Robust Task Offloading in Dynamic Edge Computing

Haibo Wang, *Student Member, IEEE*, Hongli Xu, *Member, IEEE*, He Huang, *Member, IEEE*, Min Chen, and Shigang Chen, *Fellow, IEEE*

Abstract—Multi-access edge computing achieves better application responsiveness by offloading tasks from end devices to edge servers installed at the vicinity. Practical scenarios, such as post-disaster rescuing and battlefield monitoring, make it attractive to use end devices themselves as edge servers. This, however, introduces a new challenge: Due to mobility and power limitation, the set of edge servers becomes dynamic. As some servers fail, the tasks that run on them will also fail. This paper introduces a new dynamic edge computing model and conducts the first study on robust task offloading which is tolerant to h server failures. We propose online primal-dual algorithms that offload tasks as they arrive. We evaluate the performance of our robust task offloading solutions through extensive simulations based on real task sets. The results show that our proposed solutions can well handle edge dynamics and achieve near optimal throughput (above 95 percent) compared to the optimal offline benchmark algorithm.

Index Terms—Task offloading, robust, dynamic edge computing, primal-dual, approximation

1 Introduction

The popularity of Internet-of-Things (IoT) is driving the development of diverse applications, e.g., interactive online gaming [1], face recognition [2], 3D modeling [3], [4], and VR/AR [5]. Today, tremendous data are transmitted to various cloud platforms across the Internet, which poses great challenges on transmission bandwidth and delay for real-time applications [6]. As a new solution, multi-access edge computing (MEC) [7], [8], which is also known as mobile edge computing [9], [10], has been proposed to offload tasks from end devices to edge servers installed at the vicinity for better responsiveness, lower latency and enhanced bandwidth utilization [1], [11].

Much prior work focuses on maximizing system throughput of MEC. Game-theoretic approaches are used for resource allocation [12], [13], [14] under the assumption that all tasks have the same resource demand. In [15], a Vickrey-Clarke-Groves based auction algorithm is designed to maximize system throughput under more general settings where task demands can be different. Online task offloading is another topic of great interest. An online algorithm based on Lyapunov optimization is proposed in [16] to efficiently utilize resources in MEC under the assumption that task arrival can be modeled as a Bernoulli process. An online task offloading and scheduling algorithm is designed in [17] to maximize system throughput of MEC. Additional literatures on task offloading for throughput maximization can be found in [18], [19], [20].

Most prior research on MEC adopts a model of designated edge devices/machines, e.g., using WiFi access points and base stations for edge servers [3], [21]. These devices can be made more powerful in their processing capability. However, under some situations, such as disaster rescuing and battlefield monitoring, there may not be appropriate designated devices acting as edge servers [22], [23]. Meanwhile, with the fast increasing ability and quantity of IoT devices such as smart phones and unmanned aerial vehicles (UAV), there often exist abundant resources on the end devices [24].

Researchers have proposed to use end devices themselves as edge servers [22], [23]. However, there is a key challenge that has not yet been studied before. End devices are not as reliable as traditional designated edge servers. Due to mobility, existing devices may depart from the system, while new devices may join in, with some of them having spare capacity to serve as edge servers. End devices may be powered off temporarily. In such situations, the tasks that run on these *failed* servers will be aborted [22], causing unpredictability and disruption on the applications that rely on task offloading. For instance, some applications (e.g., augmented reality) include multiple tasks with dependency [5], [25]. The output of a task (e.g., positioning) may be the indispensable input of subsequent tasks (e.g., rendering and mapping) [26]. If the execution of task positioning is delayed, the dependent tasks will be postponed as well.

- *Haibo Wang is with the School of Computer Science and Technology, University of Science and Technology of China, Hefei, Anhui 230027, China, and also with the Department of Computer and Information Science and Technology, University of Florida, Gainesville, FL 32611 USA. E-mail: wanghaibo@ufl.edu.*
- *Hongli Xu and Min Chen are with the School of Computer Science and Technology, University of Science and Technology of China, Hefei, Anhui 230027, China, and also with the Suzhou Institute for Advanced Study, University of Science and Technology of China, Suzhou, Jiangsu 215123, China. E-mail: xuhongli@ustc.edu.cn, mchen330@mail.ustc.edu.cn.*
- *He Huang is with the School of Computer Science and Technology, Soochow University, Suzhou, Jiangsu 215006, China. E-mail: huangh@suda.edu.cn.*
- *Shigang Chen is with the Department of Computer and Information Science and Technology, University of Florida, Gainesville, FL 32611 USA. E-mail: sgchen@cise.ufl.edu.*

Manuscript received 15 October 2020; revised 8 February 2021; accepted 18 March 2021. Date of publication 24 March 2021; date of current version 5 December 2022.
(Corresponding author: Hongli Xu.)
Digital Object Identifier no. 10.1109/TMC.2021.3068748

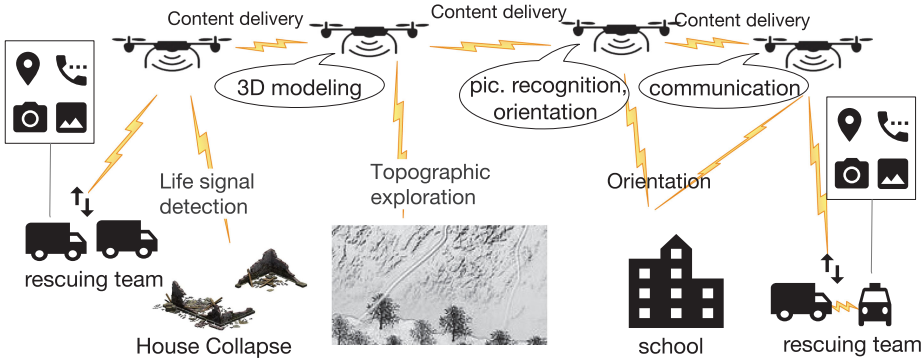


Fig. 1. A practical scenario for task execution by peer UAVs: rescuing after earthquake.

As a result, users cannot experience the visual effects in real time and consequently the quality of service is degraded.

This paper introduces a new *dynamic edge computing* (DEC) model, which is different from traditional MEC that ignores edge dynamics. Under this model, we study a new problem of robust task offloading, which guarantees the execution of tasks when up to h edge servers fail, where h is a user-defined parameter. The main contributions of this paper are summarized as follows: We propose a robust batch task offloading scheme to overcome the edge dynamics in DEC. We show the NP-hardness of this problem and demonstrate its inapproximability factor of $[1-\epsilon, O(\log n)]$, where ϵ is an arbitrarily positive value with $\epsilon \in (0,1)$ and n is the number of edge servers in DEC. We then propose online primal-dual algorithms that offload tasks as they arrive and analyze their competitive ratios of $[1-\epsilon, O(\log n + \log(1/\epsilon))]$. We evaluate the performance of our robust task offloading solutions through extensive simulations based on real task sets. The results show that our proposed solutions can well handle edge dynamics and achieve near optimal throughput (above 95 percent) compared to the optimal offline benchmark algorithm.

The rest of the paper is organized as follows. Section 2 presents the system model and problem statement. Section 3 formulates the robust task offloading problem, which is resistant to h edge server failures. In Section 4, we describe the proposed algorithms and analyze the competitive performance. Section 5 presents the simulation evaluation and practicability evaluation. The related works are presented in Section 6. Finally, we conclude this paper in Section 7.

2 Preliminary

2.1 Use Case

We introduce a typical use case of unmanned aerial vehicles (UAVs) to demonstrate that IoT devices can also act as servers for task execution.

Fig. 1 shows the post-earthquake scenarios where several rescuing teams are trekking in the earthquake region. Their duties are rescuing people, acquiring the disaster situation, and topographic exploration. After the earthquake, they need to acquire the disaster situations, locate some important sites like schools and hospitals, build real time 3D topography of the earthquake, detect the life signals to find the buried people faster, 3D modeling to rescue buried people and setup communications with each other. However, all these tasks cannot be executed by one UAV due to limited computing ability. Different UAVs may be responsible for one or several tasks [27]. As shown in the figure, one UAV executes 3D modeling while another UAV executes picture recognition. Since some tasks are very urgent and important, such as 3D modeling rescue buried people, we should guarantee that these tasks should be finished in real time.

2.2 Task Transferring Over Edge Dynamics

Recall that in the previous subsection, it is imperative to assign the failed task to an edge server instantly once the task fails on the original edge server. Traditionally, once a task fails, we should perform the task offloading algorithm in order to find an edge server to execute the failed task. However, there may be currently no enough residual resource on any edge server to execute the failed task. This is a dilemma as some practical applications require the failed task to be seamlessly transferred to other edge servers for low service delay. To assure that any task will be assigned to a new edge server once it fails, we propose robust task offloading. Specifically, a disrupted task should be assigned to the new edge server with adequate resource capacity (the new edge server assignment is part of the solution of robust task offloading). The connection to the new edge server is supported, which will be described in Section 5.7. This paper assumes that in the applications such as the use case in Section 2.1, the delay of setting up a connection to the new edge server for task transferring is small, which will be validated by our practical tests in Section 5.7 (for WiFi) and also validated by other work [28] ($<$ 640 ms for bluetooth) [29] ($\leq$200 ms for Zigbee). Based on the assumption, this paper does not consider the delay as a major factor but consider how to improve the throughput of the system.

2.3 DEC System Model

A *dynamic edge computing* (DEC) system consists of *end devices* that generate tasks, *edge servers* that provide resources for task execution, and a *manager* that takes requests from end devices and assigns their tasks to edge servers based on resource availability, where some end devices may serve as edge servers. Because they can be mobile, the set of edge servers is dynamic.

The manager may run at a dedicated machine or one of the edge servers that do not leave the system. When new edge servers enter the system or existing ones change their

capacities, they should report their information to the manager. When existing servers depart from the system, they should also inform the manager. For abrupt failure such as forced power-off, the manager needs to periodically ping the servers for their liveness. When a task completes, the edge server needs to inform the manager for resource release.

2.4 Communication Model

We introduce the communication model and give the uplink data rate when end devices offload tasks to edge servers. Let $H_{j,k}$ be the channel gain between the end device that generates task t_j and the edge server p_k. Assuming that the end device does not move much during task offloading, we can regard $H_{j,k}$ as a constant. Let S_j be the transmission power of the end device that generates task t_j. Accordingly to [30], [31], the uplink data rate from the end device that generates task t_j to the edge server p_k can be obtained as $B\log_2(1+\frac{S_jH_{j,k}}{\sigma^2})$, where σ^2 denotes the noise power, and B is the channel bandwidth.

In this paper, we assume that the downlink transmission delay and packet loss can be neglected as the data size after the task execution is usually very small and downlink rate is usually higher than the uplink [31]. In terms of the uplink transmission delay, there are two types in this paper. One is that the end devices should first request the manager that run the proposed robust offloading algorithms to obtain the offloading solution. In this case, each end device only needs to send the information about the task, e.g, the resource consumption of the task, rather than the task's input data of the task to the manager. Therefore, size of the uplink transmission data is very small and the transmission delay can be neglected. The other delay is for transmitting the task's input data to the assigned edge server. This may be large as some tasks possess large-size input data. This paper puts this delay aside first for the following reasons. We focus on robust task offloading where the principal concern is whether the task can be executed on edge servers or not immediately even if some edge servers fail. As the first solution to robust task offloading in MEC (see description in Section 6), our solution mainly considers the resource requirement of the task, including computing resource or other resources, allowing the adequate residual resources for tasks originally executed on failed edge servers. In addition, we discuss how to introduce the delay of transmitting task's input data to the edge server when running the robust task offloading algorithms in Section 4.7.

