Reliability-Enhanced Task Offloading in Mobile Edge Computing Environments

Jialei Liu, Ao Zhou, *Member, IEEE*, Chunhong Liu, *Member, IEEE*, Tongguang Zhang, Lianyong Qi, *Member, IEEE*, Shangguang Wang, *Senior Member, IEEE*, and Rajkumar Buyya, *Fellow, IEEE*

Abstract—**Internet of Things (IoT) devices have become an integral part of our lives and are increasingly used in almost every field. Subsequently, there are a large number of latency-sensitive IoT applications (e.g., face recognition and autonomous driving) targeted for mobile edge computing environments. These IoT applications are often split into multiple collaborative tasks and offloaded onto containers or virtual machines (VMs) with certain failure rates and recovery rates. If these containers or VMs are not deployed in the same edge servers, the bandwidth resources of edge clouds must be consumed to transfer data. These factors increase the completion time of IoT applications to different degrees, and then affect their reliability level. Therefore, there exists equilibrium between the reliability level and bandwidth consumption. In this article, we investigate the equilibrium of minimizing the bandwidth consumption of IoT applications while maximizing the reliability level of these IoT applications during task offloading. We propose a multiobjective optimization problem, and transform it to a single-objective optimization problem. Furthermore, we introduce two efficient approaches to acquire two near-optimal solutions. The results of simulation experiments demonstrate that our proposed approaches can observably enhance the reliability level and reduce the bandwidth consumption of IoTapplications compared with other related approaches. Meanwhile, we also make a comparative analysis of our proposed approaches.**

Index Terms—**Bandwidth consumption, Internet of Things (IoT) application, mobile edge computing, reliability level, task offloading.**

Manuscript received May 3, 2021; revised July 24, 2021; accepted September 14, 2021. Date of publication September 27, 2021; date of current version June 23, 2022. This work was supported in part by the National Key Research and Development Program of China under Grant 2020YFB1805502; in part by the National Natural Science Foundation of China under Grant 61922017, Grant 62032003, and Grant 61921003; in part by the Key Science and Technology Research Project of Henan Province under Grant 202102210146, Grant 202102210163, and Grant 202102210152; in part by the Key Science and Technology Research Project of Anyang City under Grant 2021C01GX017; and in part by the Research and Cultivation Fund Project of Anyang Normal University under Grant AYNUKPY-2019-24. *(Corresponding author: Shangguang Wang.)*

Jialei Liu is with the School of Software Engineering, Anyang Normal University, Anyang 454000, Henan, China (e-mail: jialeiliu@aynu.edu.cn).

Ao Zhou and Shangguang Wang are with the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China (e-mail: aozhou@bupt.edu.cn; sgwang@bupt.edu.cn).

Chunhong Liu is with the School of Computer and Information Engineering, Henan Normal University, Xinxiang 453003, Henan, China (e-mail: lch@htu.edu.cn).

Tongguang Zhang is with the School of 3-D Printing, Xinxiang University, Xinxiang 453003, Henan, China (e-mail: jsjoscpu@163.com).

Lianyong Qi is with the School of Information Science and Engineering, Qufu Normal University, Jining 273100, Shandong, China (e-mail: lianyongqi@qfnu.edu.cn).

Rajkumar Buyya is with the Cloud Computing and Distributed Systems Laboratory, School of Computing and Information Systems, The University of Melbourne, Melbourne, VIC 3010, Australia (e-mail: rbuyya@unimelb.edu.au).

Digital Object Identifier 10.1109/JIOT.2021.3115807

I. Introduction

With the rapid progress in software and hardware technologies, the number of Internet of Things (IoT) devices, such as wearable devices, Raspberry Pi, and smartphones has increased dramatically and have become ubiquitous in our modern digital society. It is predicted that approximately 29 billion IoT devices will be connected to the Internet by 2022 [1]. Subsequently, these IoT devices generate massive latency-sensitive IoT applications, which have stringent delay requirements (e.g., real-time responses on a timescale of 10 ms or even 1 ms [2]) and require a large number of processing and bandwidth resources [3], [4]. However, since these IoT devices often have limited resources, such as processing capacity, bandwidth, and storage space, some complex IoT applications (e.g., face recognition, augmented reality, and autonomous driving) cannot be handled locally and effectively [5]–[7].

To alleviate the resource capacity limitation of these IoT devices, some latency-sensitive IoT applications are typically split into multiple collaborative tasks and offloaded onto edge clouds for processing by containers or virtual machines (VMs) with heterogeneous failure rates and recovery rates [8], [9]. If these containers or VMs are not on the same edge servers, the communication between these collaborative tasks will consume certain bandwidth resources and communication time [10]–[12]. Moreover, since mobile edge computing enhances other Quality of Service (QoS) properties (e.g., low latency) at the expense of reliability [13], which represents the likelihood of a network component operating normally without failure within a specific time [14], a mobile edge computing environment is susceptible to all kinds of failures [15]. These failures can influence the delivery of data and the processing of requests, and then result in an increase in data loss and network delays. Presently, the major causes of reliability decreases are failures or errors of containers or VMs in cloud/mobile edge computing environments [16]. Therefore, frequent interruptions of IoT applications are caused by low

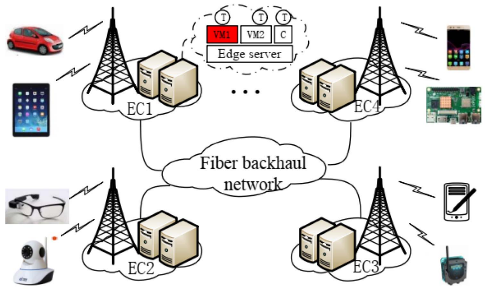

Fig. 1. Reliability-enhanced mobile edge computing system.

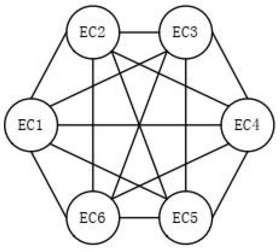

Fig. 2. Full mesh topology of edge clouds.

reliability, and then lead to a significant loss of business revenue. For example, 54 of the top 100 online retailer services are reportedly affected by a four-hour outage of AWS in 2017, which caused U.S. $150 million in losses [17]. To make matters worse, the current services have less tolerance for outages: the average outage cost of a data center has increased significantly (i.e., U.S. $505,502 in 2010 to U.S. $740,357 in 2016) [18].

Compared with general distributed systems (e.g., cloud computing), the mobile edge computing environment typically has limited processor resources (e.g., processing and bandwidth), and each IoT application cooperatively processed by multiple containers or VMs has a stringent delay requirement. To enhance the reliability of a mobile edge computing environment, failure recovery techniques, such as rollforward/rollback or checkpointing widely applied to cloud computing systems cannot be efficiently or quite effectively used in the mobile edge computing environment. This outcome is observed because these techniques usually consume plenty of processor resources to maintain a degree of reliability [17], [19], [20]. Meanwhile, the total time that the container or VM processes each task of the IoT application includes not only the processing time but also the recovery time after failures, if these recovery techniques are applied to mobile edge computing, they will prolong the completion time of the IoT application. Since latency-sensitive IoT applications usually have a certain deadline, the application completion time beyond this deadline degrades the QoS of users. Furthermore, the reliability of the IoT application is also reduced [14]. Therefore, to decrease the resource consumption of the edge clouds and enhance the reliability of the IoT application, obtaining a reliability-enhanced task offloading scheme under the finite resource capacities of edge clouds becomes the most significant challenge. Considering that the tradeoff between bandwidth consumption and reliability has not yet been studied, this article focuses on it.

Containers or VMs consume certain bandwidth resources to transfer the data from the sending tasks while the collaborative tasks are being processed [21]. Since each edge cloud consists of multiple edge servers and base station, and these edge servers often correspond to the base station, which connects with other base stations via fiber optic cables (see Figs. 1 and 2) [22], the bandwidth between collaborative tasks is mainly limited by the bandwidth capacity of the edge servers. That is, as the bandwidth between one pair of collaborative tasks increases, although their communication time decreases, the bandwidth allocated by other tasks on the edge servers is reduced under the condition that the bandwidth of each edge server is fixed, and then the communication time of other collaborative tasks may be increased. Moreover, container or VM failures may prolong the application completion time, including the communication time, the task processing time, the task processing delay, and the recovery time, and then reduce the reliability of IoT applications [14]. Therefore, there is a contradiction between reliability and bandwidth consumption when dealing with a batch of IoT applications, and how to deal with this contradiction well is still unknown.

To simultaneously maximize the reliability and minimize the bandwidth consumption of IoT applications, we propose two approaches [i.e., reliability-enhanced task offloading approach (RETO) and differential evolution (DE) [23] based task offloading approach (DETO)] to optimize task offloading schemes in terms of reliability level and bandwidth consumption. The main idea is to first establish a multiobjective optimization problem based on the bandwidth consumption model and reliability model, and then transform it into a single-objective optimization problem, which is solved by our proposed approaches.

The *key contributions* of our research are as follows.

1) We introduce two novel models: one is to formulate the bandwidth consumption while processing a batch of IoT applications; the other is to formulate the reliability level of task offloading schemes based on the Poisson process.
2) Based on the above models, we first formulate a multiobjective optimization problem to simultaneously maximize the reliability and minimize the bandwidth consumption of IoT applications. Then, we transform it into a single-objective optimization problem. Finally, we propose RETO and DETO to minimize the joint optimization value.
3) We establish a mobile edge computing environment, and conduct simulation experiments to comprehensively evaluate the effectiveness and efficiency of our proposed approaches. When compared with other related approaches, our proposed approaches can obtain better performance.

The remainder of the article is organized as follows. The related work is introduced in Section II; our system model is defined in Section III; the problem formulation is presented based on the above models in Section IV; the technical details of the problem solution are introduced in Section V; the

TABLE I
COMPARISON OF DIFFERENT TASK OFFLOADING SCHEMES

Categories	IoT Application Properties					Architectural Properties		Deployment Engine Properties			
	Replica	Task Number	Task Communication	Application Deadline	Task Deadline	Edge	Edge&Cloud	Strategy	Decision Objectives		
									Reliability	Time	Bandwidth
[24]	×	Single	×	√	×	√	×	Bayesian Network	√	√	×
[25]	×	Single	×	√	×	√	×	Machine Learning Statistical Techniques	√	×	×
[26]	√	Single	×	√	×	×	√	Genetic Algorithm	×	√	×
[27]	√	Single	×	√	×	×	√	Genetic Algorithm	√	√	×
[28]	√	Multiple	√	×	√	√	×	SAA-RS	×	√	×
[29]	×	Multiple	×	√	√	√	×	MAMTS	×	√	×
[30]	√	Single	×	×	×	×	√	SEHPA	×	√	×
[31]	×	Multiple	√	√	×	×	√	GenDoc	×	√	×
[32]	×	Multiple	√	×	×	×	√	APMA	×	√	×
[33]	×	Single	×	×	√	×	√	MBFD	√	×	×
[34]	×	Single	×	√	×	√	×	MESA	×	√	×
[35]	×	Multiple	√	×	×	×	√	DSA	×	×	√
[36]	√	Multiple	√	×	×	×	√	GSP-SS	×	×	√
[37]	×	Multiple	×	×	×	×	√	DDTO	×	√	×
[8]	×	Multiple	√	×	×	√	×	Heuristic Algorithm	×	×	√
[38]	×	Multiple	√	×	√	√	×	Dedas	√	√	×
Our Scheme	×	Multiple	√	√	×	√	×	RETO,DETO	√	√	√

evaluation experiments are conducted in Section VI. Finally, the conclusions along with future work are presented in Section VII.

