Jordanian Journal of Computers and Information Technology (JJCIT), Vol. 06, No. 04, December 2020.

IMPROVING RESPONSE TIME OF TASK OFFLOADING BY RANDOM FOREST, EXTRA-TREES AND ADABOOST CLASSIFIERS IN MOBILE FOG COMPUTING

Elham Darbanian, Dadmehr Rahbari, Roghayeh Ghanizadeh and Mohsen Nickray