2.5 Problem Statement

Considering an arbitrary DEC system, let its set of m indivisible tasks be $\mathcal{T}=\{t_1,t_2,\ldots,t_m\}$, which are generated by end devices. The resource demand of task t_i is denoted as r_i, $1\leq i\leq m$, which can be computing resources, memory resource, etc., based on the application need (we will conduct evaluation under different types of resource demands). Let the set of n edge servers be $\mathcal{P}=\{p_1,p_2,\ldots,p_n\}$, where the residual capacity of p_j is denoted as c_j, $1\leq j\leq n$. Considering that an end device may only be able to connect to the edge servers in its proximity, we denote $\mathcal{P}_i\subset\mathcal{P}$ as the set of edge servers that the end device generating t_i can connect to. The *system throughput* of DEC is defined as the sum of resource demand of the tasks that are successfully assigned to edge servers for immediate execution. The problem of *robust task offloading* is to assign $\mathcal{T}$ to $\mathcal{P}$ for maximum system throughput, subject to *the capacity constraint* such that the tasks assigned to any edge server should not exceed the server's capacity and *the robustness constraint* such that any assigned task will be guaranteed for its execution when up to h edge servers fail (due to reasons such as moving out of the system or being powered off), where h is a configurable system parameter. Some important notations are listed in Table 1.

TABLE 1
Notations

m	number of tasks
n	number of edge servers
h	number of edge servers that fail
$\mathcal{T}$	set of tasks
f,e	flow label and element identifier
$t_i,1\leq i\leq m$	task
$r_i,1\leq i\leq m$	resource demand of task t_i
$\mathcal{P}$	set of edge servers
$p_j,1\leq j\leq n$	edge server
$c_j,1\leq j\leq n$	residual capacity of p_j
$\mathcal{P}_i$	set of edge servers t_i can be assigned to

Clearly, when any h edge servers fail, their tasks will have to be transferred to other edge servers. One might think that we only need to ensure the total residual capacity of any $(n-h)$ servers exceeds the total task demand on the remaining h servers. It is not the case because tasks are indivisible. Not only must we ensure the total residual capacity to be large enough, but also the transferred tasks have to individually fit in the servers still in the system.

In the sequel, we will first formulate a periodic offline batch assignment strategy where the manager will schedule tasks at the beginning of each period. We will then move to an online assignment strategy where tasks are scheduled upon their arrivals.

3 Robust Batch Task Offloading

To conquer the edge dynamics, it is required that all the tasks can be executed correctly even if h edge servers fail. We take $h=1$ as an instance. For an arbitrary task t_i, assume that it is originally offloaded to edge server $p_{j_0}\in\mathcal{P}_i$. To overcome the failure of one edge server, there must be adequate residual resource capacity to accommodate task t_i. In other words, a backup edge server is required. Therefore, our robust task offloading scheme should determine an edge server pair $\{p_{j_0},p_{j_1}\}$ for task t_i with $p_{j_0},p_{j_1}\in\mathcal{P}_i$. Likewise, to overcome h edge server failures, we need to guarantee that there is adequate resource capacity on p_{j_h} even if prior h edge server $\{p_{j_0},p_{j_1},\ldots,p_{j_{h-1}}\}$ (with $p_{j_0},p_{j_1},\ldots,p_{j_{h-1}}\in\mathcal{P}_i$ and $h+1\leq|\mathcal{P}_i|$) fail simultaneously. Therefore, the offloading solution for task t_i includes $h+1$ edge servers, which is called *$(h+1)$-server permutation* for task t_i. For ease of expression,

we denote $\mathcal{C} = \{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_{|\mathcal{C}|}\}$ as the set of all the $(h+1)$-server permutations. Since all the elements in each $(h+1)$-server permutation should be mutually different, there are P_n^{h+1} possible $(h+1)$-server permutations, i.e., $|\mathcal{C}| = P_n^{h+1}$. Note that each $(h+1)$-server permutation $\mathcal{C}_j \in \mathcal{C}$ is denoted as $\mathcal{C}_j = \{p_{j_0}, p_{j_1}, \ldots, p_{j_h}\}$.

The binary variable $y_i^j \in \{0,1\}$ means whether the $(h+1)$-server permutation $\mathcal{C}_j = \{p_{j_0}, p_{j_1}, \ldots, p_{j_h}\}$ is the offloading solution for task t_i ($y_i^j = 1$) or not. Since all tasks are indivisible, each task t_i can only be assigned to a single $(h+1)$-server permutation $\mathcal{C}_j$, as shown in Eq. (1a). Let *constant* s_i^j represent whether there exists an edge server p_k that satisfies $p_k \in \mathcal{C}_j$ and $p_k \neq \mathcal{P}_i$ ($s_i^j = 0$) or not (s_i^j=1)). $s_i^j = 0$ means $(h+1)$-server permutation $\mathcal{C}_j$ should not be considered as at least one of its edge server cannot be connected by the end device generating task t_i. Accordingly, the binary $y_i^j = 0$ must hold, as shown in Eq. (1b). Because task t_i will be executed on the first server in $\mathcal{C}_j$ when there is no failure, we calculate the initial load on edge server p_{j_0} as $\sum_{t_i \in \mathcal{T}} \sum_{\mathcal{C}_j = \{p_{j_0}, \ldots, p_{j_h}\} \in \mathcal{C}} y_i^j \cdot r_i = \lambda_{j_0} \cdot c_{j_0}$, where $\lambda_{j_0} (\leq 1)$ denotes the *load ratio* of server j_0, and c_{j_0} is its capacity. Obviously, the load ratio of each edge server should not exceed 1 to assure no violation of the capacity constraint, as shown in Eq. (1c).

Our robust task offloading scheme should be resistant to h server failures. For simplicity, these failed servers are denoted as $\mathcal{F}_k = \{p_{k_1}, p_{k_2}., , , .p_{k_h}\}$. Since each edge server may fail due to mobility and power limitation, these h failed servers are arbitrarily selected. The set of all possible cases are denoted by $\mathcal{F} = \{\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_{|\mathcal{F}|}\}$, with $|\mathcal{F}| = P_n^h$. Given an arbitrary $(h+1)$-server permutation $\mathcal{C}_j$ and set of failed servers $\mathcal{F}_k$, let the binary variable $z_{j,k}^l$ represent whether edge server p_l is the first available edge server in $\mathcal{C}_j$ or not. For instance, when $h = 2$, $\mathcal{C}_1 = \{p_0, p_1, p_2\}$ and $\mathcal{F}_2 = \{p_0, p_3\}$, we obtain $z_{1,2}^1 = 1$ for p_1 is the first server in $\mathcal{C}_1$ that doesn't exist in $\mathcal{F}_2$. Once the h edge servers $\{p_{k_1}, p_{k_2}, \ldots, p_{k_h}\}$ fail, the task t_i assigned with $(h+1)$-server permutation $\mathcal{C}_j$ will be re-offloaded to the edge server p_l if $z_{j,k}^l = 1$, and consequently, the load on edge server p_l will increase by r_i, called the *incremental load*. To guarantee that there are enough resource to accommodate the tasks originally executed on those failed edge servers, the incremental load on each edge server should not exceed its residual resource capacity in Eq. (1d). Our objective is to maximize the system throughput of DEC, i.e., $\max \sum_{j=1}^{n} \lambda_j \cdot c_j$. The robust task offloading with throughput maximization problem (RTO-TM) is formulated as follows:

$$\max \quad \sum\nolimits_{l=1}^{n} \lambda_l \cdot c_l$$

$$\begin{cases} \sum\limits_{\mathcal{C}_j = \{p_{j_0}, p_{j_1}, \ldots, p_{j_h}\} \in \mathcal{C}} y_i^j \leq 1 & \forall t_i \in \mathcal{T} \quad (1a) \\ y_i^j \leq s_i^j & \forall t_i, \mathcal{C}_j \quad (1b) \\ \sum\limits_{t_i \in \mathcal{T}} \sum\limits_{\mathcal{C}_j = \{p_{j_0}, \ldots, p_{j_h}\} \in \mathcal{C}} y_i^j \cdot r_i = \lambda_{j_0} \cdot c_{j_0} & \forall p_{j_0} \in \mathcal{P} \quad (1c) \\ \sum\limits_{t_i \in \mathcal{T}} \sum\limits_{\mathcal{C}_j \in \mathcal{C}} y_i^j \cdot z_{j,k}^l \cdot r_i \leq c_l (1 - \lambda_l) & \forall p_l, \mathcal{F}_k \quad (1d) \\ y_i^j \in \{0,1\} & \forall t_i, \mathcal{C}_j \quad (1e) \\ \lambda_j \leq 1 & \forall p_j \quad (1f) \end{cases}.$$

Lemma 1. *RTO-TM is an NP-hard problem.*

By considering a special case of RTO-TM in which there are only two edge servers with the same capacity, our problem can be reduced to a classical knapsack problem, which is NP-hard [32]. The detailed proof is deferred to Appendix in the supplementary file, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TMC.2021.3068748.

The solution for the RTO-TM problem is resistant to h edge server failures, with $h \leq \min_{i=1}^{m}\{|\mathcal{P}_i|\} - 1$. However, there is a contradiction between the robustness and effectiveness of the resource utilization for task offloading. On one hand, the larger h is, the more robust our solution can be. On the other hand, the larger h is, the less effectively the solution makes use of the resource on edge servers. Therefore, this paper studies RTO-TM for different values of h. Without confusion, RTO-TM-h represents the case for RTO-TM which is resistant to h edge server failures.

4 Robust Online Task Offloading

In this section, we first present an online algorithm for robust task assignment with $h = 1$. It assigns new tasks as they arrive and can tolerate one server failure. We then extend the solution to the case of $h = 2$ and further to the general case of other h values.

4.1 Online Algorithm for the RTO-TM-1 Problem

We first give the formulation of RTO-TM-1 based on the general framework in Eq. (1). For $h = 1$, our task offloading scheme should determine a 2-server permutation for each task. Without confusion, each 2-server permutation is also called an edge server pair, which contains an edge server for task execution and a backup edge server to provide enough residual resources in case that the offloaded edge server fails. We define a binary variable $x_i^{j,k}$ to represent whether task t_i is offloaded to edge server pair $\{p_j, p_k\}$ or not. Let the binary constant $\mathcal{I}_{j,k}$ represent whether two variables j and k are equal ($\mathcal{I}_{j,k} = 0$) or not ($\mathcal{I}_{j,k} = 1$). Here, we consider the overall load on edge server p_v once edge server p_u fails. Without failure, the load on the edge server p_v is

$$\sum\nolimits_{t_i} \sum_{p_j, p_k \in \mathcal{P}_i; p_j \neq p_k} x_i^{j,k} (1 - \mathcal{I}_{j,v}) r_i.$$

When edge server p_u fails, the incremental load on edge server p_v from edge server p_u is

$$\sum\nolimits_{t_i} \sum_{p_j, p_k \in \mathcal{P}_i; p_j \neq p_k} x_i^{j,k} (1 - \mathcal{I}_{j,u})(1 - \mathcal{I}_{k,v}) r_i.$$

Thus, when edge server p_u fails, the overall load on edge server p_v is

$$\sum\nolimits_{t_i} \sum\nolimits_{p_j, p_k \in \mathcal{P}_i; p_j \neq p_k} x_i^{j,k} \delta(j,k,u,v) r_i,$$

where $\delta(j,k,u,v) = (1 - \mathcal{I}_{j,v}) + (1 - \mathcal{I}_{j,u})(1 - \mathcal{I}_{k,v})$. The load should not exceed the capacity on p_v. Without confusion, $\delta(j,k,u,v)$ is abbreviated as δ. Note that p_u and p_v are arbitrarily selected and the constraint should be satisfied for

$\forall p_u, p_v \in \mathcal{P}$. Accordingly, the problem formulation can be rewritten as follows:

$$\max \quad \sum\nolimits_{t_i \in \mathcal{T}} \sum\nolimits_{p_j, p_k \in \mathcal{P}_i; p_j \neq p_k} x_i^{j,k} \cdot r_i$$

$$\begin{cases} \sum\limits_{p_j, p_k \in \mathcal{P}_i; p_j \neq p_k} x_i^{j,k} \leq 1 & \forall t_i \in \mathcal{T} \quad (2a) \\ \sum\limits_{t_i} \sum\limits_{p_j, p_k \in \mathcal{P}_i; p_j \neq p_k} x_i^{j,k} \cdot \delta \cdot r_i \leq c_v & \forall p_u, p_v, p_u \neq p_v \quad (2b) \\ x_i^{j,k} \in \{0, 1\} & \forall t_i; p_j, p_k, p_j \neq p_k \quad (2c) \end{cases}$$