II. Related Work

Several works have focused on task offloading of IoT applications—key ones are shown in Table I. Aral and Brandic [24] proposed a Bayesian network model based on QoS-related parameters to estimate the availability level of VMs in edge infrastructure, and to avoid the deterioration response time limit that is critical to edge applications. Soualhia *et al.* [25] proposed a framework for detecting and predicting all faults in the edge cloud at the infrastructure level via statistical techniques and supervised machine learning, which can detect and predict some faults online in a timely manner. Maia *et al.* [26] jointly studied the vertical and horizontal load distribution and location of scalable IoT services to minimize potential QoS violations on account of the limitation of edge computing resources. Meanwhile, they investigated how to deploy replicas of applications, and proposed a genetic algorithm that minimizes not only deadlines for response time, but also other potentially conflicting goals, such as operational cost and unavailability [27]. Zhao *et al.* [28] proposed a distributed redundant scheduling algorithm to solve the availability problem of microservice-based applications caused by container failures. Although these studies improved the availability level of the applications in different ways, they did not solve the reliability problem by considering the application completion time. For this purpose, Liu *et al.* [29] introduced an efficient task scheduling algorithm to minimize the average completion time of multiple applications. This algorithm ensured the completion time constraint of applications and processing dependency requirements of tasks by prioritization of multiple applications and tasks. Zhao and Liu [30] investigated in detail the optimal deployment of VM replicas supporting multiple applications to minimize the average response time and total cost for service provision. Goudarzi *et al.* [31] presented a new application allocation approach based on the memetic algorithm to minimize the completion time and power consumption of IoT applications via a weighted cost model. Liu *et al.* [32] introduced a new approximation algorithm to effectively solve the problem of dependent task allocation and scheduling with on-demand function configuration on edge servers, to minimize the application completion time. However, they did not consider the effect of VM failures on the application completion time, or the contradictory relationship between the bandwidth consumption and the application completion time. Yao and Ansari [33] proposed an efficient algorithm to achieve a balance between maximizing the service reliability and minimizing the rental cost of VMs for fog resource provisioning in IoT networks. Yousefi *et al.* [34] transformed the task scheduling problem into a mixed-integer nonlinear optimization problem and proposed a greedy algorithm to minimize the number of SLA violations. However, the above two studies only studied the task scheduling of applications, and did not consider the communication relationship between these tasks and the impact on the application completion time.

Moreover, Hu *et al.* [35] proposed a near-optimal service allocation strategy that meets the constraints of edge server resources and bandwidth to find the tradeoff between average network delay and load balancing. Farhadi *et al.* [36] introduced a two-time-scale framework to jointly optimize service deployment and request scheduling within the constraints of storage, communication, computation, and budget, and then adapt over time to serve time-varying demands under the consideration of system stability and operation cost. Wu *et al.* [37] formulated the hybrid task deployment problem as a multiobjective optimization problem, and then introduced an effective and efficient offloading framework with intelligent decision-making capabilities to jointly minimize the system utility and the bandwidth consumption for each IoT device. However, the above three studies did not consider the effect of different task or service deployment schemes on the application completion time and bandwidth consumption of edge clouds.

Considering the joint optimization of the application completion time and the bandwidth consumption, Zhu and Huang [8] presented a cost model considering interhost network performance and CPU/memory overuse to track the impact of changing the deployment strategy of mobile edge applications on the availability and interhost network bandwidth cost. Meng *et al.* [38] jointly considered the management of network bandwidth and computing resources to satisfy the maximum number of deadlines, and then proposed an online algorithm to greedily schedule newly arrived tasks to meet the new deadlines. However, the above two studies did not study the effects of communication between collaborative tasks on the bandwidth consumption of the edge clouds and the application completion time, and thus on the reliability level.

In an analysis of the available literatures, we have found that no recent studies have focused on the collaborative task offloading of IoT applications under the consideration of container or VM failures and the bandwidth consumption between these collaborative tasks. In this article, we assume that the collaborative tasks of IoT applications are offloaded onto multiple heterogeneous containers or VMs with heterogeneous failure rates and recovery rates. The selection of different containers or VMs results in different processing times and failover times, on the other hand, different edge servers accommodate these containers or VMs, which consume different bandwidth resources, thus resulting in different communication times. Furthermore, the application completion time is greatly affected, and then affect the reliability level of these IoT applications. Therefore, we propose two near-optimal approaches to simultaneously maximize the reliability level and minimize the bandwidth consumption of IoT applications.

III. System Model

We propose a reliability-aware mobile edge computing system model, as shown in Fig. 1. This system includes multiple edge clouds (ECs) to provide IoT services by an IoT system operator. These edge clouds are interconnected through a fiber backhaul network using a full mesh topology [22], [39], as shown in Fig. 2. Each edge cloud owns a certain number of heterogeneous edge servers and is deployed near IoT devices (e.g., smart cars, iPads, wearable devices, Raspberry Pi, smartphones, etc.) [40], [41]. Each edge server accommodates a certain number of heterogeneous containers (Cs) or VMs. Each container or VM has a certain failure rate and recovery rate. Each IoT device produces a latency-sensitive IoT application at some point. Each IoT application consists of multiple collaborative tasks (Ts) and is modeled as a directed acyclic graph, as shown in Fig. 3. There are dependencies between these tasks (e.g., $T_{1,1}$, $T_{1,2}$, and $T_{1,3}$, or $T_{2,1}$, $T_{2,2}$, $T_{2,3}$, $T_{2,4}$, $T_{2,5}$, $T_{2,6}$, $T_{2,7}$) in each IoT application. For instance, $T_{1,3}$ can only be processed when $T_{1,1}$ and $T_{1,2}$ complete and send their results to $T_{1,3}$. The gray tasks (e.g., $T_{1,1}$, $T_{1,2}$, $T_{2,1}$, $T_{2,2}$, and $T_{2,3}$) represent the starting tasks of the

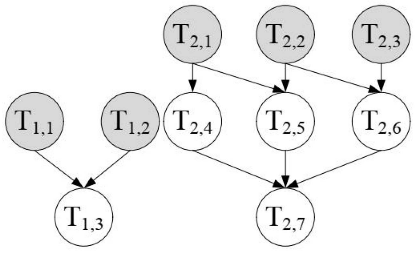

Fig. 3. Example of the directed acyclic graph of IoT applications.

IoT applications, and the arrows represent the dependencies between the tasks. When a container or VM fails (e.g., VM1), the task on it fails to execute, and then cause the corresponding IoT application to fail. Since this article focuses on the task offloading of latency-sensitive IoT applications, all IoT devices can access and completely offload all tasks of these latency-sensitive IoT applications to multiple edge clouds instead of the remote cloud, and hand over to the edge servers for processing. Please note that each container or VM deals with only an indivisible task.

In Fig. 1, P heterogeneous edge servers are randomly pre-deployed to H edge clouds of the reliability-aware mobile edge computing system. Next, M denotes the number of heterogonous containers or VMs, which are indexed by the set $Z = \{1, 2, \ldots, M\}$ and randomly assigned to these edge servers. Since containers or VMs have similar configuration information, and this article focuses on their tasks, how to distinguish between containers and VMs is our future research content. Container or VM j has a certain processing capacity c_j in units of instructions per second, bandwidth bw_j in units of Mb per second, memory capacity L_j in units of the number of instructions, failure rate λ_j, and recovery rate u_j. The IoT devices connect to this system and generate latency-sensitive IoT applications in arbitrary order and time. N IoT applications indexed by the set $I = \{1, 2, \ldots, N\}$ are generated at some point. The IoT application i has deadline requirement D_i in units of seconds and is split into $|Q_i|$ tasks, and task $q \in Q_i$ has a certain instruction length l_{iq}. When these tasks are offloaded to these edge clouds, the state of all containers or VMs will be analyzed to determine which tasks should be assigned to which containers or VMs. Meanwhile, the containers or VMs that are not assigned tasks can be shut down. As containers or VMs with higher failure rates may fail during task processing, and containers or VMs in the same cluster may be deployed in different edge servers, these reasons both affect the application completion time, bandwidth consumption, and reliability level of IoT applications. Therefore, it is critical to design a reliability-enhanced task offloading scheme for these IoT applications. For ease of reference, we show the key notations in this article, as shown in Table II.

