Reliability-Enhanced Task Offloading in Mobile Edge Computing Environments

Jialei Liu, Shangguang Wang, *Senior Member*, *IEEE*, Lianyong Qi, *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, autonomous driving) targeted for mobile edge computing environments. These IoT applications are often split into multiple collaborative tasks and offloaded onto the containers or virtual machines with certain failure rates and recovery rates. If these containers or virtual machines are not deployed in the same edge servers, the bandwidth resources of edge clouds need to be consumed to transfer data. These factors increase the completion time of IoT application to different degrees, and then affect its reliability level. Therefore, there exists equilibrium between reliability level and the bandwidth consumption. In this paper, 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 an integer linear programming problem with high time complexity. We introduce an efficient approach to acquire a near-optimal solution with high computation efficiency. The results of simulation experiments demonstrate that our proposed approach can observably enhance the reliability level and reduce the bandwidth consumption of IoT applications compared with other related approaches.

Index Terms—Mobile edge computing; IoT application; task offloading; reliability level; bandwidth consumption

1 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 have increased dramatically and they have become ubiquitous in our modern digital society. It is predicted that about 29 billion IoT devices will be connected to the Internet by 2022 [1]. Subsequently, these IoT devices generate massive latency-sensitive applications, which have stringent delay requirements (e.g., real-time responses on a timescale of 10ms or even 1ms [2]) and require a large number of processing and bandwidth resources [3]. 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, autonomous driving) cannot be handled locally effectively [4].

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 the edge clouds for processing by containers or virtual machines [5]. If these containers or virtual machines are not on the same edge servers, the communication between these collaborative tasks will consume certain bandwidth resources and communication time. Meanwhile, considering that these containers or virtual machines own heterogeneous failure rates and recovery rates [6], the total time that the container or virtual machine processes each task includes not only the processing time but also the recovery time after failures. Since the latency-sensitive IoT applications usually have a certain deadline, the application completion time beyond this deadline degrades the QoS of users. Further, the reliability level of the IoT application is also reduced [7]. Therefore, to reduce the bandwidth consumption of the edge clouds and enhance the reliability level of the IoT applications, how to make reliability-enhanced task offloading scheme with bandwidth constraint becomes the most significant challenge.

The containers or virtual machines consume certain bandwidth resources to transfer the data from the sending tasks while the collaborative tasks are being processed. Since the edge clouds are often interconnected with fiber optic cables [8], 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 and communication time of other collaborative tasks may be reduced and increased, respectively. Moreover, the container or virtual machine failures may prolong the application completion time including the communication time, task processing time, and recovery time, and then reduce the reliability level of IoT applications [7]. Therefore, there is a contradiction between the reliability level and the bandwidth consumption while dealing with a batch of IoT applications, how to deal with this contradiction well is still unknown.

- *J. Liu is with the School of Software Engineering, Anyang Normal University, Anyang, China. Email: jlliu01850@aynu.edu.cn.*
- *S. Wang is with the State Key Laboratory of Networking and Switching technology, Beijing University of Posts and Telecommunications, Beijing, China. Email: sgwang@bupt.edu.cn.*
- *L. Qi is with the School of Information Science and Engineering, Qufu Normal University, Shandong, China. Email: lianyongqi@qfnu.edu.cn.*
- *R. Buyya is with the Cloud Computing and Distributed Systems (CLOUDS) Laboratory, School of Computing and Information Systems, The University of Melbourne, Australia. Email: rbuyya@unimelb.edu.au.*

Table 1: 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
[9]	×	Single	×	√	×	√	×	Bayesian Network	√	√	×
[10]	×	Single	×	√	×	√	×	Machine learning Statistical Techniques	√	×	×
[11]	√	Single	×	√	×	×	√	Genetic Algorithm	×	√	×
[12]	√	Single	×	√	×	×	√	Genetic Algorithm	√	√	×
[13]	√	multiple	√	×	√	√	×	SAA-RS	×	√	×
[14]	×	multiple	×	√	√	√	×	MAMTS	×	√	×
[15]	√	Single	×	×	×	×	√	SEHPA	×	√	×
[16]	×	multiple	√	√	×	×	√	GenDoc	×	√	×
[17]	×	multiple	√	×	×	×	√	APMA	×	√	×
[18]	×	Single	×	×	√	×	√	MBFD	√	×	×
[19]	×	Single	×	√	×	√	×	MESA	×	√	×
[20]	×	multiple	√	×	×	×	√	DSA	×	×	√
[21]	√	multiple	√	×	×	×	√	GSP-SS	×	×	√
[22]	×	multiple	×	×	×	×	√	DDTO	×	√	×
[5]	×	multiple	√	×	×	√	×	Heuristic Algorithm	×	×	√
[23]	×	multiple	√	×	√	√	×	Dedas	×	√	×
Our Scheme	×	Multiple	√	√	×	√	×	RETO	√	√	√

To simultaneously maximize the reliability level and minimize the bandwidth consumption of IoT applications, more efficient approaches with lower time complexity are required to solve such multi-objective optimization problem. In this paper, we first transform it into a single-objective optimization problem which is considered as an integer linear programming problem. And then we propose a Reliability-Enhanced Task Offloading approach (RETO) to obtain near-optimal task offloading scheme.

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

- We formulate the collaborative task offloading problem of IoT applications as an integer linear programming problem. To simultaneously maximize the reliability level and minimize the bandwidth consumption of IoT applications, we establish a bandwidth consumption model and a Poisson process-based reliability model.
- We propose a near-optimal approach with lower time complexity, where these IoT applications are sorted in ascending order of deadlines, and their directed acyclic graphs are analyzed to efficiently and efficiently generate the collaborative task offloading schemes over the edge clouds.
- We conduct a simulation experimental environment to comprehensively evaluate the effectiveness and efficiency of our proposed approach. When compared with other related approaches, our proposed approach can obtain better performance.

The rest of the paper is organized as follows. The related work is introduced in Section 2; our system model is defined in Section 3; the problem formulation is presented based on the above models in Section 4; the technical details of the problem solution is introduced in Section 5; the evaluation experiments are conducted in Section 6. Finally, the conclusions along with future work are presented in Section 7.