The dual version of the linear program Eq. (2) can be obtained following the classical primal-dual method [33] and is exhibited as follow:

$$\min \quad \sum\nolimits_{t_i} \phi(t) + \sum\nolimits_{p_u, p_v} \chi(p_u, p_v) c_v$$

$$\begin{cases} \phi(t_i) + \sum\limits_{p_u, p_v, p_u \neq p_v} \delta \cdot r_i \cdot \chi(p_u, p_v) \geq r_i & \forall t_i; p_j, p_k \in \mathcal{P}_i; p_j \neq p_k \quad (3a) \\ \phi(t_i) \geq 0 \ \chi(p_u, p_v) \geq 0 & \forall t_i, p_u, p_v, p_u \neq p_v \quad (3b) \end{cases}.$$

We can rewrite the first constraint of Eq. (3) as

$$\phi(t_i) \geq r_i \left(1 - \sum_{p_u, p_v, p_u \neq p_v} \delta \cdot \chi(p_u, p_v)\right) \quad \forall t_i, p_j, p_k \in \mathcal{P}_i; p_j \neq p_k. \tag{4}$$

Algorithm 1. Primal-Dual Based Task Offloading Algorithm

1: $\varphi = B^*/\epsilon$
2: Initialize all the dual variables:
3: $\phi(t_i) \leftarrow 0 \quad \forall t_i$
4: $\chi(p_u, p_v) \leftarrow 0 \quad \forall p_u, p_v$
5: **for** each arrival task t_i **do**
6: $\quad \{p_{j^*}, p_{k^*}\} \leftarrow \text{argmin}_{p_j, p_k \in \mathcal{P}_i; p_j \neq p_k} V_{j,k}$
7: $\quad$ **if** $V_{j^*,k^*} < 1$ **then**
8: $\quad\quad$ Assign task t_i to edge server pair $[p_{j^*}, p_{k^*}]$
9: $\quad\quad \phi(t_i) \leftarrow r_i(1 - V_{j^*,k^*})$
10: $\quad\quad \forall p_u, p_v$: $\chi(p_u, p_v) \leftarrow \chi(p_u, p_v)[1 + \frac{\delta \cdot r_i}{c_v}] + \frac{\delta \cdot r_i}{n \cdot \varphi \cdot c_v}$
11: $\quad$ **else**
12: $\quad\quad$ Reject the task and set $\phi(t_i) \leftarrow 0$
13: $\quad$ **end if**
14: **end for**

For each arrival task, the algorithm needs to determine an edge server pair $\{p_j, p_k\}$ with $p_j, p_k \in \mathcal{P}_i; p_j \neq p_k$, which will be associated with a price $V_{j,k} = \sum_{p_u, p_v, p_u \neq p_v} \delta(j, k, u, v) \cdot \chi(p_u, p_v)$. We define a constant B^* to represent the maximum usage of resource over all edge server pairs for each task

$$B^* = \max_{t_i, p_j, p_k, p_j \neq p_k; u, p_v, p_u \neq p_v} \{(\delta \cdot r_i)/c_v\}. \tag{5}$$

Upon the arrival of a new task, the algorithm calculates the prices of all edge server pairs and figures out the edge server pair $\{p_{j^*}, p_{k^*}\}$ with the lowest price, denoted as V_{j^*,k^*}. If $V_{j^*,k^*} > 1$, the constraints of the dual program will be violated (see proof of Lemma 2). Then, the task will be rejected and the corresponding dual variable $\phi(t_i)$ will be set as 0. Otherwise, the task will be accepted and the algorithm will update the dual variables as follows:

$$\begin{cases} \phi(t_i) \leftarrow r_i(1 - V_{j^*,k^*}) & (6a) \\ \chi(p_u, p_v) \leftarrow \chi(p_u, p_v)\left[1 + \frac{\delta \cdot r_i}{c_v}\right] + \frac{\delta \cdot r_i}{n \cdot \varphi \cdot c_v} \quad \forall p_u, p_v & (6b) \end{cases}$$

The online primal-dual algorithm is described in Algorithm 1.

4.2 Performance Analysis

Before performance analysis, we first prove the feasibility of our primal-dual algorithm.

Lemma 2. *When the primal-dual algorithm terminates, there is no violation of constraints from the dual program Eqs. (3a) and (3b).*

Proof. There are two constraints in Eq. (3). On one hand, we consider the constraints from Eq. (3b). Initially, all the dual variables are 0, which satisfies the positivity constraint. When dual variables $\chi(t_i, p_u, p_v)$ with $t_i \in \mathcal{T}, p_u, p_v \in \mathcal{P}, p_u \neq p_v$ are updated, they will never decrease according to the update rules (Lines 9-10). Moreover, the update rule for dual variables, namely $\phi(t_i) = r_i(1 - V_{j^*,k^*})$, can guarantee that $\phi(t_i)$ is positive because only the task with $V_{j^*,k^*} < 1$ will be accepted. Therefore, constraints from Eq. (3b) are satisfied after the algorithm terminates.

On the other hand, let's consider the constraints in Eq. (3a). For each task, we will determine an edge server pair $\{p_{j^*}, p_{k^*}\}$ with the lowest price V_{j^*,k^*}. If it is rejected, according to Line 7 of Algorithm 1, $V_{j^*,k^*} \geq 1$ and the right side of Eq. (4) will not be positive. Since the dual variable $\phi(t_i)$ is nonnegative, the first constraint of the dual program is satisfied. If the task is accepted, according to the update rule of Algorithm 1 and the definition of V_{j^*,k^*} for each edge server pair $\{p_j, p_k\}$, it follows:

$$\phi(t_i) = r_i(1 - V_{j^*,k^*}) \geq r_i \left(1 - \sum_{p_u, p_v, p_u \neq p_v} \delta \cdot \chi(p_u, p_v)\right). \tag{7}$$

The above inequality is the same as the constraints in Eq. (3a). Therefore, the update of dual variables ϕ satisfies the constraints. Moreover, the update of dual variables $\chi(p_u, p_v)$ will only make the right side smaller, which will not violate the constraints. As a result, the first constraint of the dual program in Eq. (3a) is always satisfied as well. □

To evaluate the performance of the proposed algorithm, we define the competitive ratio as follows [34].

Definition 1. *An online algorithm is $[\alpha, \beta]$ competitive if it achieves at least $\alpha \cdot OPT$, where OPT is the result of the optimal solution, and the constraint in Eq. (3a) is violated by a factor β at most.*

Obviously, we expect the performance of our algorithm is close to that of the optimal solution, i.e., $\alpha \rightarrow 1$, and $\beta \rightarrow$

1. However, since the RTO-TM-1 problem is NP-hard, it is infeasible to reach the above expectation for any online algorithm. In the following, we will prove that the competitive ratio of our primal-dual algorithm is $[1-\epsilon, O(\log n + \log \frac{1}{\epsilon})]$.

Lemma 3. *The system throughput of our primal-dual task offloading algorithm is at least $(1-\epsilon)OPT$, where OPT is the result of the optimal solution.*

Proof. When task t_i is accepted, the objective value of the linear program increases by r_i. However, according to the update rules of dual variables in Algorithm 1, the objective value of the dual program increases by Δ_i

$$\begin{aligned}\Delta_i &= r_i[1 - V_{j^*,k^*}] \\ &\quad + \sum_{p_u,p_v,p_u\neq p_v} \chi(p_u,p_v)\left[\frac{\delta\cdot r_i}{c_v} + \frac{\delta\cdot r_i}{n\cdot\varphi\cdot c_v}\right]c_v \\ &= r_i - \sum_{p_u,p_v,p_u\neq p_v} \delta\cdot\chi(p_u,p_v)\cdot r_i \\ &\quad + \sum_{p_u,p_v,p_u\neq p_v} \chi(p_u,p_v)\left[\frac{\delta\cdot r_i}{c_v} + \frac{\delta\cdot r_i}{n\cdot\varphi\cdot c_v}\right]c_v \\ &= r_i + \sum_{p_u,p_v,p_u\neq p_v} \chi(p_u,p_v)\frac{\delta\cdot r_i}{n\cdot\varphi} \\ &= r_i + \frac{r_i}{n\cdot\varphi}\cdot\sum_{p_u,p_v,p_u\neq p_v}\chi(p_u,p_v)\delta \\ &\le r_i + \frac{r_i}{n\cdot\varphi} n\cdot B^* = (1+\epsilon)r_i.\end{aligned} \tag{8}$$

That is, our algorithm increases the objective of the dual algorithm by at most $(1+\epsilon)r_i$. As a result, the overall objective value of the dual program is at least $1/(1+\epsilon) \ge 1-\epsilon$ times as that of the optimal solution. Therefore, the system throughput of our primal-dual algorithm is at least $(1-\epsilon)OPT$. □

In the following, we consider the violation extent of the resource constraint on each edge server. For each task t_i and edge server pair $\{p_u, p_v\}$, let $L(v,i)$ be the load on p_v (including the incremental load re-offloaded from other edge servers) after task t_i has been processed. Note that $\chi(p_u,p_v,i)$ equals to $\chi(p_u,p_v)$ after task t_i is processed.

Lemma 4. *For each task t_i, and its assigned edge server pair $\{p_u,p_v\}$, we have*

$$\chi(p_u,p_v,i) \ge \frac{\exp[L(v,i)/c_v]-1}{n\cdot\varphi}. \tag{9}$$

Proof. We prove the lemma by the induction of task $t_i, i = 1,2,\ldots,m$. At the beginning, $\chi(p_u,p_v,0) = L(v,0) = 0$ for all p_u and p_v. Therefore, the inequality holds. Note that the value of $\chi(p_u,p_v)$ will be updated during the running of Algorithm 1. For ease of presentation, we denote $\chi(p_u,p_v,i)$ as the value of $\chi(p_u,p_v)$ after task t_i is processed. For each arrival task t_i, if t_i is rejected, two variables $\chi(p_u,p_v)$ and $L(v,i)$ will not be updated, i.e., $\chi(p_u,p_v,i) = \chi(p_u,p_v,i-1)$ and $L(v,i) = L(v,i-1)$. Thus the inequality holds as well. If task t_i is accepted, we have $L(v,i) = L(v,i-1) + \delta\cdot r_i$. According to the update rule of $\chi(p_u,p_v,i)$ in Line 10 of Algorithm 1, we also have

$$\chi(p_u,p_v,i) = \chi(p_u,p_v,i-1)\left[1+\frac{\delta r_i}{c_v}\right] + \frac{\delta r_i}{n\cdot\varphi\cdot c_v}. \tag{10}$$

By induction hypothesis, we apply inequality $\chi(p_u,p_v,i-1) \ge \frac{\exp[L(v,i-1)/c_v]-1}{n\cdot\varphi}$ to Eq. (10) and obtain

$$\begin{aligned}\chi(p_u,p_v,i) &\ge \frac{\exp[L(v,i-1)/c_v]-1}{n\cdot\varphi}\left[1+\frac{\delta\cdot r_i}{c_v}\right] + \frac{\delta\cdot r_i}{n\cdot\varphi\cdot c_v} \\ &= \frac{1}{n\cdot\varphi}\left[\exp\left[\frac{L(v,i-1)}{c_v}\right]\left[1+\frac{\delta\cdot r_i}{c_v}\right]-1\right] \\ &\approx \frac{1}{n\cdot\varphi}\left[\exp\left[\frac{L(v,i-1)}{c_v}\right]\exp\left[\frac{\delta\cdot r_i}{c_v}\right]-1\right] \\ &= \frac{\exp[L(v,i)/c_v]-1}{n\cdot\varphi}.\end{aligned} \tag{11}$$

Here we apply the first order approximation $\exp(x) \approx 1+x$ for a small positive value x. Strict inequality can be established by a more complicated update rule and incurs unnecessary complexity. As a result, the lemma holds. □

The following lemma guarantees the performance of our primal-dual algorithm in terms of the resource constraint on each edge server.