A. Transmission Delay Model

The IoT devices and edge clouds exploit orthogonal frequency division multiple access to realize the wireless communication between them [42]. The transmission delay of

TABLE II
KEY NOTATIONS

Notation	Description
P	the number of edge servers
H	the number of edge clouds
M	the number of Cs or VMs
Z	the set of Cs or VMs
c_j	the processing capacity of C or VM j
bw_j	the bandwidth of C or VM j
L_j	the memory capacity of C or VM j
λ_j	the failure rate of C or VM j
u_j	the recovery rate of C or VM j
N	the number of IoT applications
I	the set of IoT applications
D_i	the deadline requirement of IoT application i
Q_i	the set of all tasks of IoT application i
l_{iq}	the instruction length of the task $q \in Q_i$
t_i^{send}	the transmission delay of input data d_i
C_i	the total bandwidth resources consumed by the edge servers after processing IoT application i
d_{iqj}	the task processing delay
T_{iqj}	the average total processing time of the *q-th* task of IoT application i on the *j-th* container or virtual machine
W	the set of execution routes from the top-level tasks to the bottom-level tasks
T_i	the completion time of IoT application i
U	the total number of IoT applications with the QoS violations
R_{iqj}	the execution route with the longest execution time for container or virtual machine j that can accommodate task q
ω_q	the minimum value of all $R_{iqj}, j \in \Omega$
Ω	the set of containers or virtual machines accommodating the offloading task q
CR_n	the crossover probability of the individual x_n
F_n	the scaling factor of the individual x_n

input data d_i of IoT application i is denoted by t_i^{send}. The transmission power of IoT device i is denoted by p_i. The channel gain between IoT device i and edge cloud h is denoted by g_i^h. The data transmission rate of the link between IoT device i and edge cloud h is denoted by s_i^h which can be computed in the following:

$$s_i^h = W \log\left(1 + \frac{p_i g_i^h}{N_0}\right) \tag{1}$$

where W and N_0 represent the bandwidth of the link between IoT device i and edge cloud h and the noise power, respectively. Thus, the transmission delay (i.e., t_i^{send}) of IoT device i for offloading its IoT application i with data size d_i can be computed in the following:

$$t_i^{\text{send}} = \frac{d_i}{s_i^h}. \tag{2}$$

B. Bandwidth Consumption Model

When IoT applications are generated by IoT devices, each IoT application will be split into multiple collaborative tasks. All or part of these tasks will be offloaded to the edge clouds, and delivered multiple containers or VMs for processing. Considering the communication between the containers or VMs in the same cluster, the bandwidth resources and time required by the cluster to process an IoT application are directly related to the location of the containers or VMs in the edge clouds. This outcome is true because if the containers or VMs in the same cluster are on the same edge server, the communication between the containers or VMs does not consume bandwidth resources and time; conversely, it consumes a certain amount of bandwidth resources and time. Therefore, the total bandwidth resources consumed by the edge servers after processing IoT application i can be expressed in the following:

$$C_i = \sum_{q=1}^{|Q_i|} \sum_{h=1}^{H} \sum_{p=1}^{|P_h|} \sum_{m=1}^{|M_p|} \left(x_{iqm} \cdot \sum_{r \in Z, r \notin M_p} b_{iqmr} \cdot bw_m \right) \tag{3}$$

where P_h denotes the set of the edge servers owned by the hth edge cloud; M_p denotes the set of the containers or VMs on the pth edge server; the binary variable x_{iqm} denotes whether the qth task is deployed to the mth container or VM, $x_{iqm} = 1$ if affirmative, otherwise $x_{iqm} = 0$; the binary variable b_{iqmr} denotes whether the task q being processed by the mth container or VM is a sender and the same cluster with the other task being processed by the rth container or VM, $b_{iqmr} = 1$ if affirmative, otherwise $b_{iqmr} = 0$. Note that, the rth container or VM is not on the same edge server as the mth container or VM. bw_m denotes the bandwidth from the mth container or VM to the rth container or VM, which is specified as a random value in a certain range.

C. Reliability Model

Not only are there multiple failures in wireless channels, edge servers, mobile devices, and the links connecting edge servers of mobile edge computing systems, but also there are also containers or VMs that may fail when a container or VM is processing a task. Therefore, we mainly introduce the impact of container or VM failures on the reliability of such systems in this article. To better study the impact of container or VM failures on the reliability level of IoT applications, these failures are treated as recoverable [9]. That is, the failed container or VM resumes the task processing after a period of time (i.e., recovery time). Considering that the virtualization technology has an isolation nature, all container or VM failures are considered to be independent of each other [43]. Moreover, the processing time taken by container or VM j to complete task q of IoT application i without failure is $t_{iqj} = l_{iq}/c_j$. During the time interval $(0,\ t_{iqj}]$, failures occur and are assumed to be a Poisson process with failure rate parameter λ_j [9], and the total number of failures in the container or VM j is denoted by $N_j(t_{iqj})$.Therefore, the probability of $N_j(t_{iqj}) = k$ can be computed by formulation (4) during the time interval $(0,\ t_{iqj}]$

$$\Pr\{N_j(t_{iqj}) = k\} = \frac{(\lambda_j t_{iqj})^k}{k!} e^{-\lambda_j t_{iqj}}, k \geq 0. \tag{4}$$

The mean of $N_j(t_{iqj})$ can be computed in the following:

$$E[N_j(t_{iqj})] = \lambda_j t_{iqj}. \tag{5}$$

The recovery time $R_{jk}(t_{iqj})$ of the kth failure at container or VM j is assumed to be an exponentially distributed random variable with recovery rate parameter u_j during the time interval $(0,\ t_{iqj}]$ [9], as shown in formulation (6). Thus, the total recovery time $R_j(t_{iqj})$ at the jth container or VM can be computed by formulation (7) during the time interval $(0,\ t_{iqj}]$.

As the total recovery time $R_j(t_{iqj})$ is a compound Poisson process, its mean can be computed by formulation (8) during the time interval $(0,\ t_{iqj}]$

$$R_{jk}\left(t_{iqj}\right) = u_j \cdot e^{-u_j \cdot t_{iqj}} \tag{6}$$

$$R_j\left(t_{iqj}\right) = \sum_{k=1}^{N_j\left(t_{iqj}\right)} R_{jk}\left(t_{iqj}\right) \tag{7}$$

$$E\left[R_j\left(t_{iqj}\right)\right] = E\left[N_j\left(t_{iqj}\right)\right] \cdot E\left[R_{jk}\left(t_{iqj}\right)\right] = \frac{\lambda_j t_{iqj}}{u_j}. \tag{8}$$

The average total processing time T_{iqj} of the qth task of IoT application i on the jth container or VM consists of the task processing time t_{iqj}, the task processing delay d_{iqj}, and failure recovery time $E[R_j(t_{iqj})]$, and its value can be computed in the following:

$$T_{iqj} = t_{iqj} + d_{iqj} + \frac{\lambda_j t_{iqj}}{u_j} \quad \forall i \in I, j \in Z, q \in Q_i. \tag{9}$$

Moreover, if part or all of the containers or VMs in the same cluster are not on the same edge server, these containers or VMs with communication relationships need to take time to transfer data. Therefore, the completion time (i.e., T_i) of IoT application i consists of the task processing time, the task processing delay, the failure recovery time, and the task communication time. Meanwhile, the directed acyclic graph of IoT application i consists of multiple execution routes (i.e., W). These routes not only have different execution times, but also the execution route with the longest execution time determines the completion time of the application, as shown in formulation (10). In view of the small data size of the IoT application result, the time it takes to transmit the data from the edge cloud back to the IoT device is negligible

$$T_i = t_i^{\text{send}} + \max_{v \in W}\left(\sum_{q=1}^{|v|}\left(T_{iqj} + \sum_{z=1}^{|v|}\left(\beta_{iqz} \cdot \rho_{iqz} \cdot \frac{\text{data}_{iq}}{bw_j}\right)\right)\right)$$
$$j \in Z \tag{10}$$

where W denotes the set of execution routes, such as $\{T_{2,1}, T_{2,4}, T_{2,7}\}$, $\{T_{2,1}, T_{2,5}, T_{2,7}\}$, $\{T_{2,2}, T_{2,5}, T_{2,7}\}$, $\{T_{2,2}, T_{2,6}, T_{2,7}\}$, and $\{T_{2,3}, T_{2,6}, T_{2,7}\}$ from the top-level tasks (e.g., $T_{2,1}$, $T_{2,2}$, $T_{2,3}$) to the bottom-level tasks (e.g., $T_{2,7}$); v denotes the set of tasks for an execution route (e.g., $\{T_{2,1}$, $T_{2,4}$, and $T_{2,7}\}$) of the set W; the binary variable β_{iqz} denotes whether the task q and the task z of IoT application i are on the same edge server, $\beta_{iqz} = 0$ if affirmative, otherwise $\beta_{iqz} = 1$; the binary variable ρ_{iqz} denotes whether the task q sends data to the task z, $\rho_{iqz} = 1$ if affirmative, otherwise $\rho_{iqz} = 0$; data_{iq} denotes the amount of data sent by the task q; bw_j denotes the bandwidth of the container or VM j where task q resides.

In the process of the IoT system operator providing services to mobile users, the reliability level reflects how the system operates and hence how successfully a requested service can be provided. This article mainly studies how the failures and recoveries of the containers or VMs affect the total processing time of IoT applications (i.e., QoS). If the total processing time of IoT application i exceeds its deadline requirement D_i, QoS is considered to have been violated. Therefore, the probability of QoS violations can be characterized by the ratio of the number of unsatisfied IoT applications to the total number of IoT applications during a given time [14]. Furthermore, the reliability level of the IoT applications can be computed by the probability of QoS violations, as shown in formulation (11). U represents the total number of IoT applications with QoS violations and can be computed by the following:

$$\text{Reliability} = 1 - \frac{U}{N} \tag{11}$$

$$U = \sum_{i=1}^{N} F(T_i - D_i) \tag{12}$$

$$F(T_i - D_i) = \begin{cases} 1, & T_i - D_i > 0 \\ 0, & T_i - D_i \le 0 \end{cases} \tag{13}$$

where the difference value $(T_i - D_i)$ represents whether the QoS requirement of IoT application i is violated, $F(T_i - D_i) = 1$ if $(T_i - D_i) > 0$, otherwise $F(T_i - D_i) = 0$.

IV. Problem Formulation

In our mobile edge computing system, the edge clouds are interconnected via fiber optic cables, and the communication between the edge clouds is considered to be load independent. Therefore, if the containers or VMs handling the collaborative tasks of IoT application i are not deployed in the same edge servers, the communication time between these containers or VMs depends mostly on the bandwidth and data size of the containers or VMs in the sending state. Meanwhile, as containers or VMs have certain failure rates and recovery rates, the processing time of IoT applications is further increased, and then the reliability level of IoT applications is also greatly affected. Although the communication latency can be reduced by increasing the communication bandwidth between tasks, the edge servers have fixed bandwidth. That is, if the bandwidth between the collaborative tasks of an IoT application is increased, the bandwidth of other sending tasks on the same edge server is reduced, and then the reliability level of the IoT applications in which other tasks reside is also reduced. Therefore, we need to find a tradeoff to minimize the bandwidth consumption of N IoT applications while maximizing the reliability level of these IoT applications during task offloading under the finite resource capacities of edge clouds, which can be denoted as follows:

$$F0 : \begin{cases} \min, & \sum_{i=1}^{N} C_i \\ \max, & 1 - \frac{1}{N}\sum_{i=1}^{N} F(T_i - D_i) \end{cases} \tag{14}$$

s.t.

$$\sum_{r \in Z} x_{iqr} = 1, q \in Q_i, i \in I \tag{15}$$

$$\sum_{i=1}^{N}\sum_{q=1}^{|Q_i|} l_{iq} x_{iqr} \le L_r, r \in Z \tag{16}$$

where formulation (15) denotes that each task can only be deployed to one container or VM; since the memory capacity of the container or VM is in units of the number of instructions, formulation (16) is exploited to represent that the memory capacity of each container or VM is greater than or equal to the sum of the memory required for the deployed tasks.

