An Application Placement Technique for Concurrent IoT Applications in Edge and Fog Computing Environments

Mohammad Goudarzi, *Member, IEEE*, Huaming Wu, *Member, IEEE*, Marimuthu Palaniswami, *Fellow, IEEE*, and Rajkumar Buyya, *Fellow, IEEE*

Abstract—Fog/Edge computing emerges as a novel computing paradigm that harnesses resources in the proximity of the Internet of Things (IoT) devices so that, alongside with the cloud servers, provide services in a timely manner. However, due to the ever-increasing growth of IoT devices with resource-hungry applications, fog/edge servers with limited resources cannot efficiently satisfy the requirements of the IoT applications. Therefore, the application placement in the fog/edge computing environment, in which several distributed fog/edge servers and centralized cloud servers are available, is a challenging issue. In this article, we propose a weighted cost model to minimize the execution time and energy consumption of IoT applications, in a computing environment with multiple IoT devices, multiple fog/edge servers and cloud servers. Besides, a new application placement technique based on the Memetic Algorithm is proposed to make batch application placement decision for concurrent IoT applications. Due to the heterogeneity of IoT applications, we also propose a lightweight pre-scheduling algorithm to maximize the number of parallel tasks for the concurrent execution. The performance results demonstrate that our technique significantly improves the weighted cost of IoT applications up to 65 percent in comparison to its counterparts.

Index Terms—Fog computing, edge computing, Internet of Things (IoT), application placement, optimization, application partitioning

1 Introduction

Due to recent advances in hardware and software technologies, the number of Internet of Things (IoT) devices (e.g., smartphones, smart cameras, smart vehicles, etc) has significantly increased, so that IoT devices and their applications have become pervasive in modern digital society. However, the IoT paradigm, in which heterogeneous devices can connect and communicate together, generates a huge amount of data that needs processing and storage. According to Cisco, it is anticipated that by 2030, approximately 500 billion IoT devices will be connected to the Internet [1]. In addition, the number of real-time and latency-sensitive applications such as smart transportation, smart health-care, augmented reality, and smart buildings requiring large amounts of computing and network resources has increased significantly [2]. Moreover, performing such resource-hungry applications requires a considerable amount of energy to be consumed, which significantly affects the performance of IoT devices such as mobile devices and sensors, due to their limited battery lifetime.

- *M. Goudarzi and R. Buyya are with the Cloud Computing and Distributed Systems (CLOUDS) Laboratory, School of Computing and Information Systems, The University of Melbourne, Parkville, VIC 3010, Australia. E-mail: mgoudarzi@student.unimelb.edu.au, rbuyya@unimelb.edu.au.*
- *M. Palaniswami is with the Department of Electrical and Electronic Engineering, The University of Melbourne, Parkville, VIC 3010, Australia. E-mail: palani@unimelb.edu.au.*
- *H. Wu is with the Center for Applied Mathematics, Tianjin University, Tianjin 300072, China. E-mail: whming@tju.edu.cn.*

Manuscript received 2 Sept. 2019; revised 26 Nov. 2019; accepted 10 Jan. 2020. Date of publication 15 Jan. 2020; date of current version 4 Mar. 2021. (Corresponding author: Mohammad Goudarzi.) Digital Object Identifier no. 10.1109/TMC.2020.2967041

As a centralized solution, the cloud computing pmaradigm is one of the main enablers of the IoT, in which unlimited and elastic resources are available to execute IoT's computation-intensive applications. The execution time of IoT applications and IoT devices' energy consumption can be reduced by offloading (i.e., application/task placement) all/some of their computation-intensive tasks to different cloud servers [3]. However, IoT devices suffer from low bandwidth and high latency when communicating with cloud servers. These latter are mainly because IoT devices are connected to the cloud servers via Wide Area Network (WAN) which provides low bandwidth, and the far distance of cloud servers and IoT devices which leads to high latency [4]. Besides, the huge amount of incoming data to the cloud servers and resource-hungry nature of emerging IoT applications requiring more computing and storage resources, lead to congestion in the cloud servers. Hence, not only the cloud servers cannot efficiently satisfy the requirements of emerging resource-hungry IoT applications, but also they may incur more energy consumption for IoT devices due to their low bandwidth.

To reduce the huge amount of incoming data to the cloud servers, and alleviate the high latency and low bandwidth problem, a new computing paradigm, called Fog Computing has emerged. It provides an intermediate computing layer between cloud servers and IoT devices in which several heterogeneous fog servers are distributed. These fog servers have fewer resources (e.g., CPU, RAM) in comparison to cloud servers, while they provide higher bandwidth with less latency for IoT devices since they can be accessed via Local Area Network (LAN) [4], [5]. In our view, edge computing

harnesses only edge resources while fog computing harnesses both edge and cloud resources (although some of the works use these terms interchangeably). Considering the potential of fog computing, IoT devices can perform their resource-hungry and latency-sensitive applications with improved Quality of Service (QoS) by offloading all/some of their applications to fog or cloud servers based on their QoS requirements [6], [7]. It also leads to less congestion in cloud servers since distributed fog servers can ease the burden of cloud servers for processing and storage of incoming data from IoT devices. However, considering the large number of heterogeneous IoT devices whose applications require various level of QoS, it is challenging to decide whether the execution of such applications on remote servers (whether fog or cloud servers) is beneficial or not. Besides, the ever-increasing number of IoT devices causes more requests to be forwarded to the fog servers, which may incur congestion due to their limited resources. This latter may result in more execution time and energy consumption for IoT devices.

To address the aforementioned issues, we propose an efficient application placement technique to jointly optimize the execution time and energy consumption of IoT devices in an environment with multiple heterogeneous cloud and fog servers. The main contributions of this paper are as follows.

- We propose a weighted cost model for application placement of multiple IoT devices to minimize their execution time and energy consumption.
- We put forward a dynamic and lightweight pre-scheduling technique to maximize the number of parallel tasks for execution. Considering the NP-Complete nature of application placement in fog computing environments, we propose an optimized version of the Memetic Algorithm (MA) to achieve a well-suited solution in reasonable decision time.
- We embed a fast failure recovery method in our technique to assign failed tasks to appropriate servers in a timely manner.

The rest of the paper is organized as follows. Relevant work of application placement techniques in fog computing environments is discussed in Section 2. The system model and problem formulations are presented in Section 3. Section 4 presents our proposed applications placement technique. We evaluate the performance of our technique and compare it by the state-of-the-art techniques in Section 5. Finally, section 6 concludes the paper and draws future works.

2 Related Work

In this section, related works for application placement techniques in fog computing environments are discussed, where cloud and fog servers work collaboratively to satisfy the IoT application requirements. They are divided into independent and dependent categories based on the dependency mode of their IoT applications' constituent parts (e.g., tasks). Each IoT application can be modeled as a set of independent or dependent tasks. The dependent one refers to applications consisted of several dependent tasks so that each new task runs only when its predecessor tasks are completely performed. However, in the independent one, the applications' tasks do not have such constraints for execution.

2.1 Independent Tasks

Huang *et al.* [8] proposed a task placement algorithm where multiple mobile devices offload their independent tasks to multiple edge servers and one cloud server. In this technique, each mobile device decides whether each task should be offloaded or not, and in case of offloading, which edge or cloud server is suited for execution of each task. An energy-aware cloudlet selection technique was proposed in [9] to meet the latency requirement of incoming tasks from one IoT device. Haber *et al.* [10] proposed an offloading algorithm deployed in the cloud layer, aiming at minimizing the energy consumption of several mobile devices while satisfying the latency requirements of mobile applications. It is obtained by optimizing mobile devices' transmission power and the assigned server computation. An offloading algorithm based on the Lyapunov optimization was proposed in [11] to reduce the execution time of IoT applications by offloading the task to either the single fog server or one cloud server. Mahmud *et. al.* [12] proposed a Quality of Experience (QoE)-aware application placement technique in which independent tasks of IoT devices are placed in the fog or cloud servers. Chen *et al.* [13] considered a multi-user environment with a single computing access point and a remote cloud server, in which the independent tasks of mobile users can be processed locally, at the computing access point, or the cloud server. Hong *et al.* [14] proposed a game-theoretic approach for computation offloading, and multi-hop cooperative-messaging mechanism for IoT devices. It considers that each IoT device decides either to forward its single task to the fog or cloud server if it has access to wireless networks or to collaborate with other IoT devices that have access to wireless networks for forwarding its task.

2.2 Dependent Tasks

In the dependent category, related works modeled their applications by Directed Acyclic Graph (DAG) in which each vertex represents one task of IoT application, and each edge shows data flow (i.e., dependency) between two tasks.

Neto *et al.* [15] and Wu *et al.* [16] proposed a partitioning algorithm for a single mobile device to offload their computation-intensive tasks to a single edge or cloud server. The placement engine of these proposal are placed at the mobile device aiming at finding a group of tasks for offloading, by which the execution time of mobile application and energy consumption of mobile device become reduced. The main goal of [17], [18] is to minimize the execution time of IoT applications in an environment in which multiple fog servers and a cloud server are accessible for the application placement. Lin *et al.* [17] considered only one mobile device in its system model for offloading, while Stavrinides *et al.* [18] attempted to place tasks of multiple users requiring low communication overhead at the cloud server and those tasks that have more communication overhead at the edge layer. Mahmud *et al.* [19] proposed a latency-aware application placement policy in an environment with multiple fog servers and a single cloud server. Although the above-mentioned techniques consider task placement as their main objective, Bi *et al.* [20] proposed a solution for joint optimization of service caching placement and computation offloading.