Lemma 5. *Our primal-dual algorithm will not violate the resource constraint by a factor of $O(\log n + \log(1/\epsilon))$ on each edge server.*

Proof. Without loss of generality, we consider the violation of the resource capacity constraint on each edge server p_v if an arbitrary edge server p_u fails. According to Algorithm 1, the value of $\chi(p_u{\cdot}p_v)$ will be updated only if $V_{j^*,k^*} < 1$ (the task will be accepted in Line 7 of Algorithm 1) and $\delta(j^*,k^*,u,v) \neq 0$ (otherwise the value of $\chi(p_u{\cdot}p_v)$ will not change according to the update rule of $\chi(p_u,p_v)$ in Line 10 of Algorithm 1). Combinning that $V_{j^*,k^*} = \sum_{p_u,p_v,p_u\neq p_v}\delta(j^*,k^*,u,v)\cdot\chi(p_u,p_v) < 1$ and $\delta(j^*,k^*,u,v)) \neq 0$, we have $\chi(p_u,p_v) \le 1$ before the last update of $\chi(p_u,p_v)$. Next, we consider the last update of $\chi(p_u,p_v)$. According to the update rule of $\chi(p_u,p_v)$ in Line 10 of Algorithm 1 and the definition of B^* in Eq. (5), we obtain $\chi(p_u,p_v) \le 1 + \frac{\delta\cdot r_i}{c_v} + \frac{\delta\cdot r_i}{n\cdot\varphi\cdot c_v} \le 1 + 2B^*$. By Eq. (9) in Lemma 4, we have

$$\frac{L(v,i)}{c_v} \le \log\left((2B^*+1)\cdot n\cdot\varphi + 1\right) = O(\log n + \log\frac{1}{\epsilon}). \tag{12}$$

In other words, the load on edge server p_v will be violated by a factor $O(\log n + \log\frac{1}{\epsilon})$ at most once the edge server p_u fails. Since two edge servers p_u and p_v are chosen arbitrarily, our performance analysis can be applied for all edge server pairs. □

Combining Lemmas 3 and 5, we present the competitive ratio of Algorithm 1.

Theorem 6. *Algorithm 1 can achieve the competitive ratio of* $[1-\epsilon, O(\log n + \log(1/\epsilon))]$*, where* ϵ *is an arbitrary parameter with* $\epsilon \in (0,1)$*, and* n *is the number of edge servers in DEC.*

4.3 Complete Primal-Dual Algorithm

Under practical situations, we may hope that the online algorithm should satisfy the resource constraint on each edge server. Otherwise, there may be not enough residual resource for tasks executed on failed edge servers. In this section, we propose a complete algorithm based on our primal-dual solution. Specifically, we add a checking mechanism to determine whether there is enough resource capacity on the edge server pair $\{p_{j^*}, p_{k^*}\}$ to accommodate the task or not. This checking mechanism is formally described by Function Checking in Algorithm 2, and the function will be inserted before Line 8 of Algorithm 1. In Algorithm 2, the Checking function mainly consider two points from the perspectives of the edge server p_{j^*} and the backup edge server p_{k^*}, respectively. (1) Once task t_i is accepted on edge server p_{j^*}, the load on edge server p_{j^*} will increase, and the residual capacity will decrease. To make our task offloading method resistant to 1-edge server failure, the incremental load on edge server p_{j^*} from arbitrarily another edge servers p_s should not exceed its residual capacity. (2) If the task is accepted, the incremental load on backup edge server p_{k^*} will increase. Accordingly, we should guarantee that there is enough residual capacity on the backup edge server p_{k^*} to accommodate the incremental load from edge server p_{j^*} in case that edge server p_{j^*} fails. If one requirement is not satisfied, it means the resource constraint on edge servers will be violated. Therefore, the function will return *false* to reject the task. Otherwise, the task will be accepted.

Algorithm 2. Checking$(L\{m\}, u\{m\times m\}, t_i, j^*.k^*)$

```
1: /*array: edge server load {L_m}; Incremental load {u_{m×m}}*/
2: for each edge server p_s ∈ P − p_{j*} do
3:     if u_{s,j*} > c_{j*} − (L_{j*} + r_i) then
4:         /* the incremental load on edge server p_{j*} from edge
           server p_s is larger than the residual capacity on edge
           server p_{j*} once the task t_i is accepted*/
5:         return false
6:         /* the task will be rejected*/
7:     end if
8: end for
9: if u_{j*,k*} + r_i > c_{k*} − L_{k*} then
10:        /* the incremental load on edge server p_{k*} from edge
           server p_{j*} is larger than the residual capacity on edge
           server p_{k*} if the task t_i is accepted*/
11:        return false
12: end if
13: return true
```

4.4 Extending to the RTO-TM-2 Problem

In this section, we extend the RTO-TM-1 problem to the RTO-TM-2 problem such that our task offloading scheme can be resistant to 2 edge server failures. Accordingly, each task t_i will be assigned a 3-server permutation $\mathcal{C}_j = \{p_{j_0}, p_{j_1}, p_{j_2}\}$ with $p_{j_0}, p_{j_1}, p_{j_2} \in \mathcal{P}_i$ to resist arbitrary 2 edge server failures. Let variable $y_i^{\mathcal{C}_j}$ (also denoted as $y_i^{j_0,j_1,j_2}$) represent whether task t_i is assigned to 3-server permutation $\{p_{j_0}, p_{j_1}, p_{j_2}\}$ ($y_i^{j_0,j_1,j_2}=1$) or not ($y_i^{j_0,j_1,j_2}=0$). Without failure, the load on the edge server p_k will be $\sum_{t_i} \sum_{j_0,j_1,j_2\in\mathcal{P}_i; j_0\neq j_1\neq j_2} y_i^{j_0,j_1,j_2}(1-\mathcal{I}_{j_0,k})r_i$. When edge servers $\{p_{f_1}, p_{f_2}\}$ with $p_{f_1}, p_{f_2} \in \mathcal{P}$ fail, their load will be offloaded to other edge servers. Here, we consider the incremental load on any edge server p_v with two following cases.

- $p_{j_0} \in \{p_{f_1}, p_{f_2}\}$, $p_{j_1} \notin \{p_{f_1}, p_{f_2}\}$ and $p_v = p_{j_1}$. Under this case, the first assigned edge server fails while the second edge server p_{j_1} in the 3-server permutation does not. Then, task t_i will be executed on edge server p_{j_1} and consequently, the incremental load on edge server p_v is $(1-\mathcal{I}_{j_1,v})[(1-\mathcal{I}_{f_1,j_0})+(1-\mathcal{I}_{f_2,j_0})]\mathcal{I}_{f_1,j_1}\cdot\mathcal{I}_{f_2,j_1}\cdot r_i$.
- $p_{j_0} \in \{p_{f_1}, p_{f_2}\}$, $p_{j_1} \in \{p_{f_1}, p_{f_2}\}$ and $p_v = p_{j_2}$. That is, both the edge servers p_{j_0} and p_{j_1} fail and task t_i has to be executed on edge server p_v. The incremental load on edge server p_k is $(1-\mathcal{I}_{j_2,v})[(1-\mathcal{I}_{f_1,j_0})+(1-\mathcal{I}_{f_2,j_0})][(1-\mathcal{I}_{f_1,j_1})+(1-\mathcal{I}_{f_2,j_1})]\cdot r_i$.

When edge servers p_{j_0} and p_{j_1} fail, the incremental load on edge server p_v can be calculated as

$$\sum\nolimits_{t_i\in\mathcal{T}} \theta(j_0,j_1,j_2,f_1,f_2,v)\cdot y_i^{j_0,j_1,j_2}\cdot r_i, \tag{13}$$

where

$$\begin{aligned}&\theta(j_0,j_1,j_2,f_1,f_2,v)\\ &=(1-\mathcal{I}_{j_1,v})[(1-\mathcal{I}_{f_1,j_0})+(1-\mathcal{I}_{f_2,j_0})]\mathcal{I}_{f_1,j_1}\cdot\mathcal{I}_{f_2,j_1}\\ &+(1-\mathcal{I}_{j_2,v})[(1-\mathcal{I}_{f_1,j_0})+(1-\mathcal{I}_{f_2,j_0})][(1-\mathcal{I}_{f_1,j_1})+(1-\mathcal{I}_{f_2,j_1})].\end{aligned} \tag{14}$$

Without confusion, $\theta(j_0,j_1,j_2,f_1,f_2,v)$ is abbreviated as θ. The overall load on edge server p_v is

$$\begin{aligned}&\sum\nolimits_{t_i}\sum_{j_0,j_1,j_2,j_0\neq j_1\neq j_2} y_i^{j_0,j_1,j_2}\cdot(1-\mathcal{I}_{j_0,v}+\theta)\cdot r_i\\ &=\sum\nolimits_{t_i}\sum\nolimits_{j_0,j_1,j_2,j_0\neq j_1\neq j_2} y_i^{j_0,j_1,j_2}\kappa\cdot r_i,\end{aligned} \tag{15}$$

where $\kappa=(1-\mathcal{I}_{j_0,v}+\theta)$. Accordingly, the RTO-TM-2 problem can be rewritten as follows:

$$\max\ \sum\nolimits_{t_i\in\mathcal{T}}\sum\nolimits_{\{j_0,j_1,j_2\}} y_i^{j_0,j_1,j_2}\cdot r_i$$

$$\begin{cases}\sum\limits_{\{j_0,j_1,j_2\}}^{n} y_i^{j_0,j_1,j_2}\le 1 & \forall t_i\in\mathcal{T} \quad (16a)\\ \sum\limits_{t_i}\sum\limits_{\{j_0,j_1,j_2\}} y_i^{j_0,j_1,j_2}\kappa\cdot r_i\le c_v & \forall\{p_{f_1},p_{f_2}\},p_v\in\mathcal{P} \quad (16b)\\ y_i^{j_0,j_1,j_2}\in\{0,1\} & \forall t_i\in\mathcal{T};\{j_0,j_1,j_2\} \quad (16c)\end{cases}$$

The dual version of Eq. (16) is shown as follow:

$$\min\ \sum\nolimits_{t_i}\nu(t)+\sum_{p_{f_1},p_{f_2},p_v}\zeta(p_{f_1},p_{f_2},p_v)c_v$$

$$\begin{cases} \nu(t_i)+\sum\limits_{p_{f_1},p_{f_2},p_v} \kappa\cdot r_i\zeta(p_{f_1},p_{f_2},p_v) \geq r_i & \forall t_i,\{j_0,j_1,j_2\} \quad (17a) \\ \nu(t_i) \geq 0 \ \zeta(p_{f_1},p_{f_2},p_v) \geq 0 & \forall t_i; p_{f_1},p_{f_2},p_v \quad (17b) \end{cases}$$

We can rewrite the first constraint of Eq. (17a) as

$$\nu(t_i) \geq r_i \left(1 - \sum_{p_{f_1},p_{f_2},p_v} \kappa \cdot \zeta(p_{f_1},p_{f_2},p_v)\right) \quad \forall t_i; \{p_{j_0},p_{j_1},p_{j_2}\}. \tag{18}$$

For each arrival task, the algorithm needs to determine an 3-server permutation $\{p_{j_0},p_{j_1},p_{j_2}\}$, which will be associated with a price $W_{j_0,j_1,j_2} = \sum_{p_{f_1},p_{f_2},p_v} \kappa \cdot \zeta(p_u,p_v)$. We define a constant D^* to represent the maximum usage of resource over all 3-server permutation for each task

$$D^* = \max_{t_i,p_{j_0},p_{j_1},p_{j_2},p_{f_1},p_{f_2},p_v}\{(\kappa \cdot r_i)/c_v\}. \tag{19}$$

Upon the arrival of a new task t_i, the algorithm calculates the prices of 3-server permutation in $\mathcal{P}_i$ and figures out the edge server pair $\{p_{j_0^*},p_{j_1^*},p_{j_2^*}\}$ with the lowest price, denoted as $W_{j_0^*,j_1^*,j_2^*}$. If $W_{j_0^*,j_1^*,j_2^*} > 1$, the constraints of the dual program will be violated (see supplementary file, available online). Then, the task will be rejected and the corresponding dual variable $\nu(t_i)$ will be set as 0. Otherwise, the task will be accepted and the algorithm will update the dual variables as follows:

$$\begin{cases} \nu(t_i) \leftarrow & r_i(1 - W_{j_0^*,j_1^*,j_2^*}) \quad (20a) \\ \zeta(p_{f_1},p_{f_2},p_v) \leftarrow & \zeta(p_{f_1},p_{f_2},p_v)[1+\frac{\kappa\cdot r_i}{c_v}] \\ & +\frac{\kappa\cdot r_i}{n^2\cdot\varphi\cdot c_v} \forall p_{f_1},p_{f_2},p_v \quad (20b) \end{cases}$$

The online primal-dual algorithm is formally described in Algorithm 3.