Since $F(\cdot)$ is the unit-step function, the optimization problem $F0$ is not only nonlinear, but also a variant of the typical bin-packing problem with NP-hardness [44]. Therefore, a Boolean variable B_i is defined and assigned a value of $F(T_i - D_i)$ $\forall i \in I$.$B_i = F(T_i - D_i)$ can also be denoted as formulation (17) [33]. $\Re$ denotes a very large positive number. Furthermore, formulation (17) can be transformed into formulation (18) by adding a small positive number ε, which is regarded as the tolerance of the QoS violation in units of seconds. That is, the determination of QoS violation is a transition process and not a momentary process

$$-\Re \cdot (1 - B_i) < T_i - D_i \le \Re \cdot B_i \quad \forall i \in I \tag{17}$$

$$-\Re \cdot (1 - B_i) + \varepsilon \le T_i - D_i \le \Re \cdot B_i \quad \forall i \in I. \tag{18}$$

Then, the optimization problem $F0$ can be transformed into another form of multiobjective optimization problem, as shown in the formulation (19). Please note that the formulation (14) is nonlinear because it includes the unit-step function $F(T_i - D_i)$ $\forall i \in I$ and then must be converted into the linear formulation (19) by the Boolean variable B_i

$$F1: \begin{cases} \min, & \sum_{i=1}^{N} C_i \\ \max, & 1 - \frac{1}{N}\sum_{i=1}^{N} B_i \end{cases} \tag{19}$$

$$\text{s.t.}\ (15), (16), (18)$$

$$B_i \in \{0, 1\} \quad \forall i \in I. \tag{20}$$

However, as the optimization problem $F1$ is still the multiobjective optimization problem, a widely used weighted sum is exploited to convert it into a single-objective optimization problem by adding weights into the objectives, which denote the relative importance of the single objective [45]. The single-objective optimization problem $F2$ can be denoted as follow:

$$F2: \min\ \theta_1 \cdot \sum_{i=1}^{N} C_i + \frac{\theta_2}{N}\sum_{i=1}^{N} B_i - \theta_2$$

$$\text{s.t.}\ (15), (16), (18), (20) \tag{21}$$

where θ_1 and θ_2 are both positive tunable factors ($\theta_1 + \theta_2 = 1$).

Considering that the single-objective optimization problem $F2$ is still an NP-hard problem, it is an urgent need to adopt an approach (e.g., RETO or DETO) to obtain a near-optimal solution of the problem.

V. Problem Solution

In this section, we propose two near-optimal approaches (i.e., RETO and DETO) to solve the above single-objective optimization problem $F2$ and build a mobile edge computing system (Fig. 1) to verify the effectiveness and efficiency of these two approaches. In this system, all containers or VMs are randomly assigned to the edge servers, and each IoT application is split into multiple collaborative tasks (see Fig. 3) and offloaded to these containers or VMs via our proposed approaches. Since the single-objective optimization problem $F2$ is an NP-hard problem, we exploit RETO and DETO to obtain near-optimal solutions. When considering lower complexity, we consider using RETO to obtain the acceptable solutions; when further approximating the optimal solution, we can consider using DETO to obtain the acceptable solutions. In the next section, we introduce the implementation schemes of these two approaches.

A. RETO Approach

When these collaborative tasks of IoT applications are offloaded and assigned to containers or VMs, there can be two kinds of task offloading schemes according to the multiobjective optimization problem $F0$ and the directed acyclic graph structure of the IoT application (see Fig. 3). One way is that these tasks of the directed acyclic graph are offloaded from the bottom to the top in turn. Another way is that these tasks of the directed acyclic graph are offloaded from the top to the bottom in turn. Through experimental comparison, the effects of the two offloading schemes are not significantly different, so the second offloading approach (i.e., RETO) will be introduced next.

The RETO approach will need to exploit five phases to obtain the near-optimal offloading scheme. First, we initialize all parameters and conduct an IoT application priority queue such as $\{T_1, T_2\}$ according to the ascending order of the IoT application deadline. Second, we determine the task hierarchy for offloading in a top-down order of the directed acyclic graph, i.e., in addition to the top-level tasks (e.g., $T_{1,1}$, $T_{1,2}$, $T_{2,1}$, $T_{2,2}$, and $T_{2,3}$) being the sending tasks and the bottom-level tasks (e.g., $T_{1,3}$ and $T_{2,7}$) being the receiving tasks, the tasks in the middle level (e.g., $T_{2,4}$, $T_{2,5}$, and $T_{2,6}$) can both send and receive results at some point. Third, the top-level tasks are sorted in descending order of instruction length and select the containers or VMs to be deployed according to the formulation $\min_{j\in\Omega}(T_{iqj} + \alpha \cdot bw_j)$, $i \in I$.

After the above steps are completed, both the top-level tasks of the directed acyclic graph are offloaded onto the edge servers. Next, other tasks of the directed acyclic graph are offloaded to the edge servers. Once the tasks of the previous level are completed, these tasks are then seen as sending tasks and sending their results to the lower-level tasks. Fourth, the tasks of the second level are offloaded as receiving tasks in descending order of their instruction length. For the second-level task q of IoT application i, its serving container or VM first selects a container or VM on the same edge server as the containers or VMs where the sending tasks (e.g., $T_{2,1}$, $T_{2,2}$, and $T_{2,3}$) reside. If no such containers or VMs exist, R_{iqj} is first obtained by formulation (23), which represents the execution routes with the longest execution time for container or VM j that can accommodate task q; the most appropriate container or VM for task q is second selected via formulation (22), which can obtain the minimum value ω_q

$$\omega_q = \min_{j\in\Omega}\left(R_{iqj}\right) \tag{22}$$

s.t.

$$R_{iqj} = \max\left(T_{iqj} + \frac{\text{data}_{q'}}{bw_{j'}} + \alpha \cdot bw_j\right), i \in I, q \in Q_i, j \in \Omega \tag{23}$$

Algorithm 1 Reliability-Enhanced Task Offloading Approach

```
Input: $N$, $H$, $M$, $P$, $\alpha$, $\theta_1$, $\theta_2$,directed acyclic graphs of IoT applications.
Output: The near-optimal solution.
1: Initialize all parameters
2: Sort $N$ IoT applications in ascending order via their deadline
3: for $i$ =1 to $N$ do
4:     Obtain the number of structure levels of the IoT application $i$ $A_i$
5:     Divide $Q_i$ into $A_i$ groups, $G_l, l \in \{1, 2, ..., A_i\}$
6:     for $l$=1 to $A_i$ do
7:         Sort $G_l$ in descending order via instruction length
8:         for $q$=1 to $|G_l|$ do
9:             if the q-th task of $G_l$ is the sending task then
10:                Find the C or VM via
                   $\min_{j\in\Omega} (T_{iqj} + \alpha \cdot bw_j), i \in I$
11:            end if
12:            if the q-th task of $G_l$ is the receiving task then
13:                Find the C or VM via formulation (22)
14:            end if
15:        end for
16:    end for
17: end for
18: for $i$ =1 to $N$ do
19:     Obtain the set of execution routes $W$
20:     for $\gamma$=1 to $|W|$ do
21:         Obtain the $T_i$ via formulation (10)
22:     end for
23:     Compute $\sum_{i=1}^{N} T_i$
24:     Compute the reliability level via formulation (11)
25:     Compute $C_i$ via formulation (3)
26: end for
27: return the near-optimal solution
```

where α is a small positive tunable factor; Ω represents a set of containers or VMs accommodating the offloading task q; $bw_{j'}$ represents the bandwidth of the container or VM j', which accommodates the sending task q' of task q; $\text{data}_{q'}$ represents the amount of data sent by task q' to task q. That is, the total processing time T_{iqj} is computed according to the receiving tasks of each task of the previous level, the transmission delay is computed according to the task of the previous level.

Finally, we sort the receiving tasks of each task of the subsequent level in descending order of instruction length and offload these receiving tasks to the edge servers according to the processing method in the fourth step.

Based on the above, these collaborative tasks of N IoT applications are offloaded onto the edge servers while simultaneously enhancing the reliability level of these IoT applications and reducing the bandwidth consumption of these IoT applications. The implementation scheme of the RETO is presented in Algorithm 1. Line 2 sorts N IoT applications in ascending order via their deadlines to first offload the time-critical IoT applications. The loop in lines 3–17 obtains the near-optimal offloading scheme of all tasks of N IoT applications. Lines 4 and 5 obtain the number of structure levels A_i of each IoT application, and divide the task set Q_i into A_i groups $G_l, l \in \{1, 2, \ldots, A_i\}$. The loop in lines 6–16 selects the container or VM j for the qth task of the set G_l. Line 7 sorts tasks of the set G_l in descending order via the instruction length. Lines 9–11 select the containers or VMs for the sending tasks of the set G_l according to the formulation, i.e., $\min_{j\in\Omega} (T_{iqj} + \alpha \cdot bw_j), i \in I$. Lines 12–14 select the containers or VMs for the receiving tasks of the set $G_l, l \in \{1, 2, \ldots, A_i\}$ according to formulation (22). The loop in lines 18–26 obtains the reliability level, bandwidth consumption, and total completion time of N IoT applications.

The complexity of Algorithm 1 is mainly determined by two parts, i.e., lines 3–17 and lines 18–26. As previously known, the number of IoT applications is set to N. Therefore, the first part has the time complexity $O(N * A_i * |G_l|)$ to obtain the near-optimal task allocation in lines 3–17. That is, the time complexity of this part increases with the number of IoT applications, levels, and tasks per level. The second part has the time complexity $O(N * |W|)$ to compute the processing results in lines 18–26. That is, the time complexity of this part increases with the number of IoT applications and execution routes. Finally, the total time complexity of Algorithm 1 can be expressed as $O(N * A_i * |G_l|)$.

B. DETO Approach

To solve Chebyshev polynomials, a heuristic random search algorithm based on population differences was proposed by Storn and Price in 1997 [23]. Different from other evolutionary algorithms, the DE algorithm modifies the internal representation of an individual by an arithmetic operator, and then obtains a difference. Next, if the fitness of the newly generated difference vector is better than the current vector by the evaluation, it will be exploited to replace the current vector.

Presently, the DE algorithm has produced a variety of optimization strategies based on the number of disturbed individuals and the weighted different vectors [46]. To maintain the diversity of the population, we exploit the DE/rand/1/bin strategy to select the disturbed vectors. Therefore, three operators of variation, crossover, and selection can be defined as described below [23]. For G iterations, a population consists of an NP D-dimensional parameter vector $x_{n,G} = (v_{1n,G}, v_{2n,G}, \ldots, v_{Dn,G})$, $n = 1, 2, \ldots, NP$ in which each D-dimensional parameter vector denotes an individual with D optimization parameters.