2 Related Work

There are several works have focused on task offloading of IoT applications – key ones are shown in Table 1. Aral et al. [9] proposed a Bayesian network model based on QoS related parameters to estimate the availability level of virtual machines in edge infrastructure, so as to avoid the deterioration response time limit that is critical to edge applications. Soualhia et al. [10] 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. [11] jointly studied the vertical and horizontal load distribution and location of scalable IoT services to minimize the 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 [12]. Zhao et al. [13] proposed a distributed redundant scheduling algorithm to solve the availability problem of microservice-based applications caused by container failures. Although these literatures improved the availability level of the applications in different ways, they did not solve the reliability problem by the consideration of the application completion time. For this purpose, Liu et al. [14] introduced an efficient task scheduling algorithm to minimize the average completion time of multiple applications. This algorithm ensured the completion time con-

straint of applications and processing dependency requirements of tasks by prioritization of multiple applications and tasks. Zhao et al. [15] investigated in detail the optimal deployment of virtual machine replicas supporting multiple applications to minimize the average response time and total cost for services provision. Goudarzi et al. [16] 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. [17] introduced a new approximation algorithm to effectively solve the problem of dependent task allocation and scheduling with on-demand function configuration on edge servers, so as to minimize the application completion time. However, they did not consider the effect of the virtual machine failures on the application completion time, and the contradictory relationship between the bandwidth consumption and the application completion time. Yao et al. [18] proposed an efficient algorithm to achieve a balance between maximizing the service reliability and minimizing the rental cost of virtual machines for fog resource provisioning in IoT networks. Yousefi et al. [19] 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 literatures only studied the task scheduling of applications, but did not consider the communication relationship between these tasks and the impact on the application completion time.

Moreover, Hu et al. [20] 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. [21] introduced a two-time-scale framework to jointly optimize services deployment and the request scheduling within the constraints of the storage, communication, computation, and budget, and adapt over time to serve time-varying demands under the consideration of the system stability and operation cost. Wu et al. [22] formulized the hybrid task deployment problem as a multi-objective 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 literatures did not consider the effect of different task or service deployment schemes on the application completion time and the bandwidth consumption of edge clouds.

Considering the joint optimization of the application completion time and the bandwidth consumption, Zhu et al. [5] presented a cost model considering inter-host network performance and CPU/ memory overuse to track the impact of changing the deployment strategy of mobile edge applications on the availability and inter-host network bandwidth cost. Meng et al. [23] 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 literatures did not study the effects of communication between the collaborative tasks on the bandwidth consumption of the edge clouds and the application completion time, and thus on reliability level.

Through the analysis of the foregoing literatures, we can find that no recent studies have focused on the collaborative task offloading of IoT applications under the consideration of the container or virtual machine failures and the bandwidth consumption between these collaborative tasks. In this paper, we assume that the collaborative tasks of IoT applications are offloaded onto multiple heterogeneous containers or virtual machines with the heterogeneous failure rates and recovery rates. The selection of different containers or virtual machines results in different processing time and failover time, on the other hand, different edge servers accommodated these containers or virtual machines consume different bandwidth resources, thus resulting in different communication time. Furthermore, the application completion time is greatly affected, thus affecting the reliability level of these IoT applications. Therefore, we propose a near-optimal approach with lower time complexity to simultaneously maximize the reliability level and minimize the bandwidth consumption of IoT applications.

3 System Model

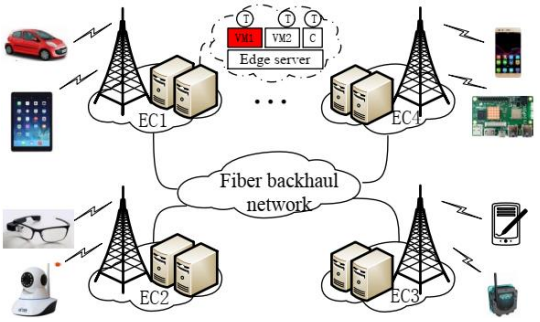


Fig. 1. Reliability-enhanced mobile edge computing system

We propose a reliability-aware mobile edge computing system model (see Fig. 1). This includes multiple edge clouds (ECs) to provide the IoT services by an IoT system operator, and these edge clouds are interconnected through fiber backhaul network using full mesh topology [8]. Since this paper focuses on the task offloading of the latency-sensitive IoT applications, all IoT devices can access and only offload their latency-sensitive IoT applications to multiple edge clouds instead of the remote cloud. Each edge cloud is deployed near the IoT devices and owns a certain number of heterogeneous edge servers. Each edge server accommodates a certain number of heterogeneous containers (Cs) or virtual machines (VMs). Each latency-sensitive IoT application consists of multiple collaborative tasks (Ts) modeled as a directed acyclic

graph, as shown in Fig. 2. Each task is indivisible and can only be processed by a container or virtual machine.

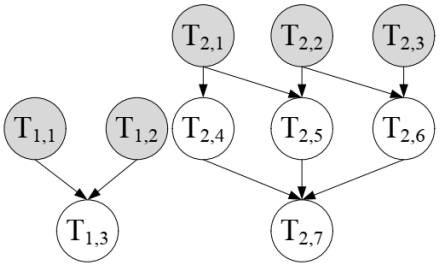


Fig. 2. An example of the directed acyclic graph of IoT applications

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 heterogonous containers or virtual machines indexed by the set $Z=\{1,2,\ldots,M\}$ are randomly assigned to these edge servers. The container or virtual machine j owns 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 the latency-sensitive IoT applications in arbitrary order and time, G IoT applications indexed by the set $\mathrm{I}=\{1,2,\ldots,G\}$ are generated at some point. The IoT application i owns deadline requirement D_i in units of seconds and is split into $|Q_i|$ tasks, and the task $q \in Q_i$ owns a certain instruction length l_{iq}. When these tasks are offloaded to these edge clouds, the state of all containers or virtual machines will be analyzed to determine that these tasks should be assigned to which containers or virtual machines. Meanwhile, the containers or virtual machines that are not assigned tasks can be shut down. As the containers or virtual machines with higher failure rates may fail during task processing, and the containers or virtual machines in the same cluster may be deployed in different edge servers, these reasons both affect the application completion time, the bandwidth consumption, and the reliability level of the IoT applications. Therefore, it is critical to design a reliability-enhanced task offloading scheme for these IoT applications.