The proposed placement engines in the aforementioned works made application placement decisions for different users at different time slots, or only consider a fraction of a

TABLE 1
The Qualitative Comparison of the Current Literature

Techniques	IoT Application Properties			Architectural Properties								Placement Engine Properties				
	Dependency Mode	Task Number	Heterogeneity	IoT Device		Edge Layer			Cloud Layer			Position	Batch Placement	Decision Parameters		
				Number	Request Number	Fog Number	Cooperation	Heterogeneity	Cloud Number	Cooperation	Heterogeneity			Time	Energy	Weighted
[8]	Independent	Multiple	✓	Multiple	Different	Multiple	×	×	Single	×	×	IoT device	No	✓	✓	✓
[9]		Single	✓	Single	Same	Multiple	×	✓	Single	×	×	Edge Layer		✓	✓	×
[10]		Single	✓	Multiple	Same	Multiple	×	×	Single	×	×	Cloud Layer		✓	✓	✓
[11]		Single	✓	Multiple	_	Single	×	×	Single	×	×	Edge Layer		✓	×	×
[12]		Single	✓	Multiple	Same	Multiple	×	✓	Single	×	×	Edge Layer		✓	×	×
[13]		Multiple	✓	Multiple	Same	Single	×	×	Single	×	×	Edge Layer		✓	✓	✓
[14]		Single	✓	Multiple	Same	Multiple	✓	✓	Single	×	×	Edge Layer		✓	✓	✓
[15]	Dependent	Multiple	✓	Single	Same	Single	×	_	Single	×	×	IoT Device		✓	✓	✓
[16]		Multiple	✓	Single	Same	Single	×	_	Single	×	×	IoT Device		✓	✓	✓
[18]		Multiple	✓	Single	Same	Multiple	✓	✓	Single	×	×	Edge Layer		✓	×	×
[20]		Multiple	✓	Single	Same	Single	×	×	_	×	×	_		✓	✓	✓
[17]		Multiple	✓	Multiple	_	Multiple	×	×	Single	×	×	Edge Layer		✓	×	×
[19]		Multiple	✓	Multiple	Different	Multiple	✓	✓	Single	×	×	Edge Layer		✓	×	×
[4]		Multiple	×	Multiple	Different	Single	×	×	Single	×	×	Edge Layer	Yes	✓	✓	✓
Our Technique		Multiple	✓	Multiple	Different	Multiple	✓	✓	Multiple	✓	✓	Edge Layer		✓	✓	✓

whole of each user's tasks at each time slot. However, Xu *et al.* [4] proposed a batch task placement based on Genetic Algorithm (GA), in which mobile applications of multiple users are forwarded to the single central edge server for application placement decision.

2.3 A Qualitative Comparison

Table 1 identifies and compares key elements of related works with ours in terms of their IoT application, architectural, and placement engine properties. In the IoT application section, the dependency mode of each proposal is studied which can be either independent or dependent. Moreover, we study how each proposal modeled IoT application in terms of the number of tasks and heterogeneity. This latter demonstrates whether IoT applications consist of homogeneous or heterogeneous tasks in terms of their computation and data flow. In the architectural section, the attributes of IoT devices, fog/edge servers, and cloud servers are studied. For IoT devices, the overall number of devices and their type of requests are identified. The different request number shows that each device has a different number of requests compared to other IoT devices. In the fog and cloud layers, the number of fog and cloud servers, the cooperation between different fog/cloud servers, and the heterogeneity in terms of servers' specifications are identified, respectively. The position of placement engine, the capability of batch placement, and decision parameters are also studied in the placement engine section.

Considering application placement techniques proposed for fog computing environments, this work proposes a batch application placement technique for an environment consisting of multiple devices in the IoT layer, multiple fog/edge servers in the edge layer, and multiple cloud servers in the cloud layer. To the best of our knowledge, this is the only work that considers the aforementioned fog computing environment and proposes a weighted cost model to jointly minimize the execution time of IoT applications and energy consumption of IoT devices. Our weighted cost model not only can be applied for our general fog computing environment, but it also can be used for simpler fog computing environments with a single IoT device, single fog server, single cloud server, or any combination thereof. In addition, it is important to note that the IoT applications are considered as heterogeneous DAGs (i.e., workflows) with a different number of tasks and data flows. Hence, we propose a lightweight pre-scheduling algorithm to organize incoming tasks of different DAGs, so that the number of tasks for parallel execution becomes maximized. Then, an optimized version of the Memetic Algorithm (MA) is proposed to perform application placement in a timely manner.

3 System Model and Problem Formulation

We consider a framework consisting of multiple IoT devices, multiple fog (i.e., edge) servers, multiple cloud servers, and brokers, in which IoT devices can locally execute their workflows (i.e., DAGs) or completely/partially place them on cloud servers and/or fog servers for execution. Fig. 1 represents an overview of our system model.

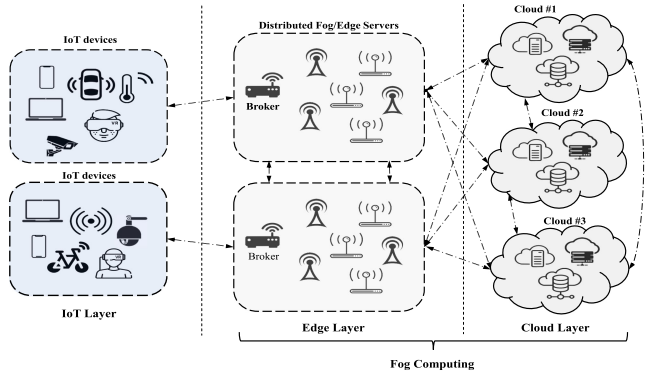


Fig. 1. An overview of our system model.

In this system framework, each broker supports up to N IoT devices, which are distributed in its proximity. The broker (which can be a fog server) receives workflows from different IoT devices, and periodically makes task placement decisions based on the requirements of IoT applications and the current status of the network. According to the result of application placement decisions, each IoT device understands to which server each constituent part of its workflow should be sent, or it should be executed locally on the IoT device.

3.1 Application Workflow

Each IoT application can be partitioned based on different levels of granularity such as class and task, just to mention a few [21]. Without loss of generality, we represent the application running on the nth IoT device as a DAG (i.e., workflow) of its tasks $G_n = (\mathcal{V}_n, \mathcal{E}_n), \forall n \in \{1, 2, \ldots, N\}$, where $\mathcal{V}_n = \{v_{n,i} | 1 \leq i \leq |\mathcal{V}_n|\}$ denotes the set of tasks running on the nth IoT device, and $\mathcal{E}_n = \{e_{n,i,j} | v_{n,i}, v_{n,j} \in \mathcal{V}_n,\ i \neq j\}$ illustrates the set of data flows between tasks. As an illustration, $e_{n,i,j}$ represents the dependency between $v_{n,i}$ and $v_{n,j}$ of the application running on the nth IoT device.

Considering the number of instructions for each task $v_{n,i}$, its corresponding weight is represented as $v^w_{n,i}$. Besides, the associated weight of each edge $e^w_{n,i,j}$ shows the amount of data that the task $v_{n,j}$ receives as an input from $v_{n,i}$. Since IoT applications are modeled as DAGs, each task $v_{n,i}$ cannot be executed unless all its predecessor tasks, denoted as $\mathcal{P}(v_{n,i})$ finish their execution.

3.2 Problem Formulation

We formulate the task placement problem as an optimization problem aiming at minimizing the overall execution time of IoT applications and energy consumption of IoT devices.

Since different servers are available to run each task $v_{n,i}$, the set of all available servers is represented as $\mathcal{S}$ with $|\mathcal{S}| = M$. The $\mathcal{S}^{y,z}$ represents one server, in which y represents the type of server (the IoT device ($y = 0$), fog servers ($y = 1$), cloud servers ($y = 2$)) and z denotes that server's index. The offloading configuration of the workflow belonging to the nth IoT device is represented as X_n, and the $x_{n,i}$ denotes the offloading configuration for each task $v_{n,i}$, which is obtained from the following criteria:

$$x_{n,i} = \begin{cases} 0, & s^{y,z} = s^{0,n}, \\ 1, & s^{y,z} \in \{s^{1,1}, s^{1,2}, \cdots, s^{1,f}\} \quad |z| = f, \\ 2, & s^{y,z} \in \{s^{2,1}, s^{2,2}, \cdots, s^{2,c}\}, \quad |z| = c \end{cases} \tag{1}$$

where $x_{n,i} = 0$ depicts that the ith task is assigned to the nth IoT device ($s^{0,n}$) for local execution, and $x_{n,i} = 1$ and $x_{n,i} = 2$ denote that the ith task is assigned to one of fog servers and cloud servers, respectively, for the remote execution. Moreover, the f and c show the number of available fog servers and cloud servers respectively.

3.2.1 Weighted Cost Model

The goal of the task placement technique is to find the best possible configuration of available servers for each IoT application so that the weighted cost of execution for each IoT device becomes minimized, as depicted in the following:

$$\min_{\psi_\gamma, \psi_\theta \in [0,1]} \Psi(X_n), \quad \forall n \in \{1, 2, \ldots, N\}, \tag{2}$$

where

$$\Psi(X_n) = \psi_\gamma \times \frac{\Gamma(X_n)}{\Gamma_{Loc_n}} + \psi_\theta \times \frac{\Theta(X_n)}{\Theta_{Loc_n}}, \tag{3}$$

s.t.

$$C1:\ VM_{fog,i} \leq C_{fog,i},\ \forall i \in \{\mathcal{S}^{1,1}, \ldots, \mathcal{S}^{1,f}\} \tag{4}$$

$$C2:\ |x_{n,i}| = 1,\ \forall n \in \{1, 2, \ldots, N\}, 1 \leq i \leq |\mathcal{V}_n| \tag{5}$$

$$C3:\ \Psi(\mathcal{P}(v_{n,i})) \leq \Psi(\mathcal{P}(v_{n,i}) + v_{n,i}), \tag{6}$$

where $\Gamma(X_n)$, $\Theta(X_n)$, Γ_{Loc_n}, and Θ_{Loc_n} demonstrate the execution time, energy consumption, local execution time and local energy consumption of the nth IoT device's workflow, respectively. Besides, ψ_γ and ψ_θ are control parameters for execution time and energy consumption, by which the weighted cost model can be tuned according to the users' requirements. Moreover, we assume that each task can be exactly assigned to one Virtual Machine (VM) of one fog or cloud server. $C1$ denotes that the number of instantiated VMs of the ith fog server $VM_{fog,i}$ is less or equal to the maximum capacity of that fog server $C_{fog,i}$. $C2$ represents that each task i belonging to the workflow of nth IoT device can only be assigned to one server in each time slot. In addition, $C3$ indicates that the predecessor tasks of $v_{n,i}$ should be executed before the execution of the task $v_{n,i}$.

3.2.2 Execution Time Model

Considering the Eq. (3), the weighted cost optimization is equal to the execution time model when $\psi_\gamma = 1$ and $\psi_\theta = 0$.