1) Variation Operator: This operator will generate a variation vector $v_{n,G+1} = (v_{1n,G+1}, v_{2n,G+1}, \ldots, v_{Dn,G+1})$ for each target vector $x_{n,G}$

$$v_{n,G+1} = x_{r1,G} + F \cdot \left(x_{r2,G} - x_{r3,G}\right) \tag{24}$$

where four randomly selected individuals are different from each other, i.e., $r1 \neq r2 \neq r3 \neq n$, and the number of parameter vectors in the cluster must be more than or equal to four; a predefined scaling factor $F \in [0, 1)$ is exploited to adjust the scaling of the difference vector and control the search step size.

2) Crossover Operator: This operator will generate a trial vector $u_{n,G+1} = (u_{1n,G+1}, u_{2n,G+1}, \ldots, u_{Dn,G+1})$ by discrete crossover between the target vector and the variation vector, as shown in the following:

$$u_{gn,G+1} = \begin{cases} v_{dn,G+1}, & \text{if } (r \le CR) \text{ or } d = d_{\text{rand}} \\ x_{dn,G}, & \text{if } (r > CR) \text{ or } d \ne d_{\text{rand}} \end{cases} \tag{25}$$

where r represents the random value in the range [0, 1]; $CR \in [0, 1]$ represents the predefined crossover probability; $CR = 0$ represents that the corresponding vector has no crossover; d can be a natural number between 1 and D; d_{rand} is a given random index from the range $[1, D]$ to ensure that $u_{n,G+1}$ can inherit at least a 1-D element from the variation vector $v_{n,G+1}$.

3) Selection Operator: This operator greedily compares the fitness $f(*)$ of the children and the corresponding parent, and the better one is kept for the $G + 1$ generation, as shown in the following:

$$x_{n,G+1} = \begin{cases} u_{n,G+1}, & \text{if } \big(f(u_{n,G+1}) \le f(x_{n,G})\big) \\ x_{n,G}, & \text{otherwise.} \end{cases} \tag{26}$$

To better solve the above joint optimization problem, the DE algorithm with the above three operators needs to be improved by adjusting the population size NP, the crossover probability CR, and the scaling factor F. Since three strategies (i.e., constant, random, and adaptive) can be exploited to adjust the controlling parameters F and CR, these strategies can have a large impact on the diversity, convergence, and search space of the algorithm [47]. Therefore, to prevent failure to achieve local optima and a low convergence rate, adaptive strategies for controlling parameters CR and F are exploited to improve the convergence rate and diversity of the DE algorithm and better solve the joint optimization problem.

The value of F is positively correlated with the variation search range of the DE algorithm. The value of F decreases with the execution of the algorithm to ensure the population diversity and the protection of the optimal solution. In the iterative process of the algorithm, the value of F_n of the individual x_n will be adjusted adaptively according to formulation (27). Meanwhile, since the higher crossover probability increases the probability that individuals with lower fitness enter the next generation, the small crossover probability CR can enhance the global search ability and population diversity. In the iterative process of the algorithm, the value of CR_n of the individual x_n will be adjusted adaptively according to formulation (28)

$$F_n = F_0 \cdot 5^{\lambda} \text{ and } \lambda = \mathrm{e}^{1-\frac{f_{\max}-f_{\min}}{f_n-f_{\min}}} \tag{27}$$

$$CR_n = \begin{cases} CR_{\min} + (CR_{\max} - CR_{\min}) \cdot \frac{f_n-f_{\min}}{f_{\max}-f_{\min}}, & \text{if } f_n < \bar{f} \\ CR_{\min}, \ \text{if } f_n \ge \bar{f} \end{cases} \tag{28}$$

where $f_{\min}$ denotes the fitness of the optimal individual in the current iteration population; $f_{\max}$ denotes the fitness of the worst individual in the current iteration population; $\bar{f}$ and f_n represent the average fitness of the current population and the individual x_n, respectively; $CR_{\max}$ and $CR_{\min}$ denote the maximum and minimum crossover probability, respectively; and F_0 denotes the initial scaling factor.

During the algorithm iteration, the crossover operator can also generate another trial vector $w_{n,G+1} = (w_{1n,G+1}, w_{2n,G+1}, \ldots, w_{Dn,G+1})$ to retain genetic information according to formulation (29). Finally, the selection operator can select the optimal individual from the variation vector $v_{n,G+1}$, the trial vector $w_{n,G+1}$, and the trial vector $u_{n,G+1}$ to further retain the genetic information of each iteration according to formulation (30)

$$w_{gn,G+1} = \begin{cases} x_{dn,G+1}, & \text{if } (r \le CR) \text{ or } d = d_{\text{rand}} \\ v_{dn,G}, & \text{if } (r > CR) \text{ or } d \ne d_{\text{rand}} \end{cases} \tag{29}$$

$$x_{n,G+1} = \begin{cases} u_{n,G+1}, & \text{if } \big(f(u_{n,G+1}) \le f(x_{n,G})\big) \\ v_{n,G+1}, & \text{else if } \big(f(v_{n,G+1}) \le f(x_{n,G})\big) \\ w_{n,G+1}, & \text{else if } \big(f(w_{n,G+1}) \le f(x_{n,G})\big) \\ x_{n,G}, & \text{otherwise.} \end{cases} \tag{30}$$

In the improved scheme of the DE algorithm, the total number of all tasks in N IoT applications is taken as the chromosome length. Each task is taken as a gene fragment, and the genetic information of each gene fragment is the VM index number. Furthermore, the DETO approach can be introduced by applying the above-improved measures to the standard DE algorithm, and solving the joint optimization problem via formulation (21). The implementation details of the DETO approach are shown in Algorithm 2.

As shown in Algorithm 2, the system parameters (e.g., the number of edge clouds H, the number of edge servers P, the number of containers or VMs M, the initial scaling factor F_0, the minimum crossover probability $CR_{\min}$, the maximum crossover probability $CR_{\max}$, the population size NP, the maximum iteration times $\max G$, the chromosome length D, etc.) are first initialized in line 1. The initial population is randomly produced in line 2. The variation operation is implemented in lines 4–15, in which three different individuals are first selected from the population and sorted from small to large by their fitness. Then, the scaling factor of the current individual is calculated by formulation (27), and finally the variation vector $v_{n,G+1}$ is obtained by formulation (24). The crossover operation is executed in lines 16–23, in which the crossover probability CR_n is computed by formulation (28) and the two trial vectors $u_{n,G+1}$ and $w_{n,G+1}$ are generated by formulations (25) and (29). The selection operation is executed in lines 24–28, in which the optimal individual is selected from the variation vector $v_{n,G+1}$, the current individual vector $x_{n,G}$, the trial vector $u_{n,G+1}$ and $w_{n,G+1}$, and then preserved for the next iteration.

The complexity of Algorithm 2 is mainly determined by three operations, i.e., variation operation, crossover operation, and selection operation. As previously known, the population size and the parameter dimension of the individual vector are set to NP and D, respectively. Therefore, the variation operation in lines 4–15 has the time complexity $O(NP * D)$ to traverse all individuals and parameter vectors of the population; the crossover operation in lines 16–23 has the time complexity $O(NP * D)$ to realize the crossover between individuals of the population; and the selection operation in lines 24–28 has the time complexity $O(NP)$ to select the best individual. Finally, the total time complexity of the algorithm can be expressed as $O(\max G * NP * D)$ after $\max G$ iterations.

Algorithm 2 DE Based Task Offloading Approach

```
Input: The system parameters.
Output: The near-optimal solution.
1: Initialize the system parameters
2: Randomly generate $NP$ $D$-dimensional parameter vectors
3: for $G$=1 to $\max G$ do
4:     for $n$=1 to $NP$ do
5:         Compute the fitness of the worst individual and the
6:         optimal individual,the average fitness and the fitness
7:         of individual $x_n$
8:         Select three different individuals $x_{r1,G}$, $x_{r2,G}$,
9:         $x_{r3,G}$ from $x_{n,G}$
10:        Compute the scaling factor of current individual
11:        by formulation (27)
12:        for $d$=1 to $D$ do
13:            Generate variation vector via formulation (24)
14:        end for
15:    end for
16:    for $n$=1 to $NP$ do
17:        Compute crossover probability of current individual
18:        via formulation (28)
19:        for d=1 to $D$ do
20:            Generate trial vectors $u_{n,G+1}$,$w_{n,G+1}$
21:            via formulations (25) and (29)
22:        end for
23:    end for
24:    for $n$=1 to $NP$ do
25:        Select the optimal individual from the variation
26:        vector $v_{n,G+1}$, trial vector$w_{n,G+1}$, trial vector
27:        $u_{n,G+1}$ by formulation (30)
28:    end for
29: end for
30: return the near-optimal solution
```

VI. Performance Evaluation

Using the extended CloudSim simulator [48] and iFogSim simulator [49], we created a reliability-aware mobile edge computing simulation environment consisting of 50 edge clouds, 252 edge servers, and 400 VMs in a machine with Intel Core i7-7500U@2.70 GHz and 8-GB memory. These edge clouds are uniformly distributed in a 5G smart city network scenario and interconnected with a full mesh topology [22]. In each edge cloud, a base station is connected to other base stations via fiber backhaul network; the number of edge servers interconnected via a switch is a randomly selected value from the set [4, 6]; multiple IoT devices exploit wireless access networks to communicate with the base station; these IoT devices generate some IoT applications with different deadlines. The configuration information for each edge server can be a randomly selected value from the set {HP ProLiant G4, HP ProLiant G5}, i.e., HP ProLiant G4 owns 3720 MIPS, 10 GB of memory, 10 GB/s of bandwidth, and 1 TB of external storage; HP ProLiant G5 owns 5320 MIPS, 10 GB of memory, 10 GB/s of bandwidth, and 1 TB of external storage [50]. The processing capacity and memory capacity for each VM can be a randomly selected value from the set {500 MIPS and 0.6 GB, 1000 MIPS and 1.7 GB, 2000 MIPS and 3.75 GB, 2500 MIPS and 0.85 GB} [50]. The disk capacity and bandwidth requirement for each VM are 1 GB and a randomly selected value from the set [10, 50] Mb/s, respectively. The initial population size *NP* and the chromosome length D were set to 20 and 350, respectively. The values of $\max G$, $CR_{\max}$, $CR_{\min}$, and F_0 are set to 100, 0.9, 0.1, and 0.2, respectively. The positive tunable factors α, θ_1, and θ_2 are set to 0.0008, 0.00005, and 0.99995, respectively. For ease of reference, we show the experimental parameters in this article, as shown in Table III.