3.1 Bandwidth Consumption Model

When the IoT applications are generated by IoT devices, each IoT application will be split into multiple collaborative tasks, and offloaded to the edge servers to deliver the cluster composed of multiple containers or virtual machines for processing. Considering the communication between the containers or virtual machines 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 virtual machines in the edge clouds. That is due to that if the containers or virtual machines in the same cluster are on the same edge server, the communication between the containers or virtual machines 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 G IoT applications can be expression by the formulation (1).

$$bandwidthC=\sum_{i=1}^{G}\sum_{q=1}^{|Q_i|}\sum_{h=1}^{H}\sum_{p=1}^{|P_h|}\sum_{m=1}^{|M_p|}\left(x_{qm}\cdot\sum_{r=1,r\notin M_p}^{M} b_{mr}\cdot bw_m\right) \quad (1)$$

where P_h denotes the set of the edge servers owned by the h-th edge cloud; M_p denotes the set of the containers or virtual machines on the p-th edge server; the binary variable x_{qm} denotes whether the q-th task is deployed to the m-th container or virtual machine, $x_{qm}=1$ if affirmative, otherwise $x_{qm}=0$; the binary variable b_{mr} denotes whether the task q being processed by the m-th container or virtual machine is a sender and the same cluster with the other tasks being processed by the r-th container or virtual machine, $b_{mr}=1$ if affirmative, otherwise $b_{mr}=0$. Note that the r-th container or virtual machine is not on the same edge server as the m-th container or virtual machine. bw_m denotes the bandwidth from the m-th container or virtual machine to the r-th container or virtual machine, which is specified as a random value in a certain range.

3.2 Reliability Model

The container or virtual machine may fail while a container or virtual machine is processing a task. To better study the impact of the container or virtual machine failures on the reliability level of IoT application, these failures are treated as recoverable [6]. That is, the failed container or virtual machine resumes the task processing after a period of time (i.e., recovery time). Considering that the virtualization technology has the isolation nature, all container or virtual machine failures are considered to be independent of each other [24]. Moreover, the processing time taken by the container or virtual machine j to complete task q of IoT application i without failure is $t_{iqj}=l_{iq}/c_j$. During the time interval $(0,\ t_{iqj}]$, the failures occur and are subject to a Poisson process with failure rate parameter λ_j [6], and the total number of failures in the container or virtual machine j is denoted by $N_j(t_{iqj})$. Therefore, the probability of $N_j(t_{iqj})=k$ can be computed by the formulation (2).

$$\Pr\{N_j(t_{iqj})=k\}=\frac{(\lambda_j t_{iqj})^k}{k!}e^{-\lambda_j t_{iqj}},k\geq 0 \quad (2)$$

The mean of $N_j(t_{iqj})$ can be computed by the formula-

tion (3).

$$E[N_j(t_{iqj})] = \lambda_j t_{iqj} \tag{3}$$

The recovery time $R_{jk}(t_{iqj})$ of the *k*-th failure at the container or virtual machine *j* is an exponentially distributed random variable with recovery rate parameter u_j [6]. Therefore, the total recovery time $R_j(t_{iqj})$ at the *j*-th container or virtual machine can be computed by the formulation (4). As the total recovery time $R_j(t_{iqj})$ is a compound Poisson process, its mean can be computed by the formulation (5).

$$R_j(t_{iqj}) = \sum_{k=1}^{N_j(t_{iqj})} R_{jk}(t_{iqj}) \tag{4}$$

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

The average total processing time T_{iqj} of the *q*-th task of IoT application *i* on the *j*-th container or virtual machine consists of the task processing time t_{iqj} and failure recovery time $E[R_j(t_{iqj})]$, its value can be computed by the formulation (6).

$$T_{iqj} = t_{iqj} + E[R_j(t_{iqj})] = t_{iqj} + \frac{\lambda_j t_{iqj}}{u_j}, \forall i \in I, j \in Z, q \in Q_i \tag{6}$$

Moreover, if part or all of the containers or virtual machines in the same cluster are not on the same edge server, these containers or virtual machines with the communication relationships need to take time to transfer data. Therefore, the total processing time (i.e., $totalT_i$) of IoT application *i* consists of the task processing time, the failure recovery time and the task communication time, as shown in formulation (7).

$$totalT_i = \arg\max_{v \in W}\left(\sum_{q=1}^{|v|}\left(T_{iqj} + \sum_{z=1}^{|v|}\left(\beta_{qz} \cdot \rho_{qz} \cdot \frac{data_q}{bw_j}\right)\right)\right), j \in Z \tag{7}$$

where *W* denotes the set of execution routes such as {{$T_{2,1}$, $T_{2,4}$, $T_{2,7}$},{$T_{2,1}$, $T_{2,5}$, $T_{2,7}$},{$T_{2,2}$, $T_{2,5}$, $T_{2,7}$},{$T_{2,2}$, $T_{2,6}$, $T_{2,7}$},{$T_{2,3}$, $T_{2,6}$, $T_{2,7}$}} 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., {$T_{2,1}$, $T_{2,4}$, $T_{2,7}$}) of the set *W*; the binary variable β_{qz} denotes whether the task *q* and the task *z* are on the same edge server, $\beta_{qz} = 0$ if affirmative, otherwise $\beta_{qz} = 1$; the binary variable ρ_{qz} denotes whether the task *q* sends data to the task *z*, $\rho_{qz} = 1$ if affirmative, otherwise $\rho_{qz} = 0$; $data_q$ denotes the amount of data sent by the task *q*; bw_j denotes the bandwidth of the container or virtual machine *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 a requested service can be successfully provided. This paper mainly studies how the container or virtual machine failures and recoveries affect the total processing time of the IoT applications (i.e., QoS). If the total processing time of the 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 [7]. Furthermore, the reliability level of the IoT applications can be computed by the probability of QoS violations, as shown in the formulation (8). *U* represents the total number of IoT applications with the QoS violations and can be computed by the formulation (9) and formulation (10).

$$\text{Re}liability = 1 - \frac{U}{G} \tag{8}$$

$$U = \sum_{i=1}^{G} F\left(totalT_i - D_i\right) \tag{9}$$

$$F(totalT_i - D_i) = \begin{cases} 1, & totalT_i - D_i > 0 \\ 0, & totalT_i - D_i \le 0 \end{cases} \tag{10}$$