The goal of execution time optimization model is to find the optimal configuration of the application running on the nth IoT device so that the execution time of that application decreases. The overall execution time of each candidate configuration can be defined as the sum of latency in task offloading ($\Gamma^{lat}_{X_n}$), the computing time of workflow's tasks based on their assigned servers ($\Gamma^{exe}_{X_n}$) and the data transmission time between each pair of dependent tasks in each workflow ($\Gamma^{tra}_{X_n}$), as depicted in the following:

$$\Gamma(X_n) = \Gamma^{exe}_{X_n} + \Gamma^{lat}_{X_n} + \Gamma^{tra}_{X_n}. \tag{7}$$

The computing execution time that corresponds to the application running on the nth IoT device is calculated by:

$$\Gamma^{exe}_{X_n} = \sum_{x_{n,i} \in X_n} \gamma^{exe}_{x_{n,i}}, \tag{8}$$

where $\gamma^{exe}_{x_{n,i}}$ shows the computing time of task $v_{n,i}$, and is calculated based on its corresponding assigned server from the following equation:

$$\gamma^{exe}_{x_{n,i}} = \begin{cases} \frac{v^w_{n,i}}{loc^{cpu}}, & x_{n,i} = 0 \\ \frac{v^w_{n,i}}{SF^f \times loc^{cpu}}, & x_{n,i} = 1, \\ \frac{v^w_{n,i}}{SF^c \times loc^{cpu}}, & x_{n,i} = 2 \end{cases} \tag{9}$$

where loc^{cpu} demonstrates the computing power of the IoT device, and SF^f and SF^c denote the speedup factor of fog servers and cloud servers, respectively.

The offloading latency $\Gamma^{lat}_{X_n}$ of tasks corresponding to the nth IoT device is calculated based on tasks' assigned servers:

$$\Gamma^{lat}_{X_n} = \sum_{x_{n,i} \in X_n} \gamma^{lat}_{x_{n,i}}, \tag{10}$$

where $\gamma^{lat}_{x_{n,i}}$ illustrates the offloading latency of task $v_{n,i}$, and is calculated according to its corresponding assigned server from the following equation:

$$\gamma^{lat}_{x_{n,i}} = \begin{cases} 0, & x_{n,i} = 0 \\ L_{LAN}, & x_{n,i} = 1 \\ L_{WAN}, & x_{n,i} = 2 \end{cases}, \tag{11}$$

where L_{LAN} and L_{WAN} correspond to the latency of LAN and WAN respectively.

The tasks' transmission time of the workflow corresponding to the nth IoT device is calculated by:

$$\Gamma^{tra}_{X_n} = \sum_{e_{n,i,j} \in \mathcal{E}_n} \gamma^{tra}_{e_{n,i,j}}, \tag{12}$$

where the transmission time of each pair of dependent tasks $v_{n,i}$ and $v_{n,j}$ is calculated as follows:

$$\gamma^{tra}_{e_{n,i,j}} = \begin{cases} \frac{e^w_{n,i,j}}{B_{LAN}}, & CT_i = CT_1, CT_3 \\ \frac{e^w_{n,i,j}}{B_{WAN}}, & CT_i = CT_2, CT_4 \\ 0, & CT_i = CT_5 \end{cases}, \tag{13}$$

where B_{LAN} and B_{WAN} stand for the bandwidth (i.e., data rate) of LAN and WAN respectively. The CT_i represents possible transmission configuration for each edge $e_{n,i,j}$ according to the assigned servers of its tasks $v_{n,i}$ and $v_{n,j}$ to calculate transmission time. The possible transmission configurations are defined as:

$$CT_i(e^w_{n,i,j}) = \begin{cases} x_{n,i} \oplus x_{n,j} = 0 \\ \& \quad x_{n,i} = 1 \\ \& SI(v_{n,i}) \oplus SI(v_{n,j}) \neq 0 & i = 1 \\ x_{n,i} \oplus x_{n,j} = 0 \\ \& \quad x_{n,i} = 2 \\ \& SI(v_{n,i}) \oplus SI(v_{n,j}) \neq 0 & i = 2 \\ x_{n,i} \oplus x_{n,j} = 1, & i = 3 \\ x_{n,i} \oplus x_{n,j} > 1, & i = 4 \\ x_{n,i} \oplus x_{n,j} = 0 \\ \& SI(v_{n,i}) \oplus SI(v_{n,j}) = 0, & i = 5 \end{cases}, \tag{14}$$

where $\oplus$ is XOR binary operation and $SI(v_{n,i})$ is a function that returns the assigned server's index (i.e., z) of ith task belonging to the nth workflow. CT_1 denotes that the invocation is between two tasks $v_{n,i}$ and $v_{n,j}$ that are assigned to two different fog servers, and CT_2 represents the configuration in which the two tasks run on two different cloud servers. The invocation between two tasks assigned to the IoT device and one of fog server is depicted in CT_3. CT_4 is used to show two different configurations. The first one is whenever the two tasks are assigned to the IoT device and one of the cloud servers, while the second one illustrates that one task is assigned to one of the cloud servers and the second task is assigned to one of the fog servers. Finally, CT_5 refers to the condition that two tasks are assigned exactly to the same server, for which the transmission time is equal to zero.

3.2.3 Energy Consumption Model

According to Eq. (2), the weighted cost optimization is equal to the energy consumption model when $\psi_\gamma = 0$ and $\psi_\theta = 1$. The energy consumption model aims at finding the best-possible configuration of the application's tasks to minimize the energy consumption of the nth IoT device.

The overall energy consumption of each candidate configuration can be defined as the sum of energy consumed in task offloading ($\Theta^{lat}_{X_n}$), the energy consumption for the computing of tasks ($\Theta^{exe}_{X_n}$), and the energy consumed for the data transmission between each pair of dependent tasks ($\Theta^{tra}_{X_n}$) of that application, as depicted in the following:

$$\Theta(X_n) = \Theta^{exe}_{X_n} + \Theta^{lat}_{X_n} + \Theta^{tra}_{X_n}. \tag{15}$$

The amount of energy consumed to compute the application belonging to the nth IoT device is defined as follows:

$$\Theta^{exe}_{X_n} = \sum_{x_{n,i} \in X_n} \theta^{exe}_{x_{n,i}}, \tag{16}$$

where $\theta^{exe}_{x_{n,i}}$ represents the energy consumption required to compute the task $v_{n,i}$, as calculated in the following:

$$\theta^{exe}_{x_{n,i}} = \begin{cases} \gamma^{exe}_{x_{n,i}} \times P_{cpu}, & x_{n,i} = 0 \\ \gamma^{idle}_{x_{n,i}} \times P_{idle}, & x_{n,i} = 1, 2 \end{cases}, \tag{17}$$

where P_{cpu} is the CPU power of the IoT device on which the task $v_{n,i}$ runs. Since we only consider the energy consumption from IoT device perspective, whenever each task is offloaded to the fog servers ($x_{n,i} = 1$) or cloud servers ($x_{n,i} = 2$), the respective energy consumption is equal to the idle time of the IoT device $\gamma^{idle}_{x_{n,i}}$ multiplied to the power consumption of that device in its idle mode P_{idle}.

The energy consumed to offload applications' tasks belonging to the nth IoT device $\Theta^{lat}_{X_n}$ is calculated by:

$$\Theta^{lat}_{X_n} = \sum_{x_{n,i} \in X_n} \theta^{lat}_{x_{n,i}}, \tag{18}$$

where $\theta^{lat}_{x_{n,i}}$ stands for the offloading energy consumption of the task $v_{n,i}$ and is obtained from:

$$\theta^{lat}_{x_{n,i}} = \begin{cases} 0, & x_{n,i} = 0 \\ \gamma^{lat}_{x_{n,i}} \times P_{idle}, & x_{n,i} = 1, 2 \end{cases}. \tag{19}$$

The transmission energy consumption $\Theta^{tra}_{X_n}$ corresponding to the nth IoT device is obtained from:

$$\Theta^{tra}_{X_n} = \sum_{x_{n,i} \in X_n} \theta^{tra}_{x_{n,i}}, \tag{20}$$

where the transmission energy between each pair of dependent tasks $v_{n,i}$ and $v_{n,j}$ is calculated as follows:

$$\theta^{tra}_{e_{n,i,j}} = \begin{cases} \frac{e^w_{n,i,j}}{B_{LAN}} \times P_{transfer}, & CE_i = CE_1 \\ \frac{e^w_{n,i,j}}{B_{WAN}} \times P_{transfer}, & CE_i = CE_2 \\ 0, & CE_i = CE_3 \end{cases}, \tag{21}$$

where the transmission power of the IoT device is denoted as $P_{transfer}$, and the CE_i shows transmission configuration for each edge $e_{n,i,j}$ based on the assigned servers of its tasks to calculate the transmission energy, which is calculated from:

$$CE_i(e^w_{n,i,j}) = \begin{cases} x_{n,i} \oplus x_{n,j} = 1, & i = 1 \\ x_{n,i} \oplus x_{n,j} = 2, & i = 2 \\ otherwise, & i = 3 \end{cases}, \tag{22}$$

where CE_1 denotes that the data flow is between two tasks $v_{n,i}$ and $v_{n,j}$ that are assigned to the IoT device and fog servers. Moreover, CE_2 is used to represent the invocation between two tasks that are assigned to IoT device and cloud servers. Because the energy consumption is considered from the IoT device perspective, the transmission energy consumption is equal to zero whenever one of the participating tasks in each edge $e^w_{n,i,j}$ is not assigned to the IoT device, as represented in CE_3.

4 A New Application Placement Technique

Our proposed application placement technique is divided into three phases: pre-scheduling, batch application placement, and failure recovery. In the pre-scheduling phase, an algorithm is proposed by which brokers can organize the concurrent IoT devices' workflows. Next, we propose an optimized version of Memetic Algorithm (MA) for batch application placement to minimize the weighted cost of each IoT device. Beside, to overcome any potential failures in the runtime, we embed a lightweight failure recovery method in our technique.

4.1 Pre-Scheduling Phase

The broker receives concurrent workflows from IoT devices in its decision time slot and organizes them based on their respective dependencies. Moreover, it calculates the local execution time and energy consumption of IoT devices based on their respective workflows.

Workflows of IoT devices are heterogeneous in terms of the number and weight of tasks, dependencies, and the amount of dataflow between each pair of dependent tasks. Moreover, the order of execution of tasks in each workflow should be sorted so that a new task $v_{n,i}$ cannot be executed unless all tasks in its $\mathcal{P}(v_{n,i})$ finish their execution.

