2014) 2101–2113.
[23] V. Vijayalakshmi, M. Saravanan, An extensive analysis of task scheduling algorithms based on fog computing Qos metrics, in: International Conference on Innovative Computing, Intelligent Communication and Smart Electrical Systems, Chennai, India, 2022, pp. 1–8.
[24] A. Avizienis, J.-C. Laprie, B. Randell, C. Landwehr, Basic concepts and taxonomy of dependable and secure computing, IEEE Trans. Dependable Secure Comput. 1 (1) 11–33.
[25] S.M. Shatz, J.-P. Wang, Models and algorithms for reliability oriented task-allocation in redundant distributed-computer systems, IEEE Trans. Reliab. 38 (1) (1989) 16–27.
[26] L. Zhao, Y. Ren, Y. Xiang, K. Sakurai, Fault-tolerant scheduling with dynamic number of replicas in heterogeneous systems, in: IEEE 12th International Conference on High Performance Computing and Communications, Melbourne, VIC, Australia, 2010, pp. 434–441.
[27] G. Xie, Y. Chen, Y. Liu, Y. Wei, R. Li, K. Li, Resource consumption cost minimization of reliable parallel applications on heterogeneous embedded systems, IEEE Trans. Ind. Inform. (2017).
[28] N. Yuan, G. Xie, R. Li, X. Chen, An effective reliability goal assurance method using geometric mean for distributed automotive functions on heterogeneous architectures, in: 2017 IEEE International Symposium on Parallel and Distributed Processing with Applications and 2017 IEEE International Conference on Ubiquitous Computing and Communications, 2017, pp. 667–674.
[29] G. Xie, G. Zeng, R. Li, K. Li, Quantitative fault-tolerance for reliable workflows on heterogeneous IaaS clouds, IEEE Trans. Cloud Comput. 8 (4) (2020) 1223–1236.
[30] S. Wang, K. Li, J. Mei, K. Li, Y. Wang, A task scheduling algorithm based on replication for maximizing reliability on heterogeneous computing systems, in: IEEE International Parallel & Distributed Processing Symposium Workshops, Phoenix, AZ, USA, 2014, pp. 1562–1571.
[31] S.M.T. Khan, L. Barik, A. Adholiya, S.S. Patra, A.N. Brahma, R.K. Barik, Task offloading scheme for latency sensitive tasks in 5G IOHT on fog assisted cloud computing environment, in: 3rd International Conference for Emerging Technology, Belgaum, India, 2022, pp. 1–5.
[32] K. Li, Scheduling precedence constrained tasks for mobile applications in fog computing, IEEE Trans. Serv. Comput. (2022).
[33] A. AlZailaa, H.R. Chi, A. Radwan, R. Aguiar, Low-latency task classification and scheduling in fog/cloud based critical e-health applications, in: IEEE International Conference on Communications, Montreal, QC, Canada, 2021, pp. 1–6.
[34] A. Tsega, A.B. Habtie, Deadline aware data offloading in fog computing, in: IEEE/ACM 15th International Conference on Utility and Cloud Computing, Vancouver, WA, USA, 2022, pp. 248–254.
[35] A. Beloglazov, R. Buyya, Y.C. Lee, A. Zomaya, A taxonomy and survey of energy-efficient data centers and cloud computing systems, in: Advances in Computers, vol. 82, Elsevier, 2011, pp. 47–111.
[36] K. Li, Computation offloading strategy optimization with multiple heterogeneous servers in mobile edge computing, IEEE Trans. Sustain. Comput..
[37] S.M. Azimi, O. Simeone, O. Sahin, P. Popovski, Ultra-reliable cloud mobile computing with service composition and superposition coding, in: Annual Conference on Information Science and Systems, Princeton, NJ, USA, 2016, pp. 442–447.
[38] H.-C. Yang, S. Choi, M.-S. Alouini, Ultra-reliable low-latency transmission of small data over fading channels: A data-oriented analysis, IEEE Commun. Lett. 24 (3) (2020) 515–519.
[39] D. Cordeiro, G. Mounié, S. Perarnau, D. Trystram, J. Vincent, F. Wagner, Random graph generation for scheduling simulations, in: Proceedings of the 3rd International Conference on Simulation Tools and Techniques, Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, Brussels, BEL, 2010, 60, 1–10.
[40] A. Yano, T. Azumi, RD-Gen: Random DAG generator considering multi-rate applications for reproducible scheduling evaluation, in: 2023 IEEE 26th International Symposium on Real-Time Distributed Computing, Nashville, TN, USA, 2023, pp. 21–31.
[41] K. Vallerio, Task graphs for free (TGFF v3. 0). Official version released in April, 2008, p. 15.

Ruihua Liu is currently working toward the M.S. degree in the Department of Computer Technology in the School of Mathematics and Computer Science of Nanchang University, Nanchang, China. His research focuses on edge computing and task offload scheduling optimization.

Wufei Wu received the Ph.D. degree in College of Computer Science and Electronic Engineering from Hunan University, Changsha, China, in 2018. He was a Visiting Scholar at the Nagoya University, Nagoya, Japan, from 2016 to 2017. He is currently a lecturer at the School of Information Engineering, Nanchang University. He has worked on the scheduling optimization for distributed systems, embedded real-time system, and cybersecurity enhancements for in-vehicle networks. He is a senior member of IEEE and senior member of CCF.

Xiaochuan Guo is currently working toward the M.S.degree in the School of Information Engineering of Nanchang University. Nanchang, China. His research focuses on fog computing and multi-objective optimization.

Gang Zeng received the Ph.D. degree in information science from Chiba University, Chiba, Japan, in 2006. He is currently an Associate Professor with the Graduate School of Engineering, Nagoya University, Nagoya, Japan. From 2006 to 2010, he was a Researcher and then Assistant Professor with the Center for Embedded Computing Systems, Graduate School of Information Science, and Nagoya University. His research interests mainly include power-aware computing, real-time embedded system design. He is also the Member IPSJ.

Keqin Li is a SUNY Distinguished Professor of computer science with the State University of New York. He is also a National Distinguished Professor with Hunan University, China. His current research interests include cloud computing, fog computing and mobile edge computing, energy-efficient computing and communication, embedded systems and cyber–physical systems, heterogeneous computing systems, big data computing, high-performance computing, CPU–GPU hybrid and cooperative computing, computer architectures and systems, computer networking, machine learning, intelligent and soft computing. He has authored or coauthored over 845 journal articles, book chapters, and refereed conference papers, and has received several best paper awards. He holds over 70 patents announced or authorized by the Chinese National Intellectual Property Administration. He is among the world's top 5 most influential scientists in parallel and distributed computing in terms of both single-year impact and career-long impact based on a composite indicator of Scopus citation database. He has chaired many international conferences. He is currently an associate editor of the ACM Computing Surveys and the CCF Transactions on High Performance Computing. He has served on the editorial boards of the IEEE Transactions on Parallel and Distributed Systems, the IEEE Transactions on Computers, the IEEE Transactions on Cloud Computing, the IEEE Transactions on Services Computing, and the IEEE Transactions on Sustainable Computing. He is an IEEE Fellow. He is also a Member of Academia Europaea (The Academy of Europe).