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

4 Problem Formulation

In our mobile edge computing system, the edge clouds are interconnected via fiber optic cables, the communication between the edge clouds is considered to be load independent. Therefore, if the containers or virtual machines handling the collaborative tasks of IoT application *i* are not deployed in the same edge servers, the communication delay between these containers or virtual machines depends mostly on the bandwidth and data size of the containers or virtual machines in the sending state. Meanwhile, as the containers or virtual machines own 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 own 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 and the reliability level of the IoT applications in which other tasks reside are reduced. Therefore, we need to find a tradeoff to minimize the bandwidth consumption of *G* IoT applications while maximizing the reliability level of these IoT applications during task offloading, which can be denoted by the formulation (11).

$$F0: \begin{cases} \min \ \ bandwidthC \\ \max \ 1 - \dfrac{1}{G}\displaystyle\sum_{i=1}^{G} F\left(totalT_i - D_i\right) \end{cases} \tag{11}$$

s.t.

$$\sum_{m=1}^{M} x_{qm} = 1, q \in Q_i, i \in I \tag{12}$$

$$\sum_{i=1}^{G}\sum_{q=1}^{|Q_i|} l_{iq} x_{qm} \le L_m, m \in Z \tag{13}$$

where the formulation (12) denotes that each task can only be deployed to one container or virtual machine; the formulation (13) denotes that the memory capacity of each container or virtual machine 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 nonlinear, and is very difficult to solve. Therefore, a Boolean variable B_i is defined and assigned a value of $F(totalT_i - D_i), \forall i \in I$. $B_i = F(totalT_i - D_i)$ can also be denoted as the formulation (14) [18]. $\Re$ denotes a very large positive number. Furthermore, the formulation (14) can be transform into the formulation (15) by adding a small positive number ε, which is regarded as the tolerance of the QoS violation in units of seconds.

$$-\Re \cdot (1-B_i) < totalT_i - D_i \le \Re \cdot B_i, \forall i \in I \quad (14)$$

$$-\Re \cdot (1-B_i) + \varepsilon \le totalT_i - D_i \le \Re \cdot B_i, \forall i \in I \quad (15)$$

Then, the optimization problem $F0$ can be turned into an integer linear programming problem, as shown in the formulation (16).

$$F1: \begin{cases} \min \ bandwidthC \\ \max \ 1 - \dfrac{1}{G}\sum_{i=1}^{G} B_i \end{cases} \quad (16)$$

s.t. (12), (13), (15)

$$B_i \in \{0,1\}, \forall i \in I \quad (17)$$

Algorithm 1: Reliability-Enhanced Task Offloading Approach (RETO)

Input: *G*, *H*, *M*, *P*, α , θ_1 , θ_2 , directed acyclic graphs of IoT applications

Output: $\mathrm{Re}liability, bandwidthC, \sum_{i=1}^{G} totalT_i$

Initialize all parameters of the RETO
Sort *G* IoT applications in ascending order via their deadline
for *i* =1 to *G* **do**
  Obtain the number of structure layers of the IoT application *i* A_i
  Divide Q_i into A_i groups, $TaskGroup[l], l \in \{1,2,...,A_i\}$
  Sort $TaskGroup[2]$ in descending order via instruction length
  for *q*=1 to $TaskGroup[2].size$ **do**
    Select the container or virtual machine *j* with minimum T_{iqj} , $j \in Z$
  end for
  Sort $TaskGroup[1]$ in descending order via instruction length
  for *q*=1 to $TaskGroup[1].size$ **do**
    Select the container or virtual machine *j* with minimum ω_q , $j \in \Omega$
  end for
  if $A_i \ge 3$ **then**
    for *l*=3 to A_i **do**
      Sort $TaskGroup[l]$ in descending order via instruction length
      for *q*=1 to $TaskGroup[l].size$ **do**
        Select the container or virtual machine *j* with minimum ω_q , $j \in \Omega$
      end for
    end for
  end if
end for
for *i* =1 to *G* **do**
  Obtain the set of execution routes *W*
  for γ =1 to $|W|$ **do**
    Obtain the $totalT_i$ via the formulation (7)
  end for
  Compute $\sum_{i=1}^{G} totalT_i$ via $totalTime += totalT_i$
  Compute $\mathrm{Re}liability$ via the formulation (8)
  Compute $bandwidthC$ via the formulation (1)
end for
return $\mathrm{Re}liability, bandwidthC, \sum_{i=1}^{G} totalT_i$

However, as the optimization problem $F1$ is still the multi-objective optimization problem, a widely used weighted sum is exploited to convert it into a single-objective optimization problem through combining the objectives with weights, which represent the relative importance of the individual objectives [25]. The single-objective optimization problem $F2$ can be denoted by the formulation (18).

$$F2: \min \ \theta_1 \cdot bandwidthC + \frac{\theta_2}{G}\sum_{i=1}^{G} B_i - \theta_2 \quad (18)$$

s.t. (12), (13), (15), (17)

where θ_1 and θ_2 are both positive tunable factors.

Although the single-objective optimization problem $F2$ is still an integer linear programming problem and can be solved by many approaches (e.g., branch-and-bound and exhaustive) or optimization tools (e.g., CPLEX), these approaches or tools usually suffer from high computational complexity [26]. Therefore, it is an urgent need to adopt a near-optimal approach with lower time complexity to solve the problem.

5 Problem Solution

As discussed earlier, the integer linear programming problem $F2$ owns great time complexity, and it is not feasible to find an optimal solution on such a large scale. Therefore, we introduce a near-optimal approach to solve it and build a mobile edge computing system (Fig. 1) to verify this approach. In this system, all containers or virtual machines are randomly assigned to the edge servers, and each IoT application is split into multiple collaborative tasks (Fig. 2) and offloaded to these containers or virtual machines via our proposed approach. In next section, we introduce the implementation scheme of our proposed approach.