4.1.1 Algorithmic Process

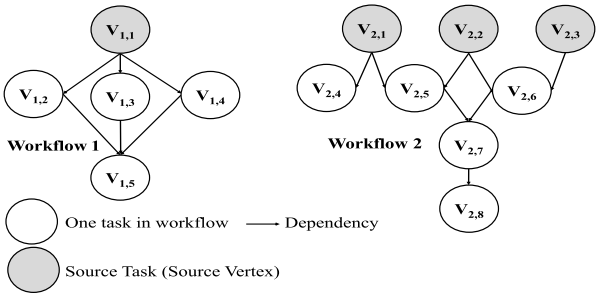
Algorithm 1 demonstrates how the pre-scheduling phase organizes tasks of each workflow and accordingly creates a list of schedules of concurrent workflows. In Algorithm 1, for each workflow, the local execution time and energy consumption are calculated and stored in $LocTime$ and $LocEnergy$, respectively (lines 3 and 4). Since DAGs can have several root vertices (i.e., source nodes), the $RootFinder$ method finds all the root vertices of each DAG and stores them in $Source_n$ (line 5). This method checks whether the $\mathcal{P}(v_{n,i})$ is equal to $null$ or not for each task i in the nth workflow, and if it equals to $null$ returns that task as one source root. The $SingleRootTransformer$ method receives the WF_n and $Source_n$ and creates a new DAG, called $DAG^{\star}_n$, in which the workflow has only a single source root (line 6). To obtain this, we create a dummy vertex (called $DummyRoot_n$) and connect this vertex to all source vertices of $Source_n$ obtained from the original DAG. This enables us to run Breadth-First-Search (BFS) algorithm over $DAG^{\star}_n$ starting from the $DummyRoot$, by which we can specify scheduling number for each vertex (i.e., BFS level of each vertex) (line 7). The main outcome of the first loop (lines 2-8) of this algorithm is providing a schedule number for tasks of each workflow, by which the concurrent tasks of each workflow are specified. Because our proposed batch application placement algorithm concurrently decides for several workflows at each time slot, it is required to combine these workflows based on their respective schedule number. To achieve this, the algorithm iterates over all workflows, so that tasks with same schedule number (either from same or different workflows) are stored in the respective row of a 2D Arraylist called $FinalOrderedList$. The $get(x)$ and $add(v_{n,i})$ methods are used to access a row in the 2D Arraylist (i.e., one schedule), and to add a new entry to a list, respectively (line 12).

Algorithm 1. Pre-Scheduling Phase

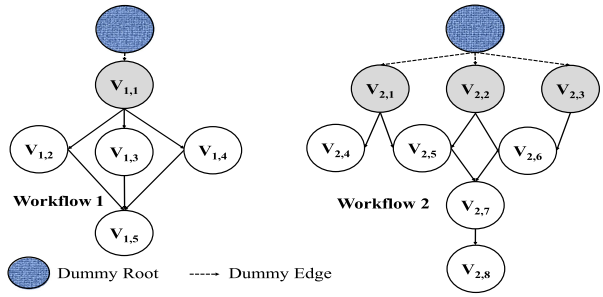
Input: WF: List of all workflows
Output: $FinalOrderedList$, $LocTime$, $LocEnergy$
/* N: Number of workflows, WF_n: The nth workflow in the WF, $LocTime$ & $LocEnergy$: Lists storing local execution time and energy consumption of workflows, $FinalOrderedList$: A 2D Arraylist in which tasks in each row can be executed in parallel */
1 $N = |WF|$
2 **for** $n = 1$ *to* N **do**
3 $LocTime$.add(CalLocalExeTime(WF_n))
4 $LocEnergy$.add(CalLocalExeEnergy(WF_n))
5 $Source_n$ = RootFinder(WF_n)
6 $DAG^{\star}_n$ = SingleRootTransformer(WF_n, $Source_n$)
7 BFS($DAG^{\star}_n$, $DummyRoot_n$)
8 **end**
9 **for** $n = 1$ *to* N **do**
10 **for** $i = 1$ *to* $|WF_n|$ **do**
11 integer x = CheckOrderNumber($v_{n,i}$)
12 $FinalOrderedList$.get(x).add($v_{n,i}$)
13 **end**
14 **end**

4.1.2 Example

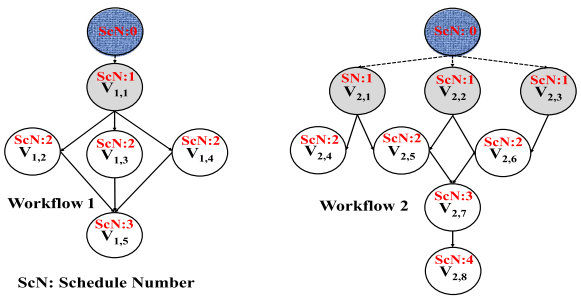
Fig. 2 demonstrates how this pre-scheduling phase works. Fig. 2a represents two workflows with five and eight vertices. The first workflow has one source vertex while the second workflow has three source vertices (represented by gray color). After identifying the source vertices, the $SingleRootTransformer$ method creates a $DAG^{\star}_n$ with single source vertex, as depicted in Fig. 2b. Next, the BFS algorithm is applied on the $DAG^{\star}_n$ to specify the schedule number for each task as depicted in Fig. 2c. This latter helps to identify how many tasks can be executed in parallel in each



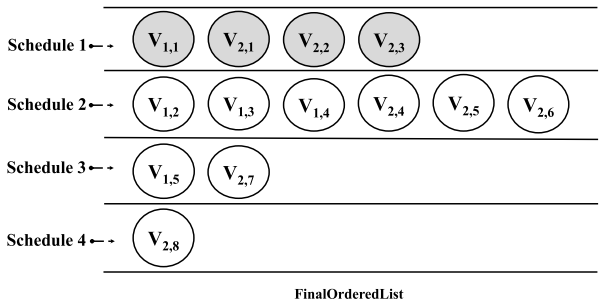
(a) Different workflows with identified source vertices



(b) Transforming workflows to single root DAG_n^*



(c) Assigning schedule numbers to tasks based on BFS



(d) List of scheduled tasks of concurrent workflows

Fig. 2. An example demonstrating the pre-scheduling phase.

schedule. When the schedule number of all tasks in all workflows are identified, the tasks with the same schedule numbers are placed together in a 2D Arraylist (called $FinalOrderedList$) as depicted in Fig. 2d.

4.2 Batch Application Placement Phase

We propose a batch application placement algorithm in which a Memetic Algorithm (MA) is employed to make placement decisions for tasks of each schedule. Because tasks in each schedule are either independent tasks in one workflow or tasks from different workflows (which do not have any dependency), they can be executed in parallel.

Algorithm 2. Batch Task Placement Phase

Input: WF: The list of all workflows ,$FinalOrderedList$: The 2D Arraylist containing all schedules
Output: $finalConfigs$, $finalCost$
/* N: Number of workflows, WF_n: The nth workflow, Q: Number of all schedules, $MAResultList$: A global 2D list container in which each row stores the offloading configuration of one schedule, $finalConfigs$: A 2D Arraylist container storing obtained severs' configuration of each workflow, $finalCost$: An array to store the execution cost of each workflow */
1 $MAResultList = null$
2 **for** $i = 1$ *to* Q **do**
3 $MAResult.get(i)$ = APMA($FinalOrderedList.get(i)$)
4 $finalConfigs$ = ResultProcessor($MAResultList.get(i)$)
5 **end**
6 **for** $n = 1$ *to* N **do**
7 $finalCost[n]$ = CostCalculator($finalConfigs$)
8 **end**

An overview of the proposed batch application placement phase is presented in Algorithm 2. This phase receives the list of all workflows WF and schedules $FinalOrderedList$ as an input, and outputs the workflows' configuration $finalConfigs$ and the execution cost of all workflows $finalCost$. Considering the number of schedules, the Application Placement Memetic Algorithm (APMA) is invoked to decide for tasks of the current schedule while considering the server assignments of previous schedules (line 3). Since tasks in each schedule are from one or several workflows, the $ResultProcessor(MAResultList)$ method receives tasks assignments of all schedules $MAResultList$, organize tasks assignments of each workflow, and stores them in a 2D Arraylist called $finalConfigs$ so that each row represents one workflow (line 4). When task assignment of all schedules is finished, the $CostCalculator(finalConfigs)$ method calculates the execution cost of each workflow based on the respective obtained configuration. Since the main function of this phase is the APMA, we illustrate how this algorithm works in detail in what follows.

4.2.1 Application Placement Memetic Algorithm (APMA)

The Memetic Algorithm (MA) is algorithmic pairing of evolutionary-based search methods such as GA with one or more refinement methods (i.e, local search, individual learning), used for different types of optimization problems such as routing and scheduling [22]. In the MA, each candidate solution is represented by an individual and the solution is extracted from a set of candidate individuals called population.

We propose an Application Placement Memetic Algorithm (APMA) based on the GA functions, in which local search is applied to the selected individuals of each iteration. This latter helps the APMA converge faster to the best-possible solution. In the APMA, each candidate configuration of servers assigned to tasks of one schedule is encoded as an individual. The atomic part of each individual is a gene which represents a task in a schedule and carries a

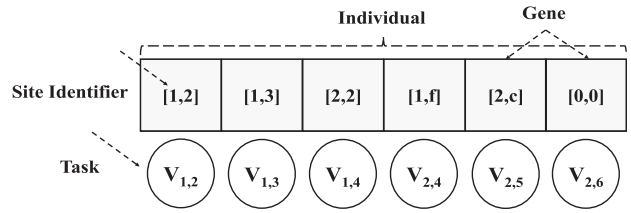


Fig. 3. An individual representing a sample server configuration for second schedule of Fig. 2d.

tuple (x, y) illustrating the type of assigned server x and the index of that server y. The values for each tuple is derived from the Eq. (1) in which values for type and index of servers are defined. Moreover, the length of individuals in each schedule depends on the number of genes (i.e., tasks) on that schedule. A sample individual in our technique is depicted in Fig. 3 representing a sample configuration for tasks in the second schedule of Fig. 2d.