TABLE III
Experimental Parameters

Parameters	Value
NP	20
D	350
$\max G$	100
$CR_{\max}$	0.9
$CR_{\min}$	0.1
F_0	0.2
α	0.0008
θ_1	0.00005
θ_2	0.99995

The foregoing parameter information is primarily used to configure the environment in which tasks can be offloaded. Next, we will set some parameters of the VMs for task offloading according to the size of the VMs. The processing capacity of each VM is a randomly selected value from the set $[0.5 \times 10^6, 10^6]$ instructions per second [51]; the memory capacity of each VM is a randomly selected value from the set $[2 \times 10^7, 4 \times 10^7]$ instructions; the failure rate of each VM is a randomly selected value from the set [3.5%, 5%]; the recovery rate of each VM is a randomly selected value from the set [1%, 3.5%]. When the IoT devices connect to the edge cloud and generate 50 latency-sensitive IoT applications in arbitrary order and time, each IoT application is modeled as the three-level directed acyclic graph with seven tasks and completely offloaded to the edge clouds.

The input data size of each IoT application is a randomly selected value from the interval [100, 300] kB. The deadline requirement of each IoT application is a randomly selected value from the interval [60, 90] ms. The transmission power of each IoT device is set to 100 mW. The channel bandwidth between IoT devices and edge clouds is set to 5 MHz. The channel gain between IoT devices and edge clouds is set to 20^{-4}. The noise power of the system is set to 10^{-10} mW. The length of each task is a randomly selected value from the set [500, 5000] instructions [33]. If one task is a sender, the amount of data it sends is a randomly selected value from the set [1, 2] MB. The task processing delay of each task in the IoT application is a randomly selected value from the interval [2, 6] ms.

According to the above configuration information, the performance of RETO and DETO is evaluated by comparing it with the following baseline approaches.

1) *Random Offloading (RO):* Randomly select one VM to accommodate each task of the IoT application from top

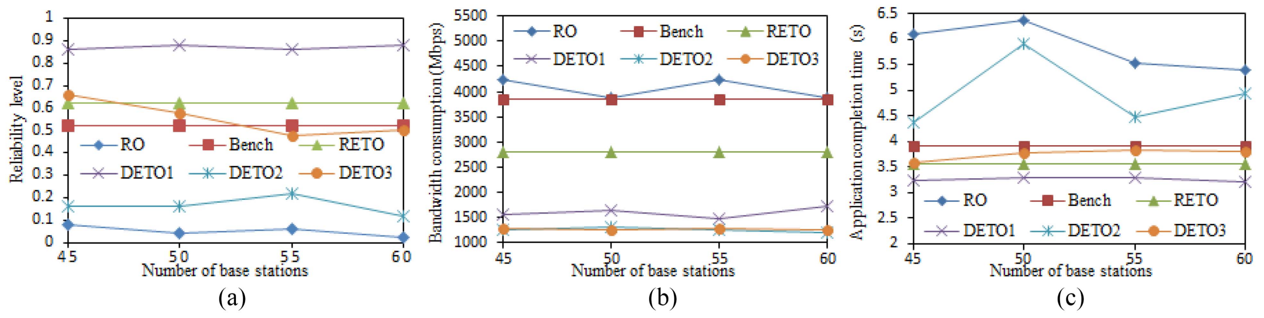

Fig. 4. Impact of the number of base stations. (a) Impact on reliability level. (b) Impact on bandwidth consumption. (c) Impact on application completion time.

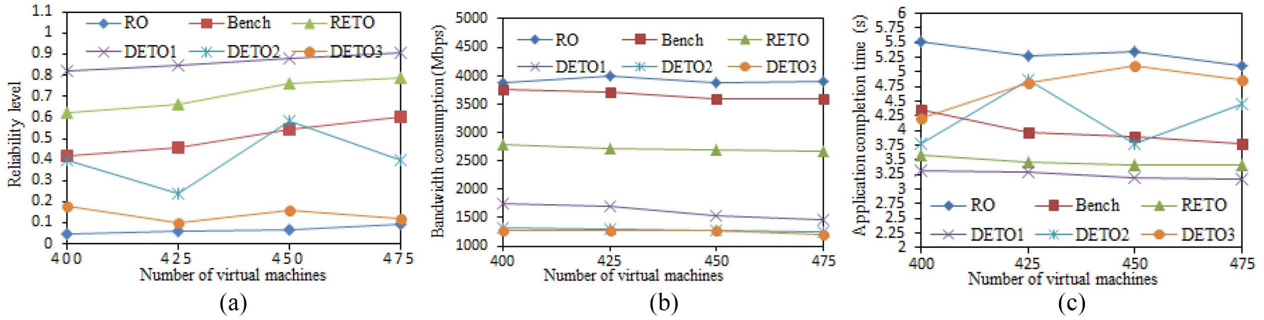

Fig. 5. Impact of the number of VMs. (a) Impact on reliability level. (b) Impact on bandwidth consumption. (c) Impact on application completion time.

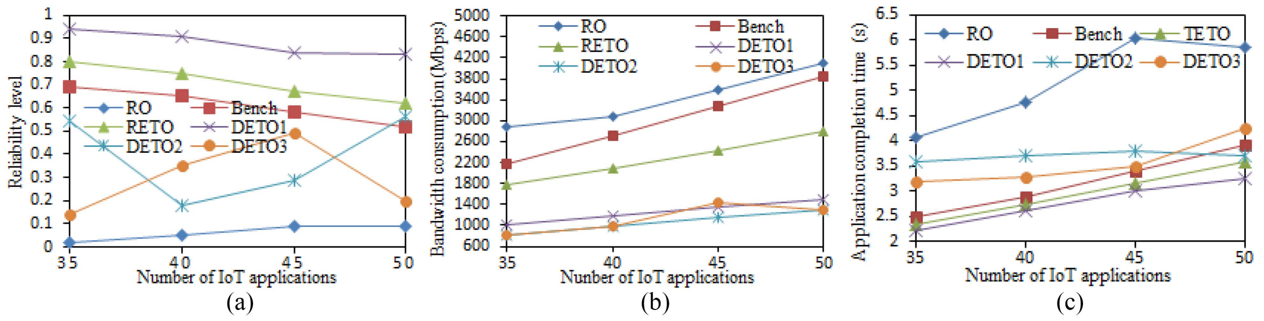

Fig. 6. Impact of the number of IoT applications. (a) Impact on reliability level. (b) Impact on bandwidth consumption. (c) Impact on application completion time.

to bottom along its directed acyclic graph when multiple VMs satisfy the constraints.

2) *Bench:* Select the VM with the satisfaction of the QoS requirements and the minimum value of the formulation $\min_{j\in \Omega} (T_{iqj} + \alpha \cdot bw_j), i \in I$ to accommodate each task of the IoT application in descending order of task instruction length when multiple VMs satisfy the constraints [33].

Next, the performance of RETO and DETO is analyzed by comparing them with RO and Bench in terms of the reliability level, bandwidth consumption, application completion time, and approach execution time while offloading and processing 50 IoT applications. We also analyze the impact of experimental parameters, including the number of base stations (as shown in Fig. 4), the number of VMs (as shown in Fig. 5), the number of IoT applications (as shown in Fig. 6), and the deadlines of IoT applications (as shown in Fig. 7) on these performance indicators. Please note that we set the positive tunable factors θ_1 and θ_2 as three different sets of values to study the influence of the DETO approach on the optimization objectives under different values of θ_1 and θ_2 [i.e., DETO1 (0.00005 and 0.99995), DETO2 (0.99995 and 0.00005), DETO3 (0.5 and 0.5)].

TABLE IV
IMPACT OF THE NUMBER OF BASE STATIONS ON APPROACH EXECUTION TIME

	RO	Bench	RETO	DETO1	DETO2	DETO3
45	23ms	82ms	86ms	18232ms	17372ms	16355ms
50	36ms	62ms	85ms	18716ms	18109ms	16897ms
55	26ms	51ms	77ms	15682ms	17398ms	17268ms
60	35ms	75ms	94ms	15718ms	17725ms	16600ms

A. Impact of the Number of Base Stations

As shown in Fig. 4 and Table IV, as the number of base stations increased by step size 5 from 45 to 60, the reliability level, bandwidth consumption, and application completion

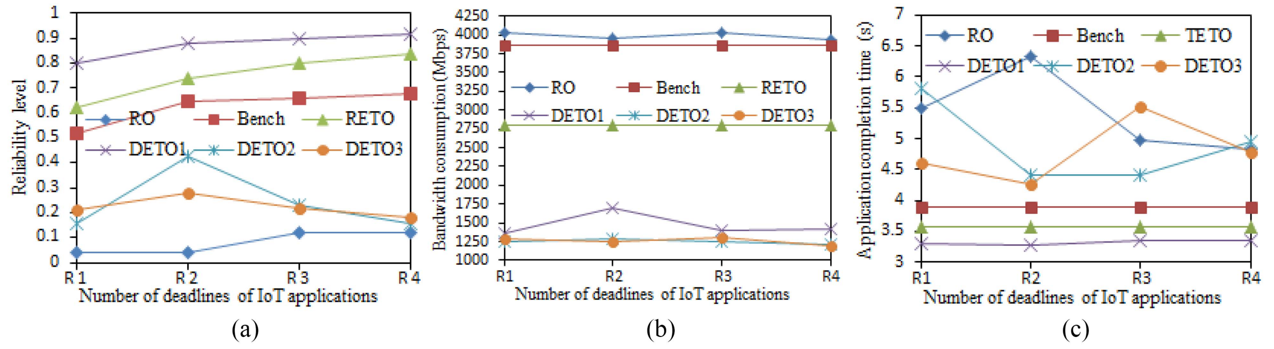

Fig. 7. Impact of the deadlines of IoT applications. (a) Impact on reliability level. (b) Impact on bandwidth consumption. (c) Impact on application completion time.

TABLE V
IMPACT OF THE NUMBER OF VMS ON APPROACH EXECUTION TIME

	RO	Bench	RETO	DETO1	DETO2	DETO3
400	30ms	65ms	83ms	16664ms	16465ms	15698ms
425	30ms	67ms	86ms	16995ms	16240ms	17133ms
450	32ms	65ms	90ms	18118ms	18039ms	19295ms
475	33ms	67ms	99ms	18451ms	18668ms	18954ms

TABLE VI
IMPACT OF THE NUMBER OF IOT APPLICATIONS ON APPROACH EXECUTION TIME

	RO	Bench	RETO	DETO1	DETO2	DETO3
35	58ms	51ms	70ms	12887ms	12405ms	12334ms
40	30ms	78ms	83ms	13796ms	14414ms	13970ms
45	26ms	65ms	93ms	14598ms	15128ms	15315ms
50	32ms	86ms	83ms	17790ms	17407ms	17669ms

time of each approach did not vary considerably. This outcome is likely observed because although the increase in the number of base stations increases the number of edge servers, the number of VMs and IoT applications does not change, and these VMs are still randomly assigned to these edge servers. The reason why we list the execution time of each approach is to show the time complexity of these approaches. Meanwhile, since different machines have different approach execution times, we do not have to go into their specific values. Because DETO is an intelligent optimization algorithm and takes massive time to iterate, its approach execution time has the maximum value of all approaches. Since the reliability level and bandwidth consumption are not the same order of magnitude and the positive tunable factors θ_1 and θ_2 determine the optimization emphasis, the optimization emphases of DETO1, DETO2, and DETO3 are the reliability level, bandwidth consumption, and bandwidth consumption, respectively. Furthermore, DETO1 has the maximum reliability level and the minimum application completion time; DETO2 and DETO3 have similar and minimum bandwidth consumption. Since DETO2 further optimizes bandwidth consumption than DETO3, DETO2 has such a low reliability level, even lower than the Bench approach. In addition to the DETO approach, RETO has the maximum reliability level, the minimum bandwidth consumption, the minimum application completion time, and the moderate approach execution time.