When these collaborative tasks of IoT applications are offloaded and assigned to the containers or virtual machines, our proposed approach will need to exploit six phases to obtain the near-optimal offloading scheme. First, we 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., $T_{1,1}$, $T_{1,2}$, $T_{2,1}$, $T_{2,2}$, $T_{2,3}$) being the sending tasks and the bottom-level tasks (e.g., $T_{1,3}$, $T_{2,7}$) being the receiving tasks, the tasks in the middle layer (e.g., $T_{2,4}$, $T_{2,5}$, $T_{2,6}$) can both send and receive results at some point. Third, we offload these tasks that receive the results of the top-level tasks in descending order of their instruction length, and respectively select the containers or virtual machines with the minimum sum of tasks processing time and recovery time via the formulation (6). Fourth, the top-level tasks are sorted in descending order of instruction length and select the containers or virtual machines to be deployed. For the top-level task *q* of IoT application *i*, its serving container or virtual machine first selects a container or virtual machine

on the same edge server as the containers or virtual machines where the receiving tasks (e.g., $T_{2,4}$, $T_{2,5}$, $T_{2,6}$) reside. If no such containers or virtual machines exist, the container or virtual machine j for the task q is selected via the formulation (19). That is, the container or virtual machine j is such container or virtual machine that minimizes the value of ω_q.

$$\omega_q = \arg\max_{j\in\Omega}(T_{iqj} + \frac{data_q}{bw_j} + \alpha \cdot bw_j), i \in I \quad (19)$$

where α is a small positive tunable factor; Ω denotes a set of containers or virtual machines accommodating the offloading task q (e.g., fourth step) or sending tasks (e.g., last step).

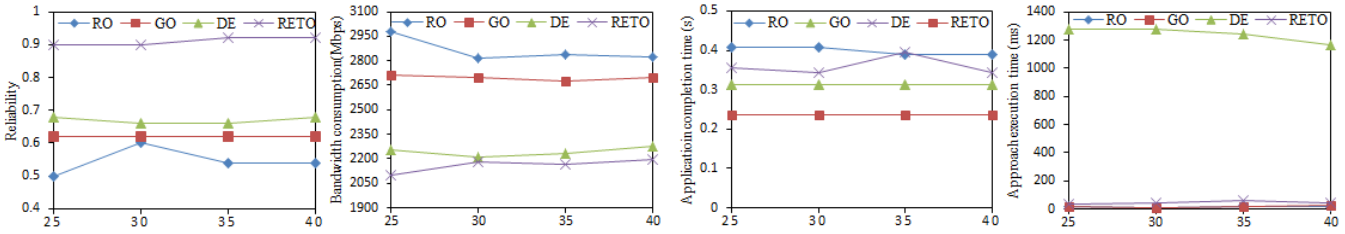
After the above steps are completed, both the top and 2- level tasks of the directed acyclic graph are offloaded onto the edge servers. Next, other tasks of the directed acyclic graph are also in turn offloaded to the edge servers. Once the tasks of the previous level are completed, these tasks are then seen as sending tasks and send their results to the lower level tasks. Fifth, we sort the tasks of the previous level in ascending order of their completion time. Last, we sort the receiving tasks of each task of the previous 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 G 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 G IoT applications in ascending order via their deadlines to first offload the time-critical IoT applications. The loop in lines 3-22 obtains the near-optimal offloading scheme of all tasks of G IoT applications. Lines 4-5 obtain the number of structure layers A_i of each IoT application, and divide the task set Q_i to A_i groups $TaskGroup[l], l \in \{1,2,...,A_i\}$. The loop in lines 6-9 selects the container or virtual machine j for task q of the set $TaskGroup[2]$ in descending order of instruction length to minimize T_{iqj}. The loop in lines 10-13 selects the container or virtual machine j for task q of the set $TaskGroup[1]$ in descending order of instruction length to minimize ω_q. Lines 14-21 select the container or virtual machine j for task q of the set $TaskGroup[l]$ in descending order of instruction length to minimize ω_q, when the number of structure layers l of the directed acyclic graph is greater than or equal to 3. The loop in lines 23-31 obtains the value of reliability level, bandwidth consumption, and total completion time of G IoT applications.

6 Performance Evaluation

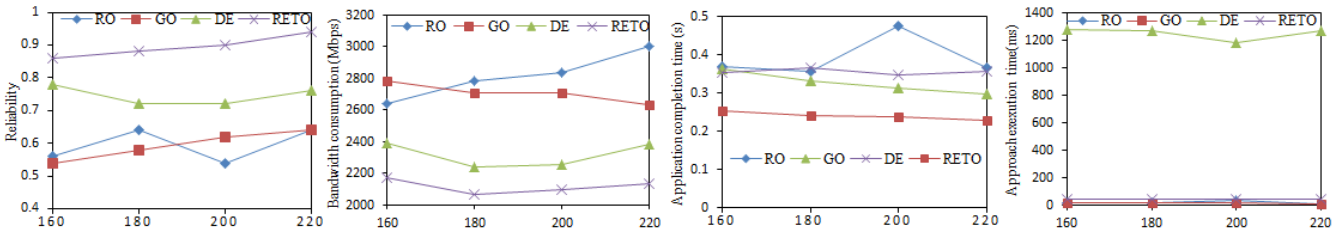
Using extended CloudSim simulator [27] and iFogSim simulator [28], we created a reliability-aware mobile edge computing simulation environment consisting of 25 edge clouds, 127 edge servers, and 200 virtual machines. These edge clouds are uniformly distributed in a 5G smart city network scenario and interconnected with a full mess topology [8]. 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 randomly selected from the set [4, 6]; multiple IoT devices exploit wireless access network to communicate with the base station. The configuration information for each edge server can be randomly selected from the set {HP ProLiant G4, HP ProLiant G5} [29]. The processing capacity and memory capacity for each virtual machine can be randomly selected 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} [29]. The disk capacity and bandwidth requirement for each virtual machine is 1GB and randomly selected from the set [10, 50] Mbps, respectively.

The foregoing parameter information is mainly used to configure the environment in which tasks can be offloaded. Next, we will set some parameters of the virtual machines for the task offloading according to the size of the virtual machines. The processing capacity of each virtual machine is randomly selected from the set [0.5×10^6, 10^6] instructions per second [30]; the memory capacity of each virtual machine is randomly selected from the set [2×10^7, 4×10^7] instructions; the failure rate and recovery rate of each virtual machine are randomly selected from the set [1%, 5%] [18]. 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 split



(a) Impact on the reliability level (b) Impact on bandwidth consumption (c) Impact on application completion time (d) Impact on approach execution time

Fig. 3. The effect of number of the base stations. The value of reliability level, bandwidth consumption, application completion time, and approach execution time is obtained by executing 50 IoT applications. As the number of base stations increased by step size 5 from 25 to 40, their value is not almost varied. Moreover, RETO has the moderate approach execution time, the maximum reliability level, the minimum bandwidth consumption, and the minimum application completion time of all approaches.