The APMA is made up of five main steps called initialization, selection, crossover, mutation, and local search. The first four steps are among population-based operations used in GA while the local search step is used as the refinement method. Besides, the utility of each candidate individual is evaluated by a fitness function enabling the APMA to select the best individuals in each iteration. An overview of the APMA is presented in Algorithm 3.

Algorithm 3. An Overview of APMA

Input: $scheduleTasks$: A set of tasks for one schedule
Output: $selectedList^{op}.get(0)$
/* I:Maximum iteration number, $selectedList$: The best individuals of respective population found in the in each iteration */
1 $selectedList^{op}$=null; $selectedList^{dp}$=null
2 Initialization($scheduleTasks$)
3 $selectedList^{op}$=Selection(OP)
4 $selectedList^{dp}$=Selection(DP)
5 **for** *i=1 to I* **do**
6 Crossover($selectedList^{op}$, $selectedList^{dp}$)
7 Mutation($selectedList^{op}$, $selectedList^{dp}$)
8 LocalSearch($selectedList^{op}$, $selectedList^{dp}$)
9 $selectedList^{op}$ = selection(OP)
10 $selectedList^{dp}$ = selection(DP)
11 **end**

4.2.2 Initialization Step

In this step, required parameters for the APMA including the maximum number of iterations I, population size $PopSize$, and individuals in the population are initialized. Moreover, alongside with Original Population (OP), a new population is defined to enhance the diversity of solutions, called Diversity Population (DP). Since the main goal of the APMA is to find the best-possible configuration of servers by which the local execution cost decreases, a pre-defined individual is produced for the OP, in which tuple values of all genes are set to their respective local servers (i.e., IoT devices). This reduces the number of low utility individuals because those whose fitness values are worse than the pre-defined individual are not selected in the subsequent iterations. The rest of the individuals in the OP and individuals in the DP are generated randomly in the initialization step.

4.2.3 Fitness Function

The APMA uses two global and local fitness functions for OP, which are used to evaluate the utility of each individual $F_g^{op}(indv)$ (representing the utility of a servers' configuration for tasks of one schedule $indv$), and each task of one workflow on that schedule $F_l^{op}(v_{n,i})$ (representing the cumulative utility of the given task plus the utility of other tasks in that workflow), respectively. The $F_l^{op}(v_{n,i})$ receives a task $v_{n,i}$ and calculates the local fitness value based on Eq. (2) with the assumption that the execution cost of unassigned tasks in one workflow is equal to zero. Moreover, Algorithm 4 demonstrates how the global fitness of each individual $F_g^{op}(indv)$ is calculated. The $F_g^{op}(indv)$ is the sum of local fitness $F_l^{op}(v_{n,i})$ of tasks on that schedule. However, due to the parallel execution of multiple tasks of one workflow in each schedule, the maximum of local fitness $F_l^{op}(v_{n,i})$ values of tasks belonging to the same workflow $MaxLoc$ are first calculated (lines 1-11). The responsibility of finding tasks of the same workflow in one schedule is handled by the $ParallelTaskCheck$ method that stores parallel tasks of one workflow in the $parallelSet$ (line 3). Then, the local fitness of each task in the $parallelSet$ is calculated and the maximum local fitness of tasks belonging to that workflow is stored in $MaxLoc$ (lines 4-10). Finally, the global fitness value $gBest$ can be obtained by summation on all values of $MaxLoc$, which stores the maximum local fitness of each workflow up to that schedule (lines 12-14).

Algorithm 4. Global Fitness Function of OP: F_g^{op}

Input: $indv$: An individual showing tasks of one schedule
Output: $gBest$
/* WF: Set of all workflows , $parallelSet$ = A container to store parallel tasks of one workflow, $MaxLoc$: A container to store the maximum local fitness of each workflow in the schedule, $gBest$: The global best fitness value, $N = |WF|$ */
1 **for** *n=1 to N* **do**
2 $parallelSet$ = null
3 $parallelSet$ = ParallelTaskCheck($indv$, WF_n)
4 $MaxLoc[n] = F_l^{op}(parallelSet.get(1))$
5 **for** *i=1 to* $|parallelSet|$ **do**
6 $tempMax = F_l^{op}(parallelSet_i)$
7 **if** $tempMax > MaxLoc[n]$ **then**
8 $MaxLoc[n] = tempMax$
9 **end**
10 **end**
11 **end**
12 **for** *i=1 to MaxLoc* **do**
13 $gBest = gBest + MaxLoc.get(i)$
14 **end**

The principal goal of the diversity population (DP) is to diversify the individuals in the APMA so that the probability of getting stuck in local optimum decreases. Hence, the fitness function of DP, $F_g^{dp}(indv)$, is different from the OP and is calculated in what follows:

$$F_g^{dp}(indv_r^{dp}) = \sum_{i=1}^{PopSize} H(indv_i^{op}, indv_r^{dp}), \tag{23}$$

where $PopSize$ represents the population size of OP and DP in the APMA. Individual of OP and DP are displayed by

$indv_i^{op}$ and $indv_r^{dp}$, respectively. Besides, $H(indv_i^{op}, indv_r^{dp})$ is the Hamming distance function that calculates the difference between individuals received as its arguments in terms of assigned servers to their tasks, and is defined as:

$$H(indv_i^{op}, indv_r^{dp}) = \sum_{k=1}^{f} df, \tag{24}$$

where f displays the size of that individual (i.e., schedule).

In Eqs. (23) and (24), to calculate the fitness of one individual of DP, we calculate its difference by all individuals in the OP, and the individual with a higher difference receives better fitness value. This helps to maintain individuals with a higher difference in the DP that better diversify the individuals in the APMA. Since different type of servers (i.e., IoT, Fog, and cloud) with different number of servers in each type (i.e., server index) are considered in the system model, a diversity factor df is defined which describes the fitness of each task according to the type and index of its assigned server. This latter is obtained from what follows:

$$df = \begin{cases} 2, & sgn(|ST(indv_{i,k}^{op}) - ST(indv_{r,k}^{dp})|) = 1 \\ 1, & \begin{array}{l} sgn(|ST(indv_{i,k}^{op}) - ST(indv_{r,k}^{dp})|) = 0 \\ \quad \& \\ sgn(|SI(indv_{i,k}^{op}) - SI(indv_{r,k}^{dp})|) = 1 \end{array}, \\ 0, & \begin{array}{l} sgn(|ST(indv_{i,k}^{op}) - ST(indv_{r,k}^{dp})|) = 0 \\ \quad \& \\ sgn(|SI(indv_{i,k}^{op}) - SI(indv_{r,k}^{dp})|) = 0 \end{array} \end{cases} \tag{25}$$

where the kth task (i.e., gene) on those individuals are depicted as $indv_{i,k}^{op}$ and $indv_{r,k}^{dp}$, respectively. sgn is the symbolic function, which is defined as:

$$sgn(|x-y|) = \begin{cases} 0, & x = y \\ 1, & x \neq y \end{cases}, \tag{26}$$

According to Eq. (25), if the server type of each task in the DP (i.e., $ST(indv_{r,k}^{dp})$) is different from the server type of corresponding task in an individual of OP (i.e., $ST(indv_{i,k}^{op})$), it receives higher fitness value. However, in condition that the server types of these tasks are equal, the df is set to 1. Moreover, if the two tasks are assigned to exactly one server (i.e., same server type and server index), the fitness value for that task in the DP is equal to zero.

4.2.4 Selection Step

The goal of selection is to choose the high utility individuals from both OP and DP based on their respective fitness functions for next iterations. To achieve this, the individuals of OP and DP are sorted based on their respective fitness functions and the top three of individuals plus one random individual from each population are selected and stored in the $selectedList^{op}$ and $selectedList^{dp}$, respectively.

4.2.5 Crossover and Mutation Steps

The goal of crossover step is to generate new individuals (called offspring) by a combination of individuals selected in the selection step (called parents). The APMA applies a two-point crossover operation to each pair of selected parents and creates two offspring from them. In each iteration, the total number of new offspring for each population is calculated based on the following equation:

$$off\ spring\ Number = \frac{n!}{(n-k)!}, \quad k = 2. \tag{27}$$

In the two-point crossover, two crossover points are randomly selected from the parents. Then, genes in between the two crossover points are exchanged between the parent individuals while the rest remain unchanged. Since the APMA uses two populations OP and DP, the crossover between individuals of each population is called inbreeding, while the crossover between individuals of different populations is called crossbreeding. The crossbreeding provides diversity in individuals which helps to avoid local optimal values with higher probability. Besides, the outcomes of crossbreeding are stored in selected list of both populations $selectedList^{op}$, $selectedList^{dp}$, while the results of inbreedings are only stored in the selected list of respective populations.

In the APMA, the mutation function, based on the predefined probability, modifies several genes of offspring in hope of generating individuals with higher utility.

4.2.6 Local Search Step

Considering the fact that crossover points and genes for the mutation are selected randomly, a new function called local search is defined which works based on the local fitness function of the OP ($F_l^{op}(v_{n,i})$). It is worth mentioning that the randomness provided by the crossover function and mutation is essential since it provides the opportunity to jump out from local optimal points with a higher probability. The local search function, alongside with those random functions, leads to faster convergence to the global optimal solution. Algorithm 5 demonstrates the process of local search step.

Algorithm 5. Local Search Step

Input: $selectedList^{op}$: Selected list of the OP, $selectedList^{dp}$: Selected list of the DP

```
/* tempList: A temporary list container storing the best-found tuple values for each gene in the individual */
size=|selectedList^op|
tempList=setList(MAXINT)
for i=1 to |indv| do
for j=1 to size do
 % j iterates over |selectedList^op|
if F_l^op(indv_{j,i}^op) < tempList.get(i) then
tempList[i]=F_l^op(indv_{j,i}^op)
end
end
end
selectedList^op.add(CreateNewIndv(tempList.get(i)))
UpdatePop(OP, selectedList^op)
UpdatePop(DP, selectedList^dp)
```

Although the local search function increases the probability to converge faster to the global optimal solutions, two problems may occur. First, if the local search functions are used solely, the probability of getting stuck in the local

optimal points increases. Second, for problems with a large solution space, the local search function requires a significant amount of time to visit the search space. Hence, these two factors should be considered while designing a local search function in the APMA. To address the first issue, the crossover and mutation functions which provide randomness are kept in the APMA. Moreover, the diversity population DP is created which ensures diversity in each iteration. To benefit from the local search function while decreasing its searching time, we reduce the search space for local search by only considering the individuals in the selected list of OP (i.e., $selectedList^{op}$) (line 1). The $setList(MAXINT)$ initializes the $tempList$ with infinite value for all its indexes. Considering individuals in the $selectedList^{op}$, genes with the same index number are evaluated in terms of their local fitness values $F_l^{op}(indv_{j,i}^{op})$ and best genes are selected and stored in the respective index number of $tempList$ (lines 3-9). Since the fitness function is defined according to the execution cost, the less fitness value means better assignment (line 5). Afterward, a new individual is created and stored in the $selectedList^{op}$ (line 10). Finally, the updated $selectedList^{op}$ in the local search step and the $selectedList^{dp}$ are then combined with the OP and DP respectively, and top individuals of each population (up to the $PopSize$) are selected for the populations of the next iteration (lines 11-12).