B. Impact of the Number of Virtual Machines

As shown in Fig. 5 and Table V, as the number of VMs increased by step size 25 from 400 to 475, the application completion time decreased, and the reliability level increased; moreover, the bandwidth consumption decreased. This is because the collaborative tasks of IoT applications are offloaded to VMs on the premise that these VMs are randomly assigned to edge servers. That is, an increase in the number of VMs can increase the probability of choosing a VM with a lower failure rate, a higher processing capacity, or a higher recovery rate, and then decrease the sum of the process time or recovery time for each task. Meanwhile, since the number of VMs increases, the probability of choosing a VM with lower bandwidth by formulation (23) or on the same edge server with other VMs is increased, and then the bandwidth consumption and the application completion time both decrease. Please note that since DETO2 and DETO3 both determine the optimization emphasis on the bandwidth consumption and the reliability level is too small relative to the bandwidth consumption, it is very difficult to take the value of the reliability level into account in the optimization process of DETO2 and DETO3. That is, the value of the reliability level is too random. Regardless of how the number of VMs changes, DETO1 has the maximum reliability level and minimum application completion time; DETO2 and DETO3 have similar and minimum bandwidth consumption. In addition to the DETO approach, RETO has the maximum reliability level, the minimum bandwidth consumption, the minimum application completion time, and the moderate approach execution time.

C. Impact of the Number of IoT Applications

As shown in Fig. 6 and Table VI, as the number of IoT applications increased by step 5 from 35 to 50, the reliability level decreased, and the bandwidth consumption and the application completion time both increased. Except for the increase in the approach execution time of DETO, other approaches are not almost varied. This is because that the increase in the number of IoT applications increases the number of collaborative tasks, which need to be offloaded onto VMs with different processing capacities, failure rates, and recovery rates. Therefore, the application completion time and the bandwidth consumption both increase to varying degrees. Furthermore,

TABLE VII
IMPACT OF THE DEADLINES OF IOT APPLICATIONS ON APPROACH EXECUTION TIME

	RO	Bench	RETO	DETO1	DETO2	DETO3
R1	24ms	75ms	83ms	18519ms	18056ms	19020ms
R2	30ms	48ms	87ms	16833ms	18310ms	17309ms
R3	30ms	72ms	86ms	17116ms	17533ms	18521ms
R4	35ms	63ms	78ms	16384ms	16092ms	18309ms

the reliability level of all approaches is reduced accordingly. Since the number of tasks determines the chromosome length of DE, its approach execution time increases with its number. Furthermore, DETO1 demonstrates the maximum reliability level and the minimum application completion time; DETO2 and DETO3 have similar and minimum bandwidth consumption. Since the optimization emphases of DETO2 and DETO3 are both bandwidth consumption, both of these approaches emphasize the optimization of bandwidth consumption, with little consideration of the reliability level. Therefore, in the optimization process of these two approaches, their bandwidth consumption is optimized very little, but the value of the reliability level will fluctuate greatly. In addition to the DETO approach, RETO has the maximum reliability level, the minimum bandwidth consumption, the minimum application completion time, and the moderate approach execution time.

D. Impact of the Deadlines of IoT Applications

As shown in Fig. 7 and Table VII, as the deadlines of IoT applications increased from R1 to R4, the reliability level increased, but the bandwidth consumption, application completion time, and approach execution time did not vary considerably. Note that R1, R2, R3, and R4 represent four ranges of deadlines of IoT applications, i.e., [60, 90], [62.5, 92.5], [65, 95], and [67.5, 97.5] ms, respectively. This is because the deadline of IoT applications is exploited only to compute the reliability level of 50 IoT applications via formulation (11). The increase in the deadline of IoT applications can prevent QoS violations and has no effect on bandwidth consumption, application completion time, or approach execution time. Therefore, when the range of deadlines of the IoT applications varied from R1 to R4, the probability of choosing a larger deadline was increased. That is, the reliability level of 50 IoT applications also increases with the increased range of deadlines of the IoT applications, i.e., from R1 to R4. Furthermore, DETO1 has a maximum reliability level and minimum application completion time; DETO2 and DETO3 have similar and minimum bandwidth consumption. In addition to the DETO approach, RETO has the maximum reliability level, the minimum bandwidth consumption, the minimum application completion time, and the moderate approach execution time.

VII. CONCLUSION AND FUTURE WORK

In this article, we investigated the collaborative task offloading problem by considering the failure rates and recovery rates of the containers or VMs. We also studied the tradeoff of minimizing the bandwidth consumption of IoT applications while maximizing the reliability level of these IoT applications during task offloading. Then, we proposed a multiobjective optimization problem, and transformed it into a single objective optimization problem. Finally, we introduced two approaches to acquire two near-optimal solutions with different time complexities. The results of simulation experiments demonstrated that our proposed approaches provide near-optimal solutions and have better performance than other approaches.

In our future work, we will further reduce the time complexity of the DETO approach. Meanwhile, we will also take remote clouds into account for our experimental environment and then study the above optimization problem based on a real data set.

REFERENCES

[1] B. Varghese and R. Buyya, "Next generation cloud computing: New trends and research directions," *Future Gener. Comput. Syst.*, vol. 79, pp. 849–861, Feb. 2018.

[2] P. Schulz *et al.*, "Latency critical IoT applications in 5G: Perspective on the design of radio interface and network architecture," *IEEE Commun. Mag.*, vol. 55, no. 2, pp. 70–78, Feb. 2017.

[3] P. Hu, S. Dhelim, H. Ning, and T. Qiu, "Survey on fog computing: Architecture, key technologies, applications and open issues," *J. Netw. Comput. Appl.*, vol. 98, pp. 27–42, Nov. 2017.

[4] K. Peng, H. Huang, S. Wan, and V. C. M. Leung, "End-edge-cloud collaborative computation offloading for multiple mobile users in heterogeneous edge-server environment," *Wireless Netw.*, to be published, doi: 10.1007/s11276-020-02385-1.

[5] C.-H. Hong and B. Varghese, "Resource management in fog/edge computing: A survey on architectures, infrastructure, and algorithms," *ACM Comput. Surveys*, vol. 52, no. 5, pp. 1–37, 2019.

[6] H. Guo, J. Liu, J. Ren, and Y. Zhang, "Intelligent task offloading in vehicular edge computing networks," *IEEE Wireless Commun.*, vol. 27, no. 4, pp. 126–132, Aug. 2020.

[7] T. Wang, Y. Lu, J. Wang, H.-N. Dai, X. Zheng, and W. Jia, "EIHDP: Edge-intelligent hierarchical dynamic pricing based on cloud-edge-client collaboration for IoT systems," *IEEE Trans. Comput.*, vol. 70, no. 8, pp. 1285–1298, Aug. 2021.

[8] H. Zhu and C. Huang, "Availability-aware mobile edge application placement in 5G networks," in *Proc. IEEE Global Commun. Conf. (GLOBCOM)*, Singapore, 2017, pp. 1–6.

[9] B. Yang, F. Tan, Y.-S. Dai, and S. Guo, "Performance evaluation of cloud service considering fault recovery," in *Proc. IEEE Int. Conf. Cloud Comput. (CLOUD)*, 2009, pp. 571–576.

[10] Q. Wei, Y. Liu, H. Zhang, and Y. Shui, "Enhancing crowd collaborations for software defined vehicular networks," *IEEE Commun. Mag.*, vol. 55, no. 8, pp. 80–86, Aug. 2017.

[11] X. Liu, S. X. Sun, and G. Huang, "Decentralized services computing paradigm for blockchain-based data governance: Programmability, interoperability, and intelligence," *IEEE Trans. Services Comput.*, vol. 13, no. 2, pp. 343–355, Mar./Apr. 2020.

[12] N. Zhao, Y.-C. Liang, D. Niyato, Y. Pei, M. Wu, and Y. Jiang, "Deep reinforcement learning for user association and resource allocation in heterogeneous cellular networks," *IEEE Trans. Wireless Commun.*, vol. 18, no. 11, pp. 5141–5152, Nov. 2019.

[13] M. S. Elbamby *et al.*, "Wireless edge computing with latency and reliability guarantees," *Proc. IEEE*, vol. 107, no. 8, pp. 1717–1737, Aug. 2019.

[14] S. K. Garg, S. Versteeg, and R. Buyya, "A framework for ranking of cloud computing services," *Future Gener. Comput. Syst.*, vol. 29, no. 4, pp. 1012–1023, 2019.

[15] A. Aral and I. Brandic, "Dependency mining for service resilience at the edge," in *Proc. IEEE/ACM Symp. Edge Comput. (SEC)*, Seattle, WA, USA, 2018, pp. 228–242, doi: 10.1109/SEC.2018.00024.

[16] P. Kumari and P. Kaur, "A survey of fault tolerance in cloud computing," *J. King Saud Univ. Comput. Inf. Sci.*, to be published, doi: 10.1016/j.jksuci.2018.09.021.