(a) Impact on the reliability level (b) Impact on bandwidth consumption (c) Impact on application completion time (d) Impact on approach execution time

Fig. 4. The effect of number of the virtual machines. The value of reliability level, bandwidth consumption, application completion time, and approach execution time is obtained by executing 50 IoT applications. As the number of virtual machines increased by step size 20 from 160 to 220, the application completion time decreases, the reliability level increases, the bandwidth consumption does not almost vary. Moreover, RETO has the maximum reliability level, the minimum bandwidth consumption, the minimum application completion time, and the moderate approach execution time of all approaches.

into multiple collaborative tasks offloaded to the edge clouds. The length of each task is randomly selected from the set [500, 5000] instructions [18]. If one task is a sender, the amount of data it sends is randomly selected from the set [1, 2] Mb. The deadline requirement of each IoT application is randomly selected from the set [5, 9] ms. The positive tunable factors α, θ_1, and θ_2 are set to 0.0037, 1, and 2000, respectively.

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

- ***Random Offloading (RO)***: Randomly selects one virtual machine to accommodate each task of the IoT application from top to bottom along its directed acyclic graph when multiple virtual machines satisfy the constraints.
- ***Greedy Offloading (GO):*** Selects the virtual machine with minimum value of the formulation (6) to accommodate each task of the IoT application from top to bottom along its directed acyclic graph when multiple virtual machines satisfy the constraints.
- ***Differential Evolution (DE):*** Selects the virtual machine via the differential evolution algorithm [31] to accommodate each task of the IoT application from top to bottom along its directed acyclic graph while multiple virtual machines satisfy the constraints.

Next, we analyze the performance of the RETO by comparing it with RO, GO, and DE in terms of the reliability level, the bandwidth consumption, and the application completion time while offloading G IoT applications. Meanwhile, we also analyze the impact of experimental parameters including the number of base stations (as shown in Fig. 3), the number of the virtual machines (as shown in Fig. 4), the number of IoT applications (as shown in Fig. 5), and the deadlines of IoT applications (as shown in Fig. 6) on these performance indicators.

6.1 Impact of number of the base stations

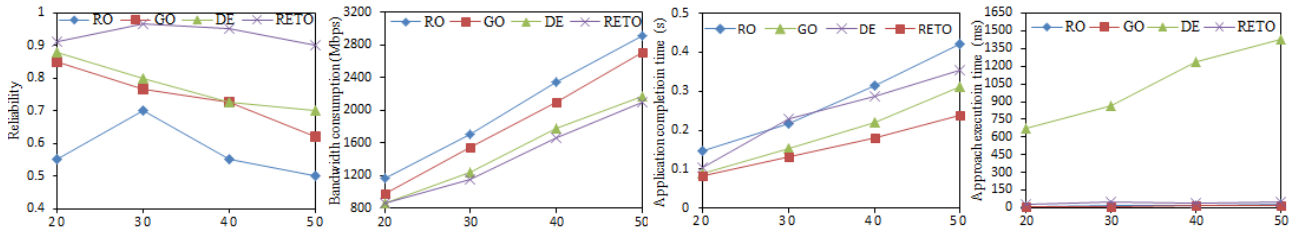
As shown in Fig. 3, as the number of the base stations increased by step size 5 from 25 to 40, the reliability level, the bandwidth consumption, the application completion time, and the approach execution time of each approach did not almost vary. That is due to that although the increase of the number of base stations increases the number of edge servers, the number of virtual machines and IoT applications does not change, and these virtual machines are still randomly assigned to these edge servers. Since DE is an intelligent optimization algorithm and take massive time to iterate, its approach execution time has the maximum value of all approaches. Moreover, RETO has the maximum reliability level, the minimum bandwidth consumption, the minimum application completion time, and the moderate approach execution time of all approaches.

6.2 Impact of number of the virtual machines

As shown in Fig. 4, as the number of the virtual machines increased by step size 20 from 160 to 220, the application completion time decreases, the reliability level increases; moreover, the bandwidth consumption does not almost vary. This is because the collaborative tasks of IoT applications are offloaded to the virtual machines on the premise that these virtual machines are randomly assigned to the edge servers. That is, the increase of the number of virtual machines can increase the probability of choosing a virtual machine with the lower failure rate, the higher processing capacity, or the higher recovery rate, and then decreases the sum of the process time or recovery time for each task. However, since the virtual machines are randomly assigned to the edge servers, the bandwidth consumption and communication time of these tasks do not almost affected. No matter how the number of virtual machines changes, RETO has the maximum reliability level, the minimum bandwidth consumption, the minimum application completion time, and the moderate approach execution time of all approaches.

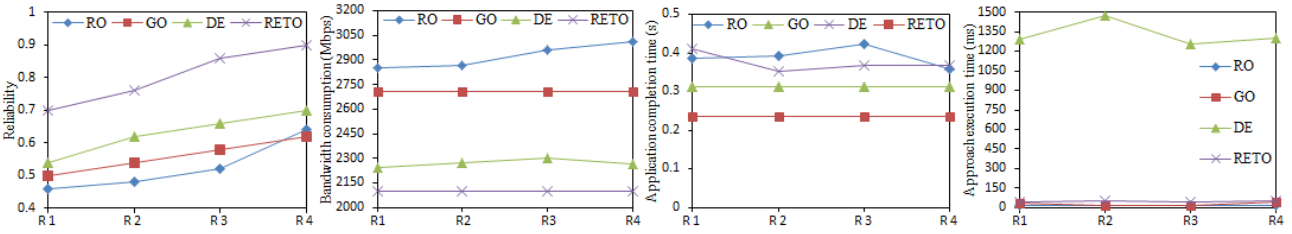
6.3 Impact of number of the IoT applications