Algorithm 3. Primal-Dual Based Task Offloading Algorithm for RTO-MT-2

1: $\varphi = D^*/\epsilon$
2: Initialize all the dual variables:
3: $\nu(t_i) \leftarrow 0 \quad \forall t_i$
4: $\zeta(p_{f_1},p_{f_2},p_v) \leftarrow 0 \quad \forall p_{f_1},p_{f_2},p_v$
5: **for** each arrival task t_i **do**
6: $\quad \{j_0^*,j_1^*,j_2^*\} \leftarrow \text{argmin}_{p_{j_0},p_{j_1},p_{j_2}\in\mathcal{P}_i} W_{j_0,j_1,j_2}$
7: $\quad$ **if** $W_{j_0^*,j_1^*,j_2^*} < 1$ **then**
8: $\quad\quad$ Assign task t_i to 3-server permutation $\{p_{j_0^*},p_{j_1^*},p_{j_2^*}\}$
9: $\quad\quad \nu(t_i) \leftarrow r_i(1 - W_{j_0^*,j_1^*,j_2^*})$
10: $\quad\quad \forall p_{f_1},p_{f_2},p_v, \zeta(p_{f_1},p_{f_2},p_v) \leftarrow \zeta(p_{f_1},p_{f_2},p_v)[1+\frac{\kappa\cdot r_i}{c_v}]+\frac{\kappa\cdot r_i}{n^2\cdot\varphi\cdot c_v}$
11: $\quad$ **else**
12: $\quad\quad$ Reject the task and set $\nu(t_i) \leftarrow 0$
13: $\quad$ **end if**
14: **end for**

The algorithm for RTO-MT-2 is consistent to the algorithm for RTO-MT-1. The different is that the formulation and corresponding algorithm of RTO-TM-2 replace constant δ in Eq. (2) with constant κ in Eq. (16). Therefore, similar to Theorem 6, we can derive the competitive ratio of the modified online algorithm.

Theorem 7. *Our modified primal-dual algorithm can achieve the competitive ratio of* $[1-\epsilon, O(\log n + \log\frac{1}{\epsilon})]$ *for the RTO-TM-2 problem, where* ϵ *is an arbitrary parameter with* $\epsilon \in (0,1)$.

The proof is put in Appendix, which can be found in the supplementary file, available online.

4.5 Extension for RTO-TM-h ($h \geq 3$)

In this section, we discuss how to extend our proposed algorithms for the general RTO-TM problem. By comparing the problem formulations of RTO-TM-1 and RTO-TM-2, the main difference is that the binary constant δ for RTO-TM-1 is replaced by constant κ for RTO-TM-2. Constant $\delta(j,k,u,v)$ for RTO-TM-1 (or $\kappa(j_0,j_1,j_2,f_1,f_2,v)$ for RTO-TM-2) means whether the task, assigned with the edge server permutation $\{p_j,p_k\}$ (or $\{j_0,j_1,j_2\}$), will eventually be offloaded to the edge server p_v or not when the failed edge server(s) is p_u (or p_{f_1},p_{f_2}). Likewise, for RTO-TM-h with $h \geq 3$, we define the binary constant ψ_h with arguments of $(h+1)$-server permutation $\mathcal{C}_j = \{p_{j_0},p_{j_1},\ldots,p_{j_h}\}$ with $p_{j_0},p_{j_1},\ldots,p_{j_h} \in \mathcal{P}_i$, the set of h failed edge servers $\mathcal{F}_k = \{p_{k_1},p_{k_2},\ldots,p_{k_h}\}$ and an arbitrary edge server p_v, denoted as $\psi_h(\mathcal{C}_j,\mathcal{F}_k,v)$. This constant means whether the task, assigned with the $(h+1)$-server permutation $\mathcal{C}_j$, will be offloaded to edge server p_v or not if h edge servers in $\mathcal{F}_k$ fail. For the detailed mathematical expression of ψ_h, we need to consider $h+1$ circumstances. 1) $p_v = p_{j_1}$ and $p_{j_0} \in \mathcal{F}_k$. That is, the first edge server p_{j_0} in $(h+1)$-server permutation fails and the second one does not. As a result , the task will be re-offloaded to p_{j_1} and $\psi_h(\mathcal{C}_j,\mathcal{F}_k,j_1) = 1$; 2) $p_v = p_{j_2}$ and $p_{j_0},p_{j_1} \in \mathcal{F}_k$, which means the first two edge servers p_{j_0} and p_{j_1} belonging to $(h+1)$-server permutation fail and the third one is active. Consequently, the task is re-offloaded to p_{j_2} and we obtain $\psi_h(\mathcal{C}_j,\mathcal{F}_k,j_2) = 1$; ... h) $p_v = p_{j_h}$ and $p_{j_0},\ldots,p_{j_h} \in \mathcal{F}_k$. We have that $\psi_h(\mathcal{C}_j,\mathcal{F}_k,j_h) = 1$; $h+1$) the task will not be re-offloaded if the prior h circumstances do not happen. Thus, the task is executed on the first edge server p_{j_0} in $\mathcal{C}_j$ and we have $\psi_h(\mathcal{C}_j,\mathcal{F}_k,j_0) = 1$. Once $\psi_h(\mathcal{C}_j,\mathcal{F}_k,v)$ is determined, the formulation of the RTO-TM-h problem will be presented as a linear program like Eq. (2) for RTO-TM-1 and Eq. (16) for RTO-TM-2. We can derive the dual program of the linear program and design similar primal-dual algorithm. The theoretical performance of the primal-dual algorithm can also be guaranteed like Theorem 6 for RTO-TM-1 and Theorem 7 for RTO-TM-2.

4.6 Time Complexity

Theorem 8. *For each task, the time complexity of Algorithm 1 (resistant to 1 edge server failure) is* $O(m^2)$ *and that of Algorithm 3 (resistant to 2 edger server failures) is* $O(m^3)$, *where* m *is the number of edge servers.*

Proof. Consider line 5 in Algorithm 1 for RTO-MT-1, which calculates the value of $V_{j,k}, \forall p_j,p_k \in \mathcal{P}$. Given an arbitrary $V_{j,k}$, according to the definition $V_{j,k} = \sum_{p_u,p_v,p_u\neq p_v}\delta(j,k,u,v)\cdot\chi(p_u,p_v)$ and the definition $\delta(j,k,u,v)$ that $\delta(j,k,u,v) = (1-\mathcal{I}_{j,v}) + (1-\mathcal{I}_{j,u})(1-\mathcal{I}_{k,v})$, we have

$$V_{j,k} = \sum_{p_u} \chi(p_u,p_j) + \chi(p_u,p_j),$$

where $\sum_{p_u}\chi(p_u,p_j)$ can be maintained by a float value and is updated every time we change the value of

$\chi(p_u, p_j)$. Therefore, the time complexity of calculating $V_{j,k}$ is $O(1)$. Since line 5 calculates $V_{j,k}, \forall p_j, p_k \in \mathcal{P}$, the time complexity of line 5 is $O(m^2)$. Line 10 updates the value of $\chi(p_u, p_j) \forall p_u, p_v \in \mathcal{P}$ and the time complexity is also $O(m^2)$. Overall, the time complexity of Algorithm 1 is $O(m^2)$. Similarly, we can conclude that the time complexity of Algorithm 3 is $O(m^3)$ as it calculates $W_{j_0,j_1,j_2} \quad \forall p_{j_0}, p_{j_1}, p_{j_2} \in \mathcal{T}$ and updates $\zeta(p_{f_1}, p_{f_2}, p_v)$ $\forall p_{f_1}, p_{f_2}, p_v \in \mathcal{P}$. □

4.7 Discussion

This paper considers the resource consumption of tasks. In fact, some other important parameters of tasks will also affect the quality of task offloading solution. For instance, the authors in [35], [36] point out that the size of input data is also an important parameter that will affect the transmission delay. The tasks with large input data may intend to be executed locally, incurring less transmission delay. In robust task offloading, if we expect to reduce the transmission delay of the task's input data, we should offload the task to the proximity edge servers. Specifically, the formulation of the robust task offloading in Section 3 shows that each task t_i can only be assigned to the edge servers from the set $\mathcal{P}_i$, which can be customized by only keeping the edge servers whose transmission delay of the in-put data of task t_i is within a threshold. In this way, we can bound the transmission delay for each task under given size of its input data.

5 Performance Evaluation

In this section, we perform the extensive simulations for the RTO-TM problem. To this end, we first evaluate the performance of our proposed online algorithms for the RTO-TM-1 problem. Then, we extend the simulations for RTO-TM-2.

5.1 Simulation Settings

We simulate the DEC system with 40-60 end devices, in which 40 percent of the end devices can act as edge servers. Each end device may only be able to set up connections to a subset of all the edge servers, which are randomly selected for each end device in the simulation. The quantities of both end devices and edge servers are common in practical edge computing scenarios [37], [38]. We conduct extensive simulations to demonstrate the performance of our algorithms using a real task set, which was collected from a Borg cell in Google [39]. There are two kinds of resources in the task set, computing resource and memory resource. The units for the computing resource and memory resource are normalized value of the average number of cores used by the task and the normalized value of the average memory consumed by the task. More details of the task set can be found in [39]. Since our algorithm does not distinguish the types of resources, we evaluate the performance of different algorithms according to different resource requirements. To this end, the task set will be divided into two kinds of task sets, namely computing intensity task set and memory intensity task set. Similar to resource capacity settings in [38], the capacity of each edge server in our simulations is randomly set as 30-50 times of the average resource consumption of all tasks.

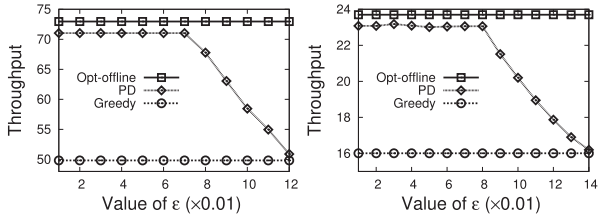


Fig. 2. Value of ϵ versus throughput in DEC. *Left plot*: Memory intensity task set; *right plot*: Computing intensity task set.

5.2 Performance Methodology and Metrics

According to Theorem 6, the competitive ratio of the proposed primal-dual algorithm is closely related to the input parameter ϵ. Therefore, we first evaluate the impact of parameter ϵ on the system throughput (see Fig. 2 for details). By the results in Fig. 2, we adopt two values of ϵ, namely 0.05 and 0.10, as our representative online primal-dual algorithms, which, for ease of description, are denoted as PD-0.05 and PD-0.10, respectively. What's more, we also adopt two benchmarks. The first one is the greedy algorithm, denoted as "Greedy" for simplicity, in which two edge servers with the larger residual capacity will be assigned as the edge server and the backup edge server for each task, respectively. The second metric is the optimal offline solution, denoted as Opt-offline. The optimal result can be obtained by solving the integer linear program formulated in Eqs. (2a)-(2c) with a linear program solver, e.g., pulp [40], which may be time consuming and needs some prior knowledge of tasks. The results of the Opt-offline solution are the upper bound of all the online algorithms, including PD-0.05 and PD-0.10.

We adopt two metrics to observe the performance of all the algorithms. The first metric is the system throughput, which means the total load of all accepted tasks in DEC. This metric can directly reflect the performance of our algorithm. The second metric is the cumulative distribution function (CDF) of load ratios among all the edge servers. Higher load ratio means better resource utilization on edge servers. Since our problem formulation is resistant to the edge server failure(s) in DEC, we will make randomly selected one/two edge server(s) fail. Accordingly, the tasks offloaded to the failed edge server(s) will be re-offloaded to the backup edge servers for further processing. Under this circumstance, we evaluate the CDF of load ratios among all the edge servers as well. To enhance the CDF evaluation, we conduct two groups of experiments with different quantities of tasks (i.e., 700 and 1000) in DEC to observe the performance of all algorithms under normal-loaded DEC and heavy-loaded DEC.