[17] A. Aral and I. Brandić, "Learning spatiotemporal failure dependencies for resilient edge computing services," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 7, pp. 1578–1590, Jul. 2021.
[18] L. Ponemon, "Cost of data center outages–data center performance benchmark series," Ponemon Inst., Traverse City, MI, USA, Rep., Jan. 2016, pp. 1–20.
[19] T. Wang, Y. Mei, X. Liu, J. Wang, H.-N. Dai, and Z. Wang, "Edge-based auditing method for data security in resource-constrained Internet of Things," *J. Syst. Archit.*, vol. 114, no. 5, 2021, Art. no. 101971.
[20] T. Wang *et al.*, "Mobile edge-enabled trust evaluation for the Internet of Things," *Inf. Fusion*, vol. 75, no. 3, pp. 90–100, 2021.
[21] Q. Wei, K. Wang, Y. Liu, N. Cheng, H. Zhang, and X. Shen, "Software-defined collaborative offloading for heterogeneous vehicular networks," *Wireless Commun. Mobile Comput.*, vol. 2018, Apr. 2018, Art. no. 3810350, doi: 10.1155/2018/3810350.
[22] J. Oueis, E. Calvanese-Strinati, A. De Domenico, and S. Barbarossa, "On the impact of backhaul network on distributed cloud computing," in *Proc. IEEE Wireless Commun. Netw. Conf. Workshops (WCNCW)*, Istanbul, Turkey, 2014, pp. 12–17.
[23] R. Storn and K. Price, "Differential evolution—A simple and efficient heuristic for global optimization over continuous spaces," *J. Global Optim.*, vol. 11, pp. 341–359, Dec. 1997.
[24] A. Aral and I. Brandic, "Quality of service channelling for latency sensitive edge applications," in *Proc. IEEE Int. Conf. Edge Comput. (EDGE)*, Honolulu, HI, USA, 2017, pp. 166–173.
[25] M. Soualhia, C. Fu, and F. Khomh, "Infrastructure fault detection and prediction in edge cloud environments," in *Proc. 4th ACM/IEEE Symp. Edge Comput. (SEC)*, 2019, pp. 222–235.
[26] A. Maia, Y. Ghamri-Doudane, D. Vieira, and M. F. de Castro, "Optimized placement of scalable IoT services in edge computing," in *Proc. IFIP/IEEE Symp. Integr. Netw. Serv. Manage. (IM)*, Arlington, VA, USA, 2019, pp. 189–197.
[27] A. Maia, Y. Ghamri-Doudane, D. Vieira, and M. F. de Castro, "A multi-objective service placement and load distribution in edge computing," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Waikoloa, HI, USA, 2019, pp. 1–7, doi: 10.1109/GLOBECOM38437.2019.9014303.
[28] H. Zhao, S. Deng, Z. Liu, J. Yin, and S. Dustdar, "Distributed redundancy scheduling for microservice-based applications at the edge," *IEEE Trans. Services Comput.*, early access, Aug. 3, 2020, doi: 10.1109/TSC.2020.3013600.
[29] Y. Liu *et al.*, "Dependency-aware task scheduling in vehicular edge computing," *IEEE Internet Things J.*, vol. 7, no. 6, pp. 4961–4971, Jun. 2020.
[30] L. Zhao and J. Liu, "Optimal placement of virtual machines for supporting multiple applications in mobile edge networks," *IEEE Trans. Veh. Technol.*, vol. 67, no. 7, pp. 6533–6545, Jul. 2018.
[31] M. Goudarzi, H. Wu, M. Palaniswami, and R. Buyya, "An application placement technique for concurrent IoT applications in edge and fog computing environments," *IEEE Trans. Mobile Comput.*, vol. 20, no. 4, pp. 1298–1311, Apr. 2021.
[32] L. Liu, H. Tan, S. H.-C. Jiang, Z. Han, X.-Y. Li, and H. Huang, "Dependent task placement and scheduling with function configuration in edge computing," in *Proc. IEEE Int. Workshop Qual. Serv. (IWQoS)*, 2019, pp. 1–10.
[33] J. Yao and N. Ansari, "Fog resource provisioning in reliability-aware IoT networks," *IEEE Internet Things J.*, vol. 6, no. 5, pp. 8262–8269, Oct. 2019.
[34] M. H. N. Yousefi, A. Ghiassi, B. S. Hashemi, and M. Goudarzi, "Workload scheduling on heterogeneous mobile edge cloud in 5G networks to minimize SLA violation," 2020. [Online]. Available: arXiv:2003.02820.
[35] B. Hu, J. Chen, and F. Li, "A dynamic service allocation algorithm in mobile edge computing," in *Proc. IEEE Int. Conf. Inf. Commun. Technol. Converg. (ICTC)*, Jeju, South Korea, 2017, pp. 104–109.
[36] V. Farhadi *et al.*, "Service placement and request scheduling for data-intensive applications in edge clouds," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Paris, France, 2019, pp. 1279–1287.
[37] H. Wu, Z. Zhang, C. Guan, K. Wolter, and M. Xu, "Collaborate edge and cloud computing with distributed deep learning for smart city Internet of Things," *IEEE Internet Things J.*, vol. 7, no. 9, pp. 8099–8110, Sep. 2020.
[38] J. Meng, H. Tan, C. Xu, W. Cao, L. Liu, and B. Li, "Dedas: Online task dispatching and scheduling with bandwidth constraint in edge computing," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Paris, France, 2019, pp. 2287–2295.
[39] G. Huang, C. Luo, K. Wu, Y. Ma, Y. Zhang, and X. Liu, "Software-defined infrastructure for decentralized data lifecycle governance: Principled design and open challenges," in *Proc. IEEE 39th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, 2019, pp. 1674–1683, doi: 10.1109/ICDCS.2019.00166.
[40] J. Liu, H. Guo, J. Xiong, N. Kato, J. Zhang, and Y. Zhang, "Smart and resilient EV charging in SDN-enhanced vehicular edge computing networks," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 1, pp. 217–228, Jan. 2020.
[41] H. Guo and J. Liu, "UAV-enhanced intelligent offloading for Internet of Things at the edge," *IEEE Trans. Ind. Informat.*, vol. 16, no. 4, pp. 2737–2746, Apr. 2020.
[42] F. Wang, J. Xu, X. Wang, and S. Cui, "Joint offloading and computing optimization in wireless powered mobile-edge computing systems," *IEEE Trans. Wireless Commun.*, vol. 17, no. 3, pp. 1784–1797, Mar. 2018.
[43] M. Bari *et al.*, "Data center network virtualization: A survey," *IEEE Commun. Surveys Tuts.*, vol. 15, no. 2, pp. 909–928, 2nd Quart., 2013.
[44] J. Békési, G. Galambos, and H. Kellerer, "A 5/4 linear time bin packing algorithm," *J. Comput. Syst. Sci.*, vol. 60, no. 1, pp. 145–160, 2000.
[45] K. Deb, "Multi-objective optimization," in *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, E. K. Burke and G. Kendall, Eds. Boston, MA, USA: Springer, 2005, pp. 273–316.
[46] R. Storn, "On the usage of differential evolution for function optimization," in *Proc. North Amer. Fuzzy Inf. Process. Soc. (NAFIPS)*, Berkeley, CA, USA, 1996, pp. 519–523.
[47] L. Tang, Y. Dong, and J. Liu, "Differential evolution with an individual-dependent mechanism," *IEEE Trans. Evol. Comput.*, vol. 19, no. 4, pp. 560–574, Aug. 2015.
[48] J. Liu, S. Wang, A. Zhou, S. A. P. Kumar, F. Yang, and R. Buyya, "Using proactive fault-tolerance approach to enhance cloud service reliability," *IEEE Trans. Cloud Comput.*, vol. 6, no. 4, pp. 1191–1202, Oct./Dec. 2018.
[49] H. Gupta, A. V. Dastjerdi, S. K. Ghosh, and R. Buyya, "iFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things, edge and fog computing environments," *Softw. Pract. Exp.*, vol. 47, no. 9, pp. 1275–1296, 2017.
[50] A. Beloglazov and R. Buyya, "Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers," *Concurrency Comput. Pract. Exp.*, vol. 24, no. 13, pp. 1397–1420, 2012.
[51] A. P. Miettinen and J. K. Nurminen, "Energy efficiency of mobile clients in cloud computing," in *Proc. 2nd USENIX Conf. Hot Topics Cloud Comput. (HotCloud)*, 2010, p. 4.

Jialei Liu received the Ph.D. degree in computer science and technology from Beijing University of Posts and Telecommunications, Beijing, China, in 2018.

He is currently an Assistant Professor with the School of Software Engineering, Anyang Normal University, Anyang, China. He has published more than 20 research papers. His major research interests include cloud computing and mobile edge computing.

Ao Zhou (Member, IEEE) received the Ph.D. degree from Beijing University of Posts and Telecommunications, Beijing, China, in 2015.

She is currently an Associate Professor with the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications. She has published more than 20 research papers. She played a key role at many international conferences. Her research interests include cloud computing and edge computing.

Chunhong Liu (Member, IEEE) received the M.E. degree in computer science and technology from Xidian University, Xi'an, China, in 2005, and the Ph.D. degree in computer science and technology from Beijing University of Posts and Telecommunications, Beijing, China, in 2018.

She is an Associate Professor with the Department of Computer and Information Engineering, Henan Normal University, Xinxiang, China. Her major research interests include cloud computing, edge computing, machine learning, and oriented-service computing.

Tongguang Zhang received the Ph.D. degree in computer science and technology from Beijing University of Posts and Telecommunications, Beijing, China, in June 2018.

He is an Associate Professor of Computer Science with Xinxiang University, Xinxiang, China. His current research interests include mobile Internet technology, Internet of Things technology, communication software and distribute computing, embedded system, and service computing.

Lianyong Qi (Member, IEEE) received the Ph.D. degree from the Department of Computer Science and Technology, Nanjing University, Nanjing, China, in 2011.

In 2010, he visited the Department of Information and Communication Technology, Swinburne University of Technology, Melbourne, VIC, Australia. He is currently a Full Professor with the School of Information Science and Engineering, Qufu Normal University, Jining, China. He has published over 90 research papers (first author or corresponding author) in international journals and conferences. His research interests include big data and recommender systems.

Shangguang Wang (Senior Member, IEEE) received the Ph.D. degree from Beijing University of Posts and Telecommunications (BUPT), Beijing, China, in 2011.

He is currently a Professor and Deputy Director with the State Key Laboratory of Networking and Switching Technology, BUPT. He has published more than 100 papers. His research interests include edge computing, service computing, and cloud computing.

Prof. Wang played key roles, such as the general chair or the PC chair for many international conferences. He is the Editor-in-Chief of the *International Journal of Web Science*.

Rajkumar Buyya (Fellow, IEEE) received the Ph.D. degree in computer science and software engineering from Monash University, Melbourne, VIC, Australia, in 2002.

He is a Redmond Barry Distinguished Professor and the Director of the Cloud Computing and Distributed Systems (CLOUDS) Laboratory with The University of Melbourne, Melbourne. He has authored over 750 publications and seven text books, including *Mastering Cloud Computing* (McGraw Hill, New York, NY, USA; China Machine Press, Beijing, China; and Morgan Kaufmann for Indian, and Chinese and international markets).

Prof. Buyya is one of the Highly Cited Authors in computer science and software engineering worldwide (H-index=134, G-index = 304, and more than 100 800 citations). He served as the founding Editor-in-Chief of the IEEE TRANSACTIONS ON CLOUD COMPUTING. He is currently serving as a Co-Editor-in-Chief of *Journal of Software: Practice and Experience*, which was established 50 years ago. For further information on Dr. Buyya, please visit his cyberhome: www.buyya.com.