As shown in Fig. 5, as the number of IoT applications increased by step 10 from 20 to 50, the reliability level decreases, the bandwidth consumption and the application completion time both increase. Except for the increase in the approach execution time of the DE, other approaches do not almost vary. That is due to that the increase in the number of IoT applications increases the number of collaborative tasks, which need to be offloaded onto the virtual machines with different processing capacities, failure rates, and recovery rates. Therefore, the ap-



(a) Impact on the reliability level (b) Impact on bandwidth consumption (c) Impact on application completion time (d) Impact on approach execution time

Fig. 5. The effect of number of the IoT applications. The value of reliability level, bandwidth consumption, application completion time, and approach execution time is obtained by executing different number of IoT applications. As the number of IoT applications increased by step size 10 from 20 to 50, the reliability level decreases, the bandwidth consumption and the application completion time both increase, except for the increase in the approach execution time of the DE, other approaches do not almost vary. Moreover, RETO has the maximum reliability level, the minimum bandwidth consumption, the minimum application completion time, and the moderate approach execution time of all approaches.



(a) Impact on the reliability level (b) Impact on bandwidth consumption (c) Impact on application completion time (d) Impact on approach execution time

Fig. 6. The effect of deadlines of the IoT applications. The value of reliability level, bandwidth consumption, application completion time, and approach execution time is obtained by executing 50 IoT applications. R1, R2, R3, and R4 represent four ranges of deadlines of IoT applications, i.e., [5, 6], [5, 7], [5, 8], and [5, 9], respectively. As the range increases from R1 to R4, the reliability level increases, the bandwidth consumption, the application completion time, and the approach execution time do not almost vary. Moreover, RETO has the maximum reliability level, the minimum bandwidth consumption, the minimum application completion time, and the moderate approach execution time of all approaches.

plication completion time and the bandwidth consumption both increase in varying degrees. Furthermore, the reliability level of all approaches is reduced accordingly. Since the number of tasks determined the chromosome length of DE, its approach execution time increases with its number. Moreover, RETO has the maximum reliability level, the minimum bandwidth consumption, the minimum application completion time, and the moderate approach execution time of all approaches.

6.4 Impact of deadlines of the IoT applications

As shown in Fig. 6, as the range of deadlines of the IoT applications increased from R1 to R4, the reliability level increases, the bandwidth consumption, the application completion time, and the approach execution time do not almost vary. Note that R1, R2, R3, and R4 represent four ranges of deadlines of IoT applications, i.e., [5, 6], [5, 7], [5, 8], and [5, 9], respectively. That is due to that the deadline of IoT application is only exploited to compute the reliability level of G IoT applications via the formulation (8), the increase of deadline of IoT application can prevent from the QoS violations and has no effect on the bandwidth consumption, the application completion time, and the 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 is increased. That is, the reliability level of G IoT applications also increases with the increase of the range of deadlines of the IoT applications, i.e., from R1 to R4. Moreover, RETO has the maximum reliability level, the minimum bandwidth consumption, the minimum application completion time, and the moderate approach execution time of all approaches.

7 Conclusions and Further Work

In this paper, we had investigated the collaborative task offloading problem with the consideration of the container or virtual machine failure rates and recovery rates. We had 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 had proposed and transformed a multi-objective optimization problem into a single objective optimization problem for integer linear programming. In order to improve computational efficiency, we had introduced an approach to acquire a near-optimal solution with lower time complexity. The results of simulation experiments had demonstrated that our proposed approach provided a near-optimal solutions and had better performance compared with other approaches.

In our future work, we will take the remote cloud into account our experimental environment, and then study the above optimization problem based on the real dataset.

Acknowledgment

This work is partially is supported by the Key Science and Technology Program of Henan Province (202102210152), Henan Province Higher Education Teaching Reform Research and Practice Project (2019SJGLX386), and the Research and Cultivation Fund Project of Anyang

Normal University (AYNUKPY-2019-24).

REFERENCES

[1] B. Varghese and R. Buyya, "Next Generation Cloud Computing: New Trends and Research Directions," Future Generation Computer Systems, 2018, 79: 849-861.

[2] P. Schulz, M. Matthe, H. Klessig, M.Simsek, G. Fettweis, J. Ansari, S. A. Ashraf, B. Almeroth, J.s Voigt, I. Riedel, A. Puschmann, A. Mitschelethiel, M. Muller, T. Elste, and M. Windisch, "Latency Critical IoT Applications in 5G: Perspective on the Design of Radio Interface and Network Architecture," IEEE Communications Magazine, 2017, 55(2): 70-78.

[3] P. Hu, S. Dhelim, H. Ning, and T. Qiu, "Survey on Fog Computing: Architecture, Key Technologies, Applications and Open Issues," Journal of Network & Computer Applications, 2017, 98(nov.): 27-42.

[4] C. Hong and B. Varghese, "Resource Management in FogEdge Computing A Survey on Architectures, Infrastructure, and Algorithms, "ACM Computing Serveys, 2019, 52(5):97:1-97:37.

[5] H. Zhu and C. Huang, "Availability-Aware Mobile Edge Application Placement in 5G Networks," Proceedings of IEEE Global Communications Conference (Globecom 2017), pp. 1-6, 2017.

[6] B. Yang, F. Tan, Y. Dai, and S. Guo, "Performance Evaluation of Cloud Service Considering Fault Recovery," Proceedings of IEEE International Conference on Cloud Computing (Cloud 2009), pp. 571-576, 2009.

[7] S. Garg, S. Versteeg, and R. Buyya, "A framework for ranking of cloud computing services," Future Generation Computer Systems, 2013, 29(4): 1012-1023.

[8] J. Oueis, E. Calvanese-Strinati, A. De Domenico, and S. Barbarossa, "On the impact of backhaul network on distributed cloud computing," Proceedings of IEEE Wireless Communications and Networking Conference Workshops (WCNCW 2014), pp. 12-17, 2014.

[9] A. Aral and I. Brandic, "Quality of Service Channelling for Latency Sensitive Edge Applications," Proceedings of IEEE International Conference on Edge Computing (EDGE 2017), pp. 166-173, 2017.

[10] M. Soualhia, C. Fu, and F. Khomh, "Infrastructure fault detection and prediction in edge cloud environments," Proceedings of the 4th ACM/IEEE Symposium on Edge Computing (SEC 2019), pp. 222-235, 2019.

[11] A. Maia, Y. Ghamri-Doudane, D. Vieira, and M. de Castro, "Optimized Placement of Scalable IoT Services in Edge Computing," Proceedings of IFIP/IEEE Symposium on Integrated Network and Service Management (IM 2019), pp. 189-197, 2019.

[12] A. Maia, Y. Ghamri-Doudane, D. Vieira, and M. de Castro, "A Multi-Objective Service Placement and Load Distribution in Edge Computing," Proceedings of IEEE Global Communications Conference (Globecom 2019), 2019, online publishing.

[13] H. Zhao, S. Deng, Z. Liu, J. Yin, and S. Dustdar, "Distributed Redundancy Scheduling for Microservice-based Applications at the Edge," IEEE Transactions on Services Computing, 2020, online publishing.

[14] Y. Liu, S. Wang, Q. Zhao, S. Du, A. Zhou, X. Ma, and F. Yang, "Dependency-Aware Task Scheduling in Vehicular Edge Computing," IEEE Internet of Things Journal, 2020, online publishing.

[15] L. Zhao and J. Liu, "Optimal Placement of Virtual Machines for Supporting Multiple Applications in Mobile Edge Networks," IEEE Transactions on Vehicular Technology, 2018, 67(7):6533-6545.

[16] M. Goudarzi, H. Wu, M. Palaniswami, and R. Buyya, "An Application Placement Technique for Concurrent IoT Applications in Edge and Fog Computing Environments," IEEE Transactions on Mobile Computing, 2020, DOI:10.1109/TMC.2020.2967041.

[17] L. Liu, H. Tan, S. Jiang, Z. Han, X. Li, and H. Huang, "Dependent task placement and scheduling with function configuration in edge computing," Proceedings of IEEE international Workshop on Quality of Service (IWQoS 2019), 20:1-20:10, 2019.

[18] J. Yao and N. Ansari, "Fog Resource Provisioning in Reliability-Aware IoT Networks," IEEE Internet of Things Journal, 2019, 6(5):8262-8269.

[19] M. Yousefi, S. Ghiassi, B. Hashemi, and M. Goudarzi, "Workload Scheduling on heterogeneous Mobile Edge Cloud in 5G networks to Minimize SLA Violation," arXiv: Distributed, Parallel, and Cluster Computing, 2020.

[20] B. Hu, J. Chen, and F. Li, "A dynamic service allocation algorithm in mobile edge computing," Proceedings of IEEE International Conference on Information and Communication Technology Convergence (ICTC 2017), pp.104-109, 2017.

[21] V. Farhadi, F. Mehmeti, and T. Porta, "Service Placement and Request Scheduling for Data-intensive Applications in Edge Clouds," Proceedings of IEEE Conference on Computer Communications (INFOCOM 2019), pp. 1279-1287, 2019.

[22] 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 of Things Journal, 2020, online publishing.

[23] 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," Proceedings of IEEE Conference on Computer Communications (INFOCOM 2019), pp. 2287-2295, 2019.

[24] M. Bari, R. Boutaba, R. Esteves, L. Granville, M. Podlesny, G. Rabbani, Q. Zhang, and M. Zhani, "Data Center Network Virtualization: A Survey," IEEE Communications Surveys and Tutorials, 2013, 15(2): 909-928.

[25] K. Deb, "Multi-Objective Optimization," in Search Methodologies. Boston, MA, USA: Springer, 2014, pp. 403-449.

[26] J. Yao and A. Nirwan, "QoS-aware Joint BBU-RRH Mapping and User Association in Cloud-RANs," IEEE Transactions on Green Communications & Networking, 2018, 2(4): 881-889.

[27] J. Liu, S. Wang, A. Zhou, S. Kumar, F. Yang, and R. Buyya, "Using Proactive Fault-tolerance Approach to Enhance Cloud Service Reliability," IEEE Transactions on Cloud Computing, 2018, 6(4):1191-1202.

[28] H. Gupta, A. Vahid Dastjerdi, S. 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," Software: Practice and Experience, 2017, 47(9): 1275–1296.

[29] 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 and Computation: Practice and Experience, 2012, 24(13): 1397-1420.

[30] A. Miettinen and J. Nurminen, "Energy efficiency of mobile clients in cloud computing," Proceedings of the 2nd USENIX Conference on Hot Topics Cloud Computing (HotCloud 2010), p. 4, 2010.

[31] R. Storn and K. Price, "Differential Evolution – A Simple and Efficient Heuristic for global Optimization over Continuous Spaces," Journal of Global Optimization, 1997, 11: 341-359.

Jialei Liu is an Assistant Professor at School of Software Engineering, Anyang Normal University, China. He received PhD degree in computer science and technology from Beijing University of Posts and Telecommunications (BUPT) in 2018 and ME in computer science and technology from Henan Polytechnic University. His major research interests include cloud computing and mobile edge computing.

Shangguang Wang received his Ph.D. degree at Beijing University of Posts and Telecommunications in 2011. He is currently a professor and deputy director at the State Key Laboratory of Networking and Switching Technology, BUPT. He has published more than 100 papers, and played key roles such as general chair or PC chair for many international conferences. His research interests include edge computing, service computing, and cloud computing. He is a senior member of the IEEE, and the Editor-in-Chief of the International Journal of Web Science.

Lianyong Qi received his PhD degree in Department of Computer Science and Technology from Nanjing University, China, in 2011. In 2010, he visited the Department of Information and Communication Technology, Swinburne University of Technology, Australia. Now, he is a full professor of the School of Information Science and Engineering, Qufu Normal University, China. His research interests include big data and recommender systems. He has published over 90 research papers (first author or corresponding author) in international journals and conferences.

Rajkumar Buyya is a Redmond Barry Distinguished Professor and Director of the Cloud Computing and Distributed Systems (CLOUDS) Laboratory at the University of Melbourne, Australia. He has authored over 750 publications and seven text books including ”Mastering Cloud Computing” published by McGraw Hill, China Machine Press, and Morgan Kaufmann for Indian, Chinese and international markets respectively. He is one of the highly cited authors in computer science and software engineering worldwide (h-index=134, g-index=304, 100,800+ citations). He served as the founding Editor-in-Chief of the IEEE Transactions on Cloud Computing. He is currently serving as 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