5.3 Simulation Results of System Throughput

We first analyze the impact of parameter ϵ on the system throughput in DEC for the memory intensity task set and computing intensity task set, respectively. The results are shown in Fig. 2. For the memory intensity task set, when the value of ϵ is smaller than 0.07, the throughput of our primal-dual algorithm is above 95 percent of the upper bound by Fig. 2a. As ϵ exceeds 0.07, the system throughput of our primal-dual algorithm decreases. That's because our

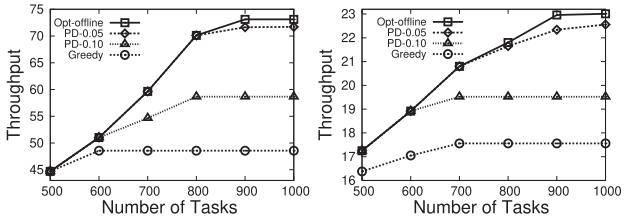


Fig. 3. Number of tasks versus throughput in DEC. *Left plot*: Memory intensity task set; *right plot*: Computing intensity task set.

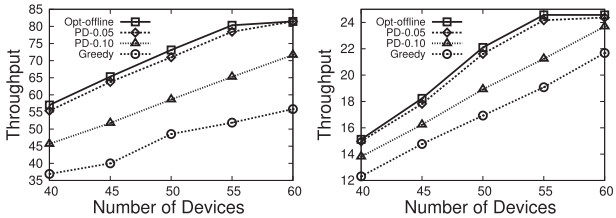


Fig. 4. Number of end devices versus throughput in DEC. *Left plot*: Memory intensity task set; *right plot*: Computing intensity task set.

primal-dual algorithm can achieve $(1-\epsilon)$ times as the optimal result, which is decreasing with the value of ϵ increasing. However, the system throughput of the primal-dual algorithm is still larger than that of the greedy algorithm until ϵ exceeds 0.12. Moreover, for the computing intensity task set, Fig. 2b shows that our primal-dual algorithm can keep high throughput (above 95 percent as that of the Opt-offline algorithm) if ϵ is less than 0.08. When ϵ increases from 0.09 to 0.14, the throughput of DEC decreases but is still larger than that of the greedy algorithm. From the above results, we conclude that the performance of our primal-dual algorithm can be divided into two phases, near optimal phase when ϵ is in range of [0, 0.07] for the memory intensity task set (or [0, 0.08] for the computing intensity task set) and decreasing phase when ϵ is larger than 0.07 for the memory intensity task set (or 0.08 for the computing intensity task set). Accordingly, we pick 0.05 and 0.10 as two inputs of parameter ϵ to represent the near optimal phase and decreasing phase, respectively.

The second group of simulations observes the impact of the number of tasks (from 500 to 1000 [34]) on the system throughput performance for two task sets. The results are shown in Fig. 3. Generally, the system throughput goes up when the number of tasks increases. Our PD-0.05 algorithm can achieve near optimal throughput while the PD-0.10 and greedy algorithms cannot accommodate as much throughput as the Opt-offline algorithm. For example, when there are 900 memory intensity tasks, the PD-0.05 algorithm can achieve 97 percent of the throughput obtained by the Opt-offline algorithm, while the PD-0.10 and greedy algorithms reduce the throughput by 22 and 35 percent, respectively, compared with the Opt-offline algorithm. Another example is that, the system throughput of the greedy algorithm stops increasing after the number of tasks exceeds 600 for the computing intensity task set. Under the same circumstance, PD-0.10 does not improve the system throughput when there are over 700 tasks, and PD-0.05 keeps increasing when the number of tasks is increasing from 500 to 900. The above simulation results show that, compared with the greedy algorithm, our primal-dual algorithms can accommodate more tasks and achieve higher system throughput.

The third group of simulations focuses on how the number of edge servers affects the system throughput of DEC. The simulation results for 900 tasks are shown in Fig. 4. As the number of edge servers increases, the througputs of all the algorithms increase. When there are more than 55 devices, the throughputs of both Opt-offline and PD-0.05 remain stable. By comparison, the throughputs of the PD-0.10 and greedy algorithms keep increasing when the number of devices is in the range of [40, 60]. The reason is that, Opt-offline or PD-0.05 can accommodate all the 900 tasks when there are 55 devices. However, the PD-0.10 and greedy algorithms cannot handle 900 tasks even when the number of devices is 60. In conclusion, PD-0.05 can efficiently offload the tasks to reduce resource consumption compared with PD-0.10 and the greedy algorithm.

5.4 Simulation Results of CDF of Load Ratios

This section evaluates the CDF of load ratios among all edge servers under DEC. For comparison, we first show the results when all the edge servers run without failure. Then, we will show the CDF when one randomly chosen edge server fails. Since the Opt-offline algorithm can only accommodate up to 1000 tasks, which is shown in Fig. 3, we adopt a value less than 1000 (e.g., 700) and 1000 as the number of tasks to represent the normal-loaded and heavy-loaded instances, respectively.

The fourth group of simulations studies CDF of load ratios among all edge servers with 700 tasks, and the simulation results are shown in Fig. 5. We observe that the Opt-offline algorithm leads to the highest average load ratio on edge servers among these solutions, which means Opt-offline can utilize the resource on edge servers effectively. As for our PD-0.05 algorithm, its CDF is very close to that of Opt-offline, especially for the results under the memory intensity task set. For example, the load ratios among all edge servers using PD-0.05 locate in the range of [0.75,0.80], and those using Opt-offline are in the range of [0.77,0.80] by Fig. 5a. By comparison, the greedy algorithm leads to much lower load ratios on edge servers compared with other algorithms. For instance, the load ratios for the greedy algorithm are within [0.64,0.67], reducing the average load ratio of these edge servers by about 15 percent compared with PD-0.05. These results are consistent with Fig. 3, in which the greedy algorithm cannot achieve as high throughput as PD-0.05 and Opt-offline.

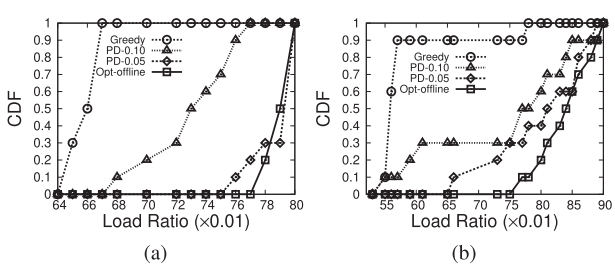


Fig. 5. Load ratio on edge servers versus CDF for 700 tasks. *Left plot*: Memory intensity task Set; *right plot*: Computing intensity task set.

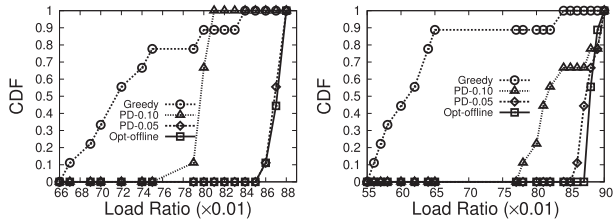


Fig. 6. Load ratio versus CDF for 700 tasks if one edge server fails. *Left plot*: Memory intensity task set; *right plot*: Computing intensity task set.

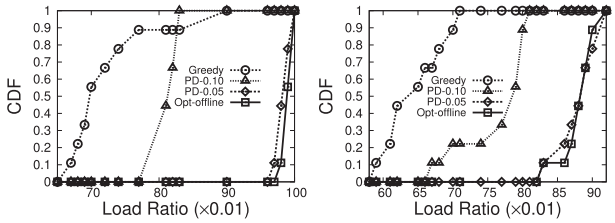


Fig. 8. Load ratio versus CDF for 1000 tasks if one edge server fails. *Left plot*: Memory intensity task set; *right plot*: Computing intensity task set.

Next, we let one (randomly selected) edge server fail and observe the CDF of load ratios among other edge servers. The results are shown in Fig. 6. We should highlight three points compared with the results in Fig. 5 when all edge servers run normally. First, our PD-0.05 algorithm can still maintain the even load distribution on all edge servers. For instance, the load ratios among edge servers using PD-0.05 are in the range of [0.85,0.88] and so are those using Opt-offline by Fig. 6a. Second, our primal-dual algorithm with $\epsilon = 0.10$ can achieve even load distribution among other edge servers when one edge server fails. Compared with the CDF results in Fig. 5 when all edge servers work normally, load ratios among edge servers (except the failed one) when one server failure occurs are very close to each other. Specifically, the load ratios among edge servers locate in the range of [0.75,0.81] for memory intensity task set by Fig. 6a, while those locate in the range of [0.67,0.77] when all edge servers work normally by Fig. 5a. Third, the greedy algorithm cannot achieve fair task offloading when one edge server fails. The load ratios on different edge servers vary a lot when edge server failure occurs. For instance, the minimum load ratio among all edge servers is 0.66, while the maximum load ratio is 0.83, which is a pretty large gap (0.17) compared with the gap for PD-0.10 (0.06), PD-0.05 (0.03) and Opt-offline (0.03).

The fifth group of the simulations studies the CDF performance under DEC. In the simulations, we generate 1000 tasks to make DEC heavily loaded. We present the CDF of load ratios among all edge servers in Fig. 7 when each edge server works normally. Compared with CDF results for 700 tasks, the load ratios among edge servers for PD-0.50, PD-0.10 and Opt-offline increase while those for the greedy algorithm do not change too much. Moreover, PD-0.05 and PD-0.10 guarantee the fair task offloading among all edge servers. For instance, the load ratios among all edge servers for PD-0.05 are in the range of [0.84,0.90]. As another example, the minimum load ratio among all the edge servers for PD-0.10 is 0.69, which is large than the maximum load ratio among all edge servers for the greedy algorithm.

In the following, one (randomly chosen) edge server will break down and we present the updated load ratios among all other edge servers. The results in Fig. 8 show that our PD-0.05 algorithm and Opt-offline can achieve nearly 100 percent of load ratios among all edge servers for memory intensity task set. That means our PD-0.05 algorithm can fully utilize the resources of all edge servers under heavy-loaded DEC. By comparison, the greedy algorithm may leave some spare resource and can only achieve the load ratios in the range of [0.65,0.90] for memory intensity task set. That's because our primal-dual algorithm aims to maximize the throughput of DEC while considering the case of one edge server failure. As a result, our algorithm can well adapt to the DEC system.

5.5 CDF of Load Ratios Under Two Server Failures

In this section, we observe the CDF of load ratios among all edge servers for the RTO-TM-2 problem. Under this case, our primal-dual algorithm can be resistant to 2 edge server failures. We first present the CDF of load ratios among all the edge servers without edge server failures. After that, we let two randomly chosen edge servers fail and give the updated CDF of load ratios among other edge servers. The simulation results are based on 1000 tasks and 50 devices, in which 20 devices act as edge servers.

As shown in Fig. 9, our PD-0.05 algorithm only increases the average load ratio of all the edge servers by 5 percent, compared with Opt-offline. Both algorithms can make use of majority resources on each edge server, even if there is no edge server failure. By comparison, the greedy algorithm can only utilize a minority of the resources on each edge server. For instance, PD-0.05, Opt-offline and the greedy algorithm utilize over 75, 80 and 40 percent, respectively, of the resources on each edge server by Fig. 9a. Next, we let

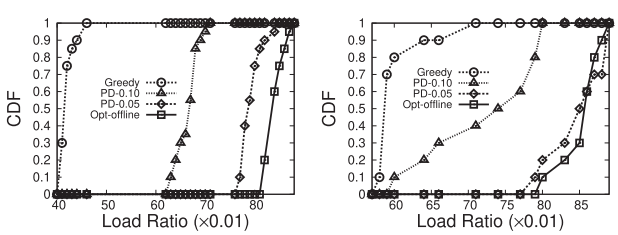


Fig. 7. Load ratio versus CDF for 1000 tasks. *Left plot*: Memory intensity task set; *right plot*: Computing intensity task set.

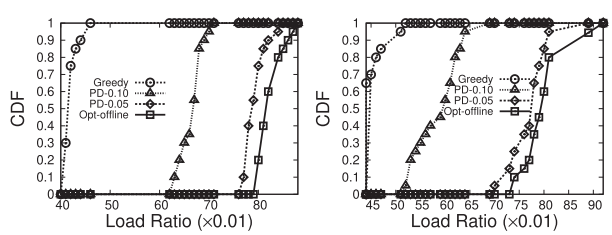


Fig. 9. Load ratio versus CDF for RTO-TM-2. *Left plot*: Memory intensity task set; *right plot*: Computing intensity task set.

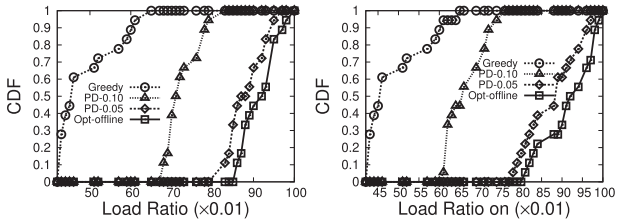


Fig. 10. Load ratio versus CDF if two edge servers fail. *Left plot*: Memory intensity task set; *right plot*: Computing intensity task set.

two randomly selected edge servers fail. Under this situation, all the tasks on the failed edge servers will be offloaded to peer edge servers with enough residual resources. The CDF of load ratios among other edge servers is shown in Fig. 10. As we can see, both PD-0.05 and Opt-offline can maintain high resource utilization on each edge server. By Fig. 10a, the average load ratios among all edge servers by PD-0.05 and Opt-offline are 0.90 and 0.92, respectively. On the contrary, the greedy algorithm can make use of 0.53 resources of edge servers on average.

From the above simulation results, we can draw some conclusions. First, our primal-dual algorithms, especially PD-0.05, can achieve near optimal system throughput compared with the Opt-offline algorithm. Second, when all edge servers run normally (i.e., without failure), our proposed algorithms can guarantee fair task offloading compared with the greedy algorithm. Specifically, the minimum load ratio among all edge servers of our algorithms is larger than the maximum one of the greedy algorithm. Third, our proposed algorithms are suitable for task offloading in DEC. Our proposed algorithm for RTO-TM-1 achieves fair task offloading among all edge servers even one edge server fails. Moreover, our online algorithm for RTO-TM-2 can be resistant to 2 edge server failures and only reduces the system throughput by 5 percent compared with the Opt-offline algorithm.

5.6 Running Time

Recall that the proposed prime-dual algorithms can handle the incoming task in real time. This section presents the running time of Algorithms 1 and 3. We stress that the running time will not be affected by the parameter ϵ. The results in Table 2 show that both algorithms can calculate the solution for the arrival task very fast. For instance, Alg. 1 outputs an edge server pair (one for task execution and one acts as a backup edge server) for each task in 19 ms, which is much smaller than the connection setup delay (i.e., [160-179] ms. Refer to Section 5.7 for details).

5.7 Practicability Evaluation

In this section, we evaluate the practicability of our proposed system model. We build a small-scale testbed, which consists of 7 Android phones equipped with the CPU of Qualcomm Snapdragon 660 and 6 GB RAM and 7 Raspberry Pis 3 Model B+ equipped with the SoC of Broadcom BCM2837B0, Cortex-A53 (ARMv8) and 1 GB LPDDR2 SDRAM, as shown in Fig. 11. The raspberry Pi acts as the end device and the phone acts as either the end device or the edge server. In our implementation, the specific task is

TABLE 2
Running Time of Algorithms 1 for RTO-MT-1 and 3 for RTO-MT-2 to Handle Each Arrival Task

Algorithm	Algorithm 1 for RTO-MT-1	Algorithm 3 for RTO-MT-2
Time (ms/task)	19	189

Internet sharing service, which is an example of sharing the peer devices' data budgets. The end devices that need the access to the Internet produce multiple tasks. The tasks include downloading HTML text contents of a web page from the URL and collecting its resource URLs (image, audio, video etc.). Cell phones that are reachable to the Internet can share their data budget to act as edge servers.

There is a manager module that collects the resource consumption of each task and the data budget of each edge server. When an end device produces a task (i.e., an Internet request), it will send the information to the manager, which runs our online task offloading algorithm and figures out the assigned edge server and backup edge server(s). On receiving the decision from the manager, the end device will connect to the hotpot of the assigned edge server so as to have Internet access. The time from sending the task request to setting up the connection to the assigned edge server is called *offloading delay*. Due to edge dynamics, the edge server may fail. Then the end device will reconnect to the backup edge server. The time during reconnection is called *reconnection delay*. The manager module is run in a third-party host or an edge server, which correspond to the cases with third-party host and without third-party host, respectively. The experimental results show that there is less than 1 percent battery consumption if the manager module is installed in a cell phone.

The results about the offloading delay and reconnection delay are shown in Figs. 12 and 13, respectively. In Fig. 12, when the number of edge servers increases from 3 to 7, the offloading delay increases from 205 ms to 239 ms. That is because more edge servers will cost more time for the controller to make offloading decisions. In Fig. 13, the reconnection delay locates in the range of [160, 179] ms under different numbers of edge servers. In addition, we also care

Fig. 11. System components: 7 Raspberry Pis acting as end devices, and 7 cell phones acting as either end devices or edge servers. A host is required if the manager module is installed in a third-party entity.

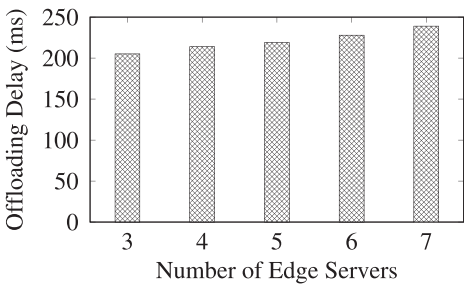


Fig. 12. Offloading delay versus number of edge servers.

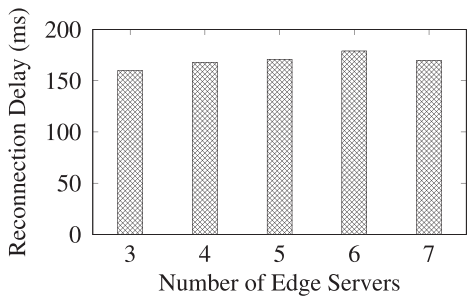


Fig. 13. Reconnection delay versus number of edge servers.

about the data usage for the control message, including the task request from the end device to the controller, offloading decision from the controller to the end device, and the connection setup from one end device to one edge server. Using Android Studio Network Profiler for inspecting network traffic, we find that all the control messages can be carried by one packet with the estimated packet size less than 1.2 kB. This shows the cost of control message is much less than the data sharing usage among cell phones (usually $\geq$ 1 MB). These experimental results show high practicability of our proposed solutions.

6 Related Work

This paper focuses on the task execution by peer end devices rather than the edges or the mobile clouds. In the literature about task offloading in MEC, several works aim to motivate end devices to offer their resources [12], [41], [42]. For instance, The work in [41] designs task assignment mechanism through possible incentive schemes and the authors of [42] construct a Stackelberg game model to investigate the interaction between the tasks and the end devices participating in task execution. Moreover, some literature adopts the auction mechanism to allocate the resources of end devices. For instance, two works [12], [13] design two auction mechanisms for cloudlet resource sharing in multi-access cloud computing. However, these works assume the homogeneous resource requirement of all the tasks. Moreover, they only consider the one-to-one matching manner, which neglects the fact that some devices can support multiple tasks in practical systems. Furthermore, the work in [15] designs a task assignment algorithm considering different tasks require different amount of resources. It also enables multiple tasks to be executed at one end device if there are adequate resources. The authors in [43], [44], [45] study the problem of determining the optimal number of edge servers for task offloading. For instance, the work [45] proposed a fully distributed framework, where both the MEC servers and the IoT devices act in an intelligent manner, making the most beneficial actions for the problem of autonomous MEC servers' activation and the problem of task offloading, respectively. The above literature does not consider the edge dynamics in MEC, which is the focus of this paper.

Edge dynamic is an important property in MEC, which may be caused by the mobility of end devices or unavoidable device failure [32]. Only a few works have noticed this characteristics and tried to improve the reliability of the MEC system. The work in [32] considers the end devices as the complement of the edge cloud. That is, the peer end devices act as backups to serve the tasks if the edge cloud is unreachable. In the MEC system with only end devices, the work in [46] considers the task offloading among peers with the objective of minimizing the offloading failure probability. The work in [47] proposes an offloading scheme to minimize the resource consumption with the constraint of offloading failure probability. Although these two works can improve the reliability of the offload scheme, they does not answer the question "what if the offloading failure happens?". There is some work applying replication idea to task offloading against the task execution failure or long task completion time in the cloud [48], [49]. They replicate a task on multiple servers and wait for the earliest completion of one replica of the task. Under effective scheduling, this method can reduce the task completion time and prevent task execution failure on one servers. However, In MEC system, resources like computing capacity is very limited on end devices/edge servers. If we assign one task to k devices, roughly speaking, we reduce the system throughput by $\frac{1}{k}$, which is a significant reduction.

7 Conclusion

To conquer the dynamics of edge servers, we have proposed a robust task offloading scheme which is resistant to h ($\geq$1) edge server failure(s). We study the problem for different values of h. For case of $h=1$, we have designed an online primal-dual-based algorithm and theoretically derive its competitive ratio. We have also extended the primal-dual algorithm to the case of $h=2$, and further to general cases for $h \geq 3$. A small-scale testbed is built to prove the practicability of the proposed system model. The extensive simulation results show that our algorithms can well handle the edge server failures and achieve near optimal throughput of DEC compared to the optimal offline benchmark algorithm.

Acknowledgments

This work was supported by the National Science Foundation of China (NSFC) under Grants 61822210, 61936015, and U1709217.

References

[1] X. Lyu, H. Tian, W. Ni, Y. Zhang, P. Zhang, and R. P. Liu, "Energy-efficient admission of delay-sensitive tasks for mobile edge computing," *IEEE Trans. Commun.*, vol. 66, no. 6, pp. 2603–2616, Jun. 2018.

[2] Y. Mao, S. Yi, Q. Li, J. Feng, F. Xu, and S. Zhong, "A privacy-preserving deep learning approach for face recognition with edge computing," in *Proc. USENIX Workshop Hot Topics Edge Comput.*, 2018, pp. 1–6.

[3] J. Wang, J. Pan, F. Esposito, P. Calyam, Z. Yang, and P. Mohapatra, "Edge cloud offloading algorithms: Issues, methods, and perspectives," *ACM Comput. Surv.*, vol. 52, no. 1, 2019, Art. no. 2.

[4] P. Zhou *et al.* "QoE-aware 3D video streaming via deep reinforcement learning in software defined networking enabled mobile edge computing," *IEEE Trans. Netw. Sci. Eng.*, vol. 8, no. 1, pp. 419–433, Jan.–Mar. 2021.

[5] R.-A. Cherrueau, A. Lebre, D. Pertin, F. Wuhib, and J. M. Soares, "Edge computing resource management system: A critical building block! Initiating the debate via OpenStack," in *Proc. USENIX Workshop Hot Topics Edge Comput.*, 2018, pp. 1–6.

[6] Q. Fan and N. Ansari, "Application aware workload allocation for edge computing-based IoT," *IEEE Internet Things J.*, vol. 5, no. 3, pp. 2146–2153, Jun. 2018.

[7] M. Mehrabi, D. You, V. Latzko, H. Salah, M. Reisslein, and F. H. P. Fitzek, "Device-enhanced MEC: Multi-access edge computing (MEC) aided by end device computation and caching: A survey," *IEEE Access*, vol. 7, pp. 166 079–166 108, 2019.

[8] Q.-V. Pham *et al.* "A survey of multi-access edge computing in 5G and beyond: Fundamentals, technology integration, and state-of-the-art," *IEEE Access*, vol. 8, pp. 116 974–117 017, 2020.

[9] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, "Mobile edge computing—A key technology towards 5G," *ETSI White Paper*, vol. 11, no. 11, pp. 1–16, 2015.

[10] J. Zhao, Q. Li, Y. Gong, and K. Zhang, "Computation offloading and resource allocation for cloud assisted mobile edge computing in vehicular networks," *IEEE Trans. Veh. Technol.*, vol. 68, no. 8, pp. 7944–7956, Aug. 2019.

[11] Y. Wang, M. Sheng, X. Wang, L. Wang, and J. Li, "Mobile-edge computing: Partial computation offloading using dynamic voltage scaling," *IEEE Trans. Commun.*, vol. 64, no. 10, pp. 4268–4282, Oct. 2016.

[12] A.-L. Jin, W. Song, P. Wang, D. Niyato, and P. Ju, "Auction mechanisms toward efficient resource sharing for cloudlets in mobile cloud computing," *IEEE Trans. Services Comput.*, vol. 9, no. 6, pp. 895–909, Nov./Dec. 2016.

[13] A.-L. Jin, W. Song, and W. Zhuang, "Auction-based resource allocation for sharing cloudlets in mobile cloud computing," *IEEE Trans. Emerg. Topics Comput.*, vol. 6, no. 1, pp. 45–57, Jan.–Mar. 2018.

[14] S. Long, W. Long, Z. Li, K. Li, Y. Xia, and Z. Tang, "A game-based approach for cost-aware task assignment with QoS constraints in large data centers," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 7, pp. 1629–1640, Jul. 2021.

[15] X. Wang, X. Chen, and W. Wu, "Towards truthful auction mechanisms for task assignment in mobile device clouds," in *Proc. IEEE Conf. Comput. Commun.*, 2017, pp. 1–9.

[16] Y. Mao, J. Zhang, and K. B. Letaief, "Dynamic computation offloading for mobile-edge computing with energy harvesting devices," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 12, pp. 3590–3605, Dec. 2016.

[17] H. Tan, Z. Han, X.-Y. Li, and F. C. M. Lau, "Online job dispatching and scheduling in edge-clouds," in *Proc. IEEE Conf. Comput. Commun.*, 2017, pp. 1–9.

[18] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 3, pp. 1628–1656, Third Quarter 2017.

[19] S. Wang, X. Zhang, Y. Zhang, L. Wang, J. Yang, and W. Wang, "A survey on mobile edge networks: Convergence of computing, caching and communications," *IEEE Access*, vol. 5, pp. 6757–6779, 2017.

[20] X. Shan, H. Zhi, P. Li, and Z. Han, "A survey on computation offloading for mobile edge computing information," in *Proc. IEEE 4th Int. Conf. Big Data Secur. Cloud IEEE Int. Conf. High Perform. Smart Comput. IEEE Int. Conf. Intell. Data Secur.*, 2018, pp. 248–251.

[21] M. Afshang, H. S. Dhillon, and P. H. J. Chong, "Modeling and performance analysis of clustered device-to-device networks," *IEEE Trans. Wireless Commun.*, vol. 15, no. 7, pp. 4957–4972, Jul. 2016.

[22] X. Lyu *et al.*, "Selective offloading in mobile edge computing for the green Internet of Things," *IEEE Netw.*, vol. 32, no. 1, pp. 54–60, Jan./Feb. 2018.

[23] Z. Zhou, J. Feng, L. Tan, Y. He, and J. Gong, "An air-ground integration approach for mobile edge computing in IoT," *IEEE Commun. Mag.*, vol. 56, no. 8, pp. 40–47, Aug. 2018.

[24] X. Li, S. Liu, F. Wu, S. Kumari, and J. J. P. C. Rodrigues, "Privacy preserving data aggregation scheme for mobile edge computing assisted IoT applications," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4755–4763, Jun. 2019.

[25] Y. Dai, D. Xu, S. Maharjan, and Y. Zhang, "Joint computation offloading and user association in multi-task mobile edge computing," *IEEE Trans. Veh. Technol.*, vol. 67, no. 12, pp. 12 313–12 325, Dec. 2018.

[26] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 4, pp. 2322–2358, Fourth Quarter 2017.

[27] Q. Guo *et al.* "Minimizing the longest tour time among a fleet of UAVs for disaster area surveillance," *IEEE Trans. Mobile Comput.*, early access, Nov. 16, 2020, doi: 10.1109/TMC.2020.3038156.

[28] T. Salonidis, P. Bhagwat, and L. Tassiulas, "Proximity awareness and fast connection establishment in bluetooth," in *Proc. 1st Annu. Workshop Mobile Ad Hoc Netw. Comput.*, 2000, pp. 141–142.

[29] H. Qin and W. Zhang, "Zigbee-assisted power saving management for mobile devices," *IEEE Trans. Mobile Comput.*, vol. 13, no. 12, pp. 2933–2947, 2013.

[30] C. Singhal and S. De, *Resource allocation next-generation broadband wireless access networks. IGI Global*, 2017.

[31] Y. Hao, M. Chen, L. Hu, M. S. Hossain, and A. Ghoneim, "Energy efficient task caching and offloading for mobile edge computing," *IEEE Access*, vol. 6, pp. 11 365–11 373, 2018.

[32] A. Drexl, "A simulated annealing approach to the multiconstraint zero-one knapsack problem," *Computing*, vol. 40, no. 1, pp. 1–8, 1988.

[33] The primal-dual method for approximation algorithms and its application to network design problems. Accessed: 2009. [Online]. Available: https://math.mit.edu/ goemans/PAPERS/book-ch4.pdf

[34] L. Guo, J. Pang, and A. Walid, "Joint placement and routing of network function chains in data centers," in *Proc. IEEE Conf. Comput. Commun.*, 2018, pp. 612–620.

[35] T. X. Tran and D. Pompili, "Joint task offloading and resource allocation for multi-server mobile-edge computing networks," *IEEE Trans. Veh. Technol.*, vol. 68, no. 1, pp. 856–868, Jan. 2019.

[36] A. Younis, T. X. Tran, and D. Pompili, "Energy-latency-aware task offloading and approximate computing at the mobile edge," in *Proc. IEEE 16th Int. Conf. Mobile Ad Hoc Sensor Syst.*, 2019, pp. 299–307.

[37] C. You, K. Huang, H. Chae, and B.-H. Kim, "Energy-efficient resource allocation for mobile-edge computation offloading," *IEEE Trans. Wireless Commun.*, vol. 16, no. 3, pp. 1397–1411, Mar. 2017.

[38] N. Yu, Q. Xie, Q. Wang, H. Du, H. Huang, and X. Jia, "Collaborative service placement for mobile edge computing applications," in *Proc. IEEE Global Commun. Conf.*, 2018, pp. 1–6.

[39] Google cluster traces. 2020. [Online]. Available: https://github.com/google/cluster-data

[40] Optimization with pulp. 2020. [Online]. Available: https://pypi.org/project/PuLP/

[41] E. Miluzzo, R. Cáceres, and Y.-F. Chen, "Vision: mClouds-computing on clouds of mobile devices," in *Proc. 3rd ACM Workshop Mobile Cloud Comput. Services*, 2012, pp. 9–14.

[42] X. Wang, X. Chen, W. Wu, N. An, and L. Wang, "Cooperative application execution in mobile cloud computing: A Stackelberg game approach," *IEEE Commun. Lett.*, vol. 20, no. 5, pp. 946–949, May 2016.

[43] S. Ranadheera, S. Maghsudi, and E. Hossain, "Mobile edge computation offloading using game theory and reinforcement learning," 2017, *arXiv: 1711.09012*.

[44] S. Ranadheera, S. Maghsudi, and E. Hossain, "Minority games with applications to distributed decision making and control in wireless networks," *IEEE Wireless Commun.*, vol. 24, no. 5, pp. 184–192, Oct. 2017.

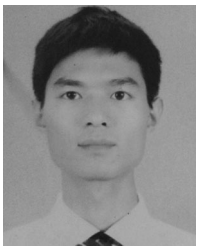
[45] P. A. Apostolopoulos, E. E. Tsiropoulou, and S. Papavassiliou, "Game-theoretic learning-based QoS satisfaction in autonomous mobile edge computing," in *Proc. Global Inf. Infrastructure Netw. Symp.*, 2018, pp. 1–5.

[46] J. Liu and Q. Zhang, "Offloading schemes in mobile edge computing for ultra-reliable low latency communications," *IEEE Access*, vol. 6, pp. 12 825–12 837, 2018.

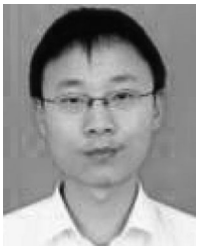
[47] C.-F. Liu, M. Bennis, M. Debbah, and H. V. Poor, "Dynamic task offloading and resource allocation for ultra-reliable low-latency edge computing," *IEEE Trans. Commun.*, vol. 67, no. 6, pp. 4132–4150, Jun. 2019.

[48] D. Wang, G. Joshi, and G. Wornell, "Efficient task replication for fast response times in parallel computation," in *Proc. ACM Int. Conf. Meas. Model. Comput. Syst.*, 2014, pp. 599–600.

[49] Y. Sun, C. E. Koksal, and N. B. Shroff, "On delay-optimal scheduling in queueing systems with replications," 2016, *arXiv:1603.07322*.

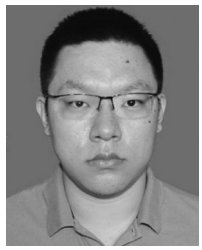


Haibo Wang (Student Member, IEEE) received the BE degree in nuclear science and the master's degree in computer science from the University of Science and Technology of China, Hefei, China, in 2016 and 2019, respectively. He is currently working toward the PhD degree in computer and information science and engineering at the University of Florida, Gainesville, Florida. His main research interest include Internet traffic measurement, software defined networks, and optical circuit scheduling.



Hongli Xu (Member, IEEE) received the BS degree in computer science from the University of Science and Technology of China (USTC), Hefei, China, in 2002, and the PhD degree in computer software and theory from USTC, Hefei, China, in 2007. He is currently a professor at the School of Computer Science and Technology, University of Science and Technology of China. He was awarded the Outstanding Youth Science Foundation of NSFC in 2018. He has won the Best Paper Award or the Best Paper Candidate in several famous conferences. He has published more than 100 papers in famous journals and conferences, including the *IEEE/ACM Transactions on Networking*, the *IEEE Journal on Selected Areas in Communications*, the *IEEE Transactions on Mobile Computing*, the *IEEE Transactions on Parallel and Distributed Systems*, Infocom and ICNP, etc. He has also held more than 30 patents. His main research interests include software defined networks, edge computing, and Internet of Things.

He Huang (Member, IEEE) received the PhD degree from the Department of Computer Science and Technology, University of Science and Technology of China, Hefei, China, in 2011. He is currently a professor at the School of Computer Science and Technology, Soochow University, Suzhou, China. He is also with the Anhui Provincial Key Laboratory of Network and Information Security, Wuhu, China. His current research interests include network traffic measurement, spectrum auctions, privacy preserving, crowdsourcing, software defined networks, and satellite networks. He is also a member of the ACM.



Min Chen received the bachelor's degree in software engineering from the Xi'an Jiaotong University, Xi'an, China, in 2018. He is currently working toward the master's degree at the School of Software Engineering, University of Science and Technology of China, Hefei, China. His research interests include edge computing and distributed machine learning.

Shigang Chen (Fellow, IEEE) received the BS degree in computer science from the University of Science and Technology of China, Hefei, China, in 1993, and the MS and PhD degrees in computer science from the University of Illinois at Urbana-Champaign, Champaign, Illinois, in 1996 and 1999, respectively. He was with Cisco Systems, San Jose, California, where he was involved with network security for 3 years and helped start up the network security company, Protego Networks, Milpitas, California. He joined the University of Florida, Gainesville, Florida, as an assistant professor in 2002, and became an associate professor in 2008, and where he has been a professor at the Department of Computer and Information Science and Engineering since 2013. He has authored or coauthored 190 peerreviewed journal/conference papers and holds 12 U.S. patents. He was a recipient of the IEEE Communications Society Best Tutorial Paper Award in 1999, the NSF CAREER Award in 2007, and the Cisco University Research Award in 2007 and 2012. He holds the University of Florida Research Foundation Professorship in 2017-2020 and the University of Florida Term Professorship in 2017-2020. He is also an ACM distinguished member and an IEEE ComSoc distinguished lecturer.