Whenever the APMA reaches to its stopping criteria, the best individual of the OP stored in $selectedList^{op}.get(0)$ is returned as the result of the APMA.

4.3 Failure Recovery Phase

Failures can happen in any systems, and hence, providing an efficient failure recovery method is of paramount importance. In our system, brokers always keep records of free servers and check whether they are planned to perform a task in the near future or not. Besides, considering the assigned server to each task, they estimate the completion cost of each task based on its local fitness value $F_l^{op}(v_{n,i})$. So, if the execution of any tasks fails, the failure recovery method is called to select a surrogate server for that task. The failure recovery method receives the list of current free servers (including IoT devices) and failed task as inputs. Then, it calculates the local fitness value $F_l^{op}(v_{n,i})$ of that tasks for free servers. Finally, tasks will be forwarded to the server with the least $F_l^{op}(v_{n,i})$ for the execution.

4.4 Complexity Analysis

The Time Complexity (TC) of our technique depends on its three phases. We consider the number of incoming workflows to the broker as N and the maximum number of tasks for all workflows as L. The most time-consuming part in the pre-scheduling phase (Algorithm 1) is the BFS which requires $O(L + |E|)$ time to visit all tasks of one workflow in which $|E|$ represents the number of data flows. In the dense DAG, the $|E| = O(L^2)$. Hence, the TC of pre-scheduling phase at the worst case is of $O(N \times L^2)$. In addition, in the best-case scenario, if we assume $N = 1$, and $|E| = O(L)$ for sparse DAGs, the TC is of $O(L)$.

The batch task placement phase (Algorithm 2) calls the APMA (Algorithm 3) Q times where Q represents the number of schedules. To calculate the TC of the second phase, we ignore the iteration size I and the population size $popSize$ of the APMA since they are constant values. In the APMA, the local fitness function $F_l^{op}(v_{n,i})$ and $ParallelTaskCheck$ which are invoked from the global fitness function (Algorithm 4) are the most repeated functions, defining the TC of the batch application placement phase. The TC of $ParallelTaskCheck$ depends on the size of $indv$ which at most can be $N \times (L-1)$ in the case that each workflow has $L-1$ parallel tasks in one schedule. Hence, the TC of $parallelTaskCheck$ at the worst case is of $O(Q \times N^2 \times L)$. The maximum length of $parallelSet$ (line 5 of Algorithm 4) is $L-1$, and hence, the local fitness function $F_l^{op}(v_{n,i})$ is called $Q \times N \times (L-1)$ times. Moreover, the instructions in the $F_l^{op}(v_{n,i})$ at most can be executed L times since the local fitness function only considers tasks of one workflow which are at most L. Finally, the TC of the batch task placement phase (Algorithm 2) at the worst case is of $O(Q \times (N \times L^2 + N^2 \times L))$. In addition, in the best-case scenario, if we assume $N = 1$, the TC is of $O(Q \times L^2)$.

The TC of the failure recovery phase depends on the TC of local fitness function $F_l^{op}(v_{n,i})$ which is of $O(L)$, and the number of free servers which at most is equal to all available servers in the system M. Hence the TC of this phase at the worst case is of $O(M \times L)$. In addition, in the best-case scenario, no failure happens in the system.

Considering that in all cases $2 \leq Q$, the TC of our technique at the worst case is polynomial and is represented as $O(Q(NL^2 + N^2L) + ML)$. Besides, in the best-case scenario, where $N = 1$, $Q = 2$, and no failures occur in the system, the TC is of $O(L^2)$.

5 Performance Evaluation

In this section, the system setup and parameters, and detailed performance analysis of our technique in comparison to its counterparts (especially [4]) are provided.

5.1 System Setup and Parameters

In our experiments, all techniques are implemented and evaluated using iFogSim simulator [23]. We used two types of workflows, namely, real workflows of applications and synthetic workflows. For the real workflows, we used the DAGs extracted from the face recognition application [16] ($Workflow_1$) and the QR code recognition application [24] ($Workflow_2$). Moreover, to consider other possible forms of workflows, several synthetic workflows are generated ($Workflow_3$ to $Workflow_6$). We consider an environment in which six IoT devices are available and each IoT device has one specific workflow from $Workflow_1$ to $Workflow_6$. Each group of six IoT devices is connected to one fog broker, and fog brokers have access to six fog servers and three cloud servers. In this setup, each fog server has three VMs while each cloud server is assumed to have 16 VMs. The computing power of IoT devices is considered as 500 MIPS [4] and their power consumption in processing and idle states are 0.9W and 0.3W respectively. Besides, the transmission power consumption of IoT devices is 1.3W [25]. We also assume that the computing power of each VM of fog servers is 6 or 8 times more than IoT devices [4], [26] while the computing power of each VM of cloud servers are 10 or 12 times more than IoT devices [4]. The summary of our evaluation parameters and their respective values is presented in Table 2.

TABLE 2
Evaluation Parameters

Evaluation Parameters	Value
Number of IoT devices	6
Number of Fog/Edge servers	6
Number of Cloud servers	3
Bandwidth of LAN	(2000,4000) KB/s
Bandwidth of WAN	(500,1000) KB/s
Delay of LAN	0.5 ms
Delay of WAN	30 ms
Computing power of IoT devices	500 MIPS
Speedup Factor of Fog/Edge Servers' VMs	(6, 8)
Speedup Factor of Cloud Servers' VMs	(10, 12)
Idle Power Consumption of IoT device	0.3 W
CPU power of IoT devices	0.9 W
Transmission Power of IoT devices	1.3 W

5.2 Performance Study

We employed three quantitative parameters including execution time, energy consumption, and weighted cost to comprehensively study the behavior of our technique in different experiments. Five experiments are conducted to analyze the efficiency of techniques in terms of various bandwidths, different iteration sizes, techniques' decision times, failure recovery, and system size analysis. Both ψ_γ and ψ_θ are set to 0.5 meaning that the importance of execution time and energy consumption is equal in the results. However, these parameters can be adjusted based on the users' requirements and network conditions. To analyze the efficiency of our technique, the following methods are implemented for comparisons:

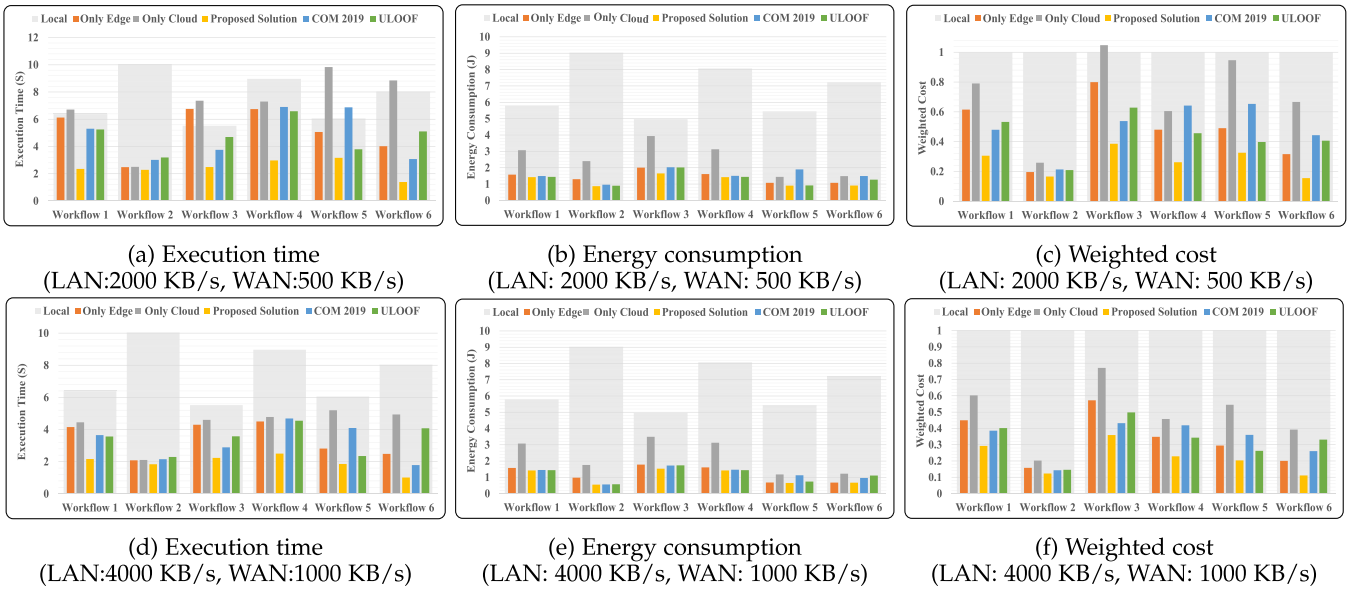
- *Local*: In this method, all tasks of workflows are executed locally on their respective IoT devices, and hence, no parallel execution of tasks can be performed for workflows. The results of this method can be used as a reference point to analyze the gain of application placement techniques.
- *Only Edge*: In this method, all tasks of workflows are offloaded to the fog/edge servers in the edge layer for the execution. If the VMs of all servers are full and there is no free VMs, the remaining tasks have to wait until free computing resources become available.
- *Only Cloud*: In this method, all tasks of workflows are executed on the cloud servers.
- *COM2019*: To the best of our knowledge, there is no work considering batch application placement in a scenario with multiple IoT devices, multiple fog servers, and multiple cloud servers. Therefore, we updated the fitness function and chromosome structure of the [4], which only consider single fog server and single cloud server, to become compatible with our system model. Afterward, the efficiency of its heuristics and searching methods are compared with the other techniques.
- *ULOOF*: This is the extended version of user level online offloading technique [15], so that it can consider scenarios with multiples cloud and fog/edge server for task placement.

The obtained results of each workflow are the average of 10,000 runs with a 95 percent confidence interval.

5.2.1 Bandwidth Analysis

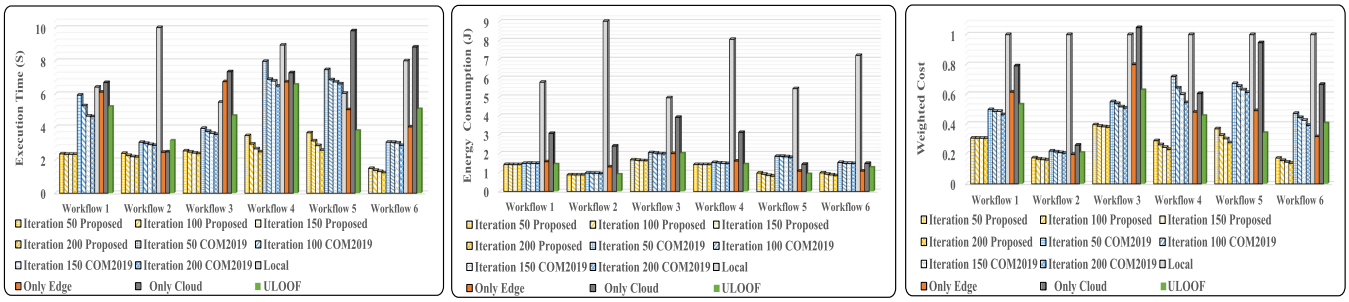
In this experiment, we study the behavior of techniques in various bandwidth values as depicted in Fig. 4. The maximum iteration size I and population size $PopSize$ are set to 100 and 20, respectively.

Fig. 4 shows that as the bandwidth increases, the execution time, energy consumption, and weighted cost of workflows decrease, meaning better application placement gain in comparison to local execution of workflows. Moreover, in most of cases, the only edge method outperforms the only cloud because the fog servers are distributed at the proximity of IoT devices and can be accessed by higher Bandwidth and less latency. However, since the resources of fog servers are limited compared to cloud servers, it cannot obtain the best-possible outcome. This is why the COM2019 and the ULOOF obtain better results in most scenarios than only cloud and only edge methods. They use



(a) Execution time (LAN:2000 KB/s, WAN:500 KB/s)
(b) Energy consumption (LAN: 2000 KB/s, WAN: 500 KB/s)
(c) Weighted cost (LAN: 2000 KB/s, WAN: 500 KB/s)
(d) Execution time (LAN:4000 KB/s, WAN:1000 KB/s)
(e) Energy consumption (LAN: 4000 KB/s, WAN: 1000 KB/s)
(f) Weighted cost (LAN: 4000 KB/s, WAN: 1000 KB/s)

Fig. 4. Execution cost of workflows with different bandwidth values.



(a) Execution time (LAN: 2000 KB/s, WAN: 500 KB/s)

(b) Energy consumption (LAN: 2000 KB/s, WAN: 500 KB/s)

(c) Weighted cost (LAN: 2000 KB/s, WAN: 500 KB/s)

Fig. 5. Execution cost of workflows with different maximum iteration number values.

the resources of cloud and fog servers simultaneously, resulting in the parallelization of more tasks. As it can be seen, our proposed technique is superior to all other methods due to two important reasons. First, similar to the COM2019 and the ULOOF, it utilizes the resources of fog and cloud servers simultaneously. Second, due to its local fitness function, local search, and the diversity provided by the DP, it stays away from local optimal values with higher probability, converges faster to the optimal solution, and hence, outperforms the COM2019 and the ULOOF.

It is worth mentioning that in some cases such as $Workflow_5$ in Fig. 4c, the weighted cost of the only cloud method is less than the local execution, however, its execution time in Fig. 4a is far more than the local execution. This is because the ψ_γ and ψ_θ are set to 0.5, which give equal importance to execution time and energy consumption. Therefore, due to lower value for the energy consumption in this workflow compared to its obtained execution time, the weighted cost shows low gain for the task placement.

5.2.2 Maximum Iteration Number Analysis

One of the important parameters for comparing evolutionary application placement techniques is the maximum iteration number, through which their convergence speed to the optimal solution can be evaluated. In this experiment, the performance of COM2019 and our technique are studied. Since the solution of the local execution, only edge, only cloud, and ULOOF methods do not change in different iterations, the obtained results of these methods are just depicted to better understand the efficiency of other techniques. For this experiment, the $PopSize$, the LAN, and WAN bandwidths are set to 20, 2000 KB/s and 500 KB/s, respectively.

It can be seen from Fig. 5 that the increase in maximum number of iterations I leads to better solutions for both our technique and the COM2019 for all workflows in comparison to the ULOOF, local, only edge, and only cloud methods. However, our technique converges to the better solution in a smaller number of iteration compared to the COM2019. The Fig. 5a shows that the obtained results of our technique in $I = 50$ for all workflows outperform the obtained results of the COM2019 even at $I = 200$. This trend can also be seen in Fig. 5b for weighted cost of execution, while in Fig. 5c the obtained results of the COM2019 and our technique are closer to each other. It is important to note that although better solutions can be found by increasing the maximum number of iterations (if the techniques do not get stuck in the local optimal points), the decision time of algorithms also increases that can be critical for some of workflows, especially for latency-sensitive ones.

5.2.3 Decision Time Analysis

This experiment analyzes the efficiency of each technique based on the decision time required to obtain a well-suited solution. Although application placement algorithms offer server configurations by which the execution time and energy consumption of IoT applications can be reduced, the time that they spend to reach that solution is also important. This is mainly because obtaining good server configurations for IoT applications in a long period of time can negatively affect the execution time requirements of IoT applications. Another important reason elaborating the importance of the decision time analysis, especially for evolutionary algorithms, is that only iteration size analysis cannot solely judge the efficiency of one application placement technique. This is because one technique can reach to better solutions in a small number of iterations compared to its counterparts, however, the time spent on each iteration may be far more than other techniques resulting in longer decision time. Hence, although the maximum iteration size analysis is required, the decision time analysis acts as a supplementary analysis to ensure the efficiency of one technique. In this experiment, the population size $PopSize$ is set to 20, and the LAN and WAN bandwidths are 2000 KB/s and 500 KB/s, respectively.

Table 3 represents obtained execution times of our proposed solution and COM2019 for four different decision

TABLE 3
Decision Time Analysis

Decision Time	Technique	Workflow Execution Time Result					
		WF1	WF2	WF3	WF4	WF5	WF6
100 ms	Proposed	2.412	2.467	2.758	3.638	3.837	1.649
	COM2019	4.333	2.917	3.422	6.276	6.526	3.09
200 ms	Proposed	2.345	2.397	2.610	3.430	3.384	1.446
	COM2019	4.073	2.707	2.984	5.344	5.109	2.529
300 ms	Proposed	2.288	2.302	2.455	2.869	3.362	1.344
	COM2019	3.656	2.494	2.868	4.388	4.709	2.746
400 ms	Proposed	2.229	2.204	2.403	2.587	2.870	1.304
	COM2019	3.623	2.445	2.753	3.663	4.295	2.523

TABLE 4
Failure Recovery Analysis

Technique	Workflow Execution Time Results					
	WF1	WF2	WF3	WF4	WF5	WF6
Proposed (FR Mode)	2.7132	2.6243	2.8642	3.4125	3.6321	1.4685
Local	6.4354	10.031	5.5194	8.9654	6.0520	8.0180

times. Since the execution time result of the ULOOF does not change in different decision times, its respective results are not presented in Table 3, however, its average decision time is roughly 30 ms. As the decision time of techniques increases from 100 ms to 400 ms, the execution time of techniques decreases meaning that the higher utility results are obtained. The obtained results of our solution gradually decrease from 100 ms to 400 ms, while the results of COM2019 has a significant decreasing trend in the range of 100-200 ms and 200-300 ms, and gradually decrease between 300-400 ms, which means that the results of COM2019 approximately converged at 400 ms. It can be clearly seen that our technique not only provides better values compared to the COM2019 in the equivalent decision time, but its results at 100 ms also outperform the results of the COM2019 at 400 ms. This demonstrates that, regardless of number of iterations, our technique converges faster to the optimal solutions.

5.2.4 Failure Recovery Analysis

This experiment analyzes the effect of failure recovery method in application placement techniques. Since the COM2019 and ULOOF do not have any failure recovery method, we present results of our technique with failure recovery mode (FR Mode) when the probability of failure occurrence is 5 percent in comparison to the local execution, as depicted in Table 4. In this experiment, the maximum iteration size I is equal to 100 and values of the rest of parameters are set as same as parameters in decision time analysis.

Table 4 shows that obtained results of our technique with FR mode still outperform results of local execution for all workflows and achieve offloading gain. In techniques ignoring failure recovery in their consideration, failed tasks result in incomplete execution of workflows due to dependencies among tasks of one workflow. However, our technique, by accepting a small overhead of failure recovery phase, can achieve a reasonable gain in comparison to local execution.

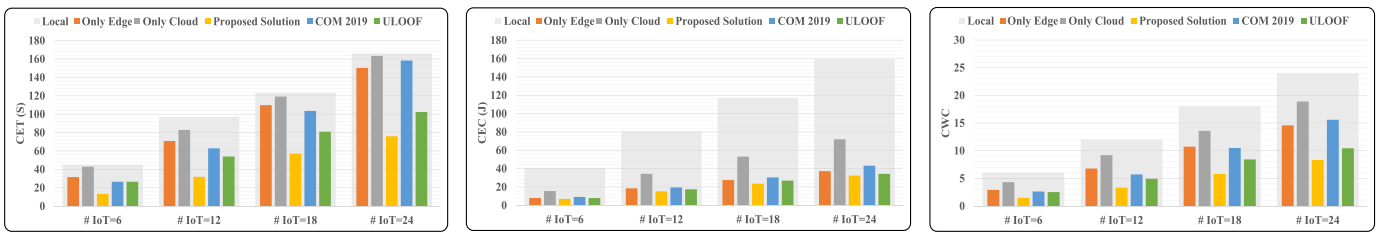
5.2.5 System Size Analysis

In this experiment, we analyze the effect of system size on different application placement techniques. In our system, each fog broker makes application placement decisions for its respective IoT devices. Hence, to analyze the performance of our proposed technique, we increase the number of IoT devices and fog servers per each fog broker from 6 to 24 by the step of 6. Moreover, in this experiment, we use the same workflows as the previous experiments. In addition, the LAN, and WAN bandwidths are set to 2000 KB/s and 500 KB/s, respectively, and the rest of parameters are as the same as values of Table 2.

The Fig. 6 shows the result of Cumulative Execution Time (CET), Cumulative Energy Consumption (CEC), and Cumulative Weighted Cost (CWC) when different numbers of IoT devices are connected to one fog broker. The term cumulative refers to the aggregate execution cost of all IoT devices (e.g., the CET shows the aggregate execution time of all IoT devices in scenarios with different number of IoT devices). In Fig. 6, the CET, CEC, and CWC increase as the number of IoT devices increases. In all scenarios, the CET, CEC, and CWC of all methods are lower than the local execution cost, however, our proposed technique outperforms other methods in all scenarios and results in lower cost. In addition, the performance of the ULOOF and COM2019 is roughly the same in scenarios with six IoT devices, however the ULOOF shows better performance for the rest of scenarios. This latter is because ULOOF is independent of maximum number of iteration while the performance of the COM2019 largely depends on the maximum number of iterations.

6 Conclusion and Future Work

We proposed a weighted cost model for optimizing the execution time and energy consumption of IoT devices in a heterogeneous computing environment, in which multiple IoT devices, multiple fog servers, and multiple cloud servers are available. We also proposed a batch application placement technique based on the Memetic Algorithm to efficiently place tasks of different workflows on appropriate servers in a timely manner. Besides, a light-weight failure recovery technique is proposed to overcome the potential failures in the execution of tasks in runtime. The effectiveness of our technique is analyzed through extensive experiments and comparisons by the state-of-the-art techniques in the literature. The obtained results demonstrate that our technique improves its counterparts by 65 and 51 percent in terms of weighted cost in bandwidth analysis and execution time in decision time analysis, respectively. The performance results demonstrate that our technique achieves up to 65 percent improvement over existing counterparts in terms of the weighted cost.



(a) Cumulative Execution Time (CET) (b) Cumulative Energy Consumption (CEC) (c) Cumulative Weighted Cost (CWC)

Fig. 6. System size analysis with different number of IoT devices per fog broker.

As part of future work, we plan to extend our proposed weighted cost model to consider other aspects such as monetary cost. Moreover, we plan to apply mobility models in this scenario and adapt our proposed application placement technique accordingly.

REFERENCES

[1] Internet of Things at-a-glance, Accessed on: Aug. 4, 2019. 2016. [Online]. Available: https://www.cisco.com/c/dam/en/us/products/collateral/se/internet-of-things/at-a-glance-c45-731471.pdf

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

[3] M. Goudarzi, M. Zamani, and A. T. Haghighat, "A fast hybrid multi-site computation offloading for mobile cloud computing," *J. Netw. Comput. Appl.*, vol. 80, pp. 219–231, 2017.

[4] X. Xu *et al.*, "A computation offloading method over big data for IoT-enabled cloud-edge computing," *Future Gener. Comput. Syst.*, vol. 95, pp. 522–533, 2019.

[5] M. Goudarzi, M. Palaniswami, and R. Buyya, "A fog-driven dynamic resource allocation technique in ultra dense femtocell networks," *J. Netw. Comput. Appl.*, vol. 145, 2019, Art. 102407.

[6] Z. Zhu, T. Liu, Y. Yang, and X. Luo, "Blot: Bandit learning-based offloading of tasks in fog-enabled networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 12, pp. 2636–2649, Dec. 2019.

[7] J. Wang, K. Liu, B. Li, T. Liu, R. Li, and Z. Han, "Delay-sensitive multi-period computation offloading with reliability guarantees in fog networks," *IEEE Trans. Mobile Comput.*, to be published, doi: 10.1109/TMC.2019.2918773.

[8] L. Huang, X. Feng, L. Zhang, L. Qian, and Y. Wu, "Multi-server multi-user multi-task computation offloading for mobile edge computing networks," *Sensors*, vol. 19, no. 6, 2019, Art. no. 1446.

[9] D. G. Roy, D. De, A. Mukherjee, and R. Buyya, "Application-aware cloudlet selection for computation offloading in multi-cloudlet environment," *J. Supercomputing*, vol. 73, no. 4, pp. 1672–1690, 2017.

[10] E. El Haber, T. M. Nguyen, D. Ebrahimi, and C. Assi, "Computational cost and energy efficient task offloading in hierarchical edge-clouds," in *Proc. 29th IEEE Int. Symp. Personal Indoor Mobile Radio Commun.*, 2018, pp. 1–6.

[11] Y. Nan, W. Li, W. Bao, F. C. Delicato, P. F. Pires, and A. Y. Zomaya, "A dynamic tradeoff data processing framework for delay-sensitive applications in cloud of things systems," *J. Parallel Distrib. Comput.*, vol. 112, pp. 53–66, 2018.

[12] R. Mahmud, S. N. Srirama, K. Ramamohanarao, and R. Buyya, "Quality of experience (QoE)-aware placement of applications in fog computing environments," *J. Parallel Distrib. Comput.*, vol. 132, pp. 190–203, 2018.

[13] M.-H. Chen, B. Liang, and M. Dong, "Joint offloading and resource allocation for computation and communication in mobile cloud with computing access point," in *Proc. IEEE Conf. Comput. Commun.*, 2017, pp. 1–9.

[14] Z. Hong, W. Chen, H. Huang, S. Guo, and Z. Zheng, "Multi-hop cooperative computation offloading for industrial IoT-edge-cloud computing environments," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 12, pp. 2759–2774, Dec. 2019.

[15] J. L. D. Neto, S.-Y. Yu, D. F. Macedo, J. M. S. Nogueira, R. Langar, and S. Secci, "ULOOf: A user level online offloading framework for mobile edge computing," *IEEE Trans. Mobile Comput.*, vol. 17, no. 11, pp. 2660–2674, Nov. 2018.

[16] H. Wu, W. Knottenbelt, and K. Wolter, "An efficient application partitioning algorithm in mobile environments," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 7, pp. 1464–1480, Jul. 2019.

[17] L. Lin, P. Li, X. Liao, H. Jin, and Y. Zhang, "Echo: An edge-centric code offloading system with quality of service guarantee," *IEEE Access*, vol. 7, pp. 5905–5917, 2018.

[18] G. L. Stavrinides and H. D. Karatza, "A hybrid approach to scheduling real-time iot workflows in fog and cloud environments," *Multimedia Tools Appl.*, vol. 78, pp. 24639–24655, 2018.

[19] R. Mahmud, K. Ramamohanarao, and R. Buyya, "Latency-aware application module management for fog computing environments," *ACM Trans. Internet Technol.*, vol. 19, no. 1, 2018, Art. no. 9.

[20] S. Bi, L. Huang, and Y.-J. A. Zhang, "Joint optimization of service caching placement and computation offloading in mobile edge computing system," 2019, *arXiv:1906.00711*

[21] M. Goudarzi, M. Zamani, and A. Toroghi Haghighat, "A genetic-based decision algorithm for multisite computation offloading in mobile cloud computing," *Int. J. Commun. Syst.*, vol. 30, no. 10, 2017, Art. no. e3241.

[22] X. Chen, Y.-S. Ong, M.-H. Lim, and K. C. Tan, "A multi-facet survey on memetic computation," *IEEE Trans. Evol. Comput.*, vol. 15, no. 5, pp. 591–607, Oct. 2011.

[23] H. Gupta, A. Vahid Dastjerdi, S. K. Ghosh, and R. Buyya, "iFogSim: A toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments," *Softw., Practice Experience*, vol. 47, no. 9, pp. 1275–1296, 2017.

[24] L. Yang, J. Cao, Y. Yuan, T. Li, A. Han, and A. Chan, "A framework for partitioning and execution of data stream applications in mobile cloud computing," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 40, no. 4, pp. 23–32, 2013.

[25] K. Kumar and Y.-H. Lu, "Cloud computing for mobile users: Can offloading computation save energy?" *Computer*, vol. 43, no. 4, pp. 51–56, 2010.

[26] L. F. Bittencourt, J. Diaz-Montes, R. Buyya, O. F. Rana, and M. Parashar, "Mobility-aware application scheduling in fog computing," *IEEE Cloud Comput.*, vol. 4, no. 2, pp. 26–35, Mar.-Apr. 2017.

Mohammad Goudarzi (Member, IEEE) is currently working toward the PhD degree at the Cloud Computing and Distributed Systems (CLOUDS) Laboratory, Department of Computing and Information Systems, the University of Melbourne, Australia. He was awarded the Melbourne International Research Scholarship (MIRS) supporting his studies. His research interests include Internet of Things (IoT), Fog Computing, Wireless Networks, and Distributed Systems.

Huaming Wu (Member, IEEE) received the BE and MS degrees in electrical engineering from the Harbin Institute of Technology, China, in 2009 and 2011, respectively, and the PhD degree with the highest honor in computer science at Freie Universität Berlin, Germany, in 2015. He is currently an associate professor with the Center for Applied Mathematics, Tianjin University. His research interests include mobile cloud computing, edge computing, fog computing, Internet of Things (IoTs), and deep learning.

Marimuthu Palaniswami (Fellow, IEEE) received the PhD degree from the University of Newcastle, Australia before joining the University of Melbourne, where he is currently a professor of Electrical Engineering. Previously, he a distinguished lecturer of the IEEE Computational Intelligence Society. He was also a co-director of Centre of Expertise on Networked Decision & Sensor Systems. He has published more than 500 refereed journal and conference papers, including three books, ten edited volumes.

Rajkumar Buyya (Fellow, IEEE) is currently a Redmond Barry distinguished professor and director of the Cloud Computing and distributed systems (CLOUDS) Laboratory at the University of Melbourne, Australia. He has authored more than 625 publications and seven text books including "Mastering Cloud Computing" published by McGraw Hill, China Machine Press, and Morgan Kaufmann for Indian, Chinese and international markets respectively. He is one of the highly cited authors in computer science and software engineering worldwide (h-index=130, g-index=280, 90,300+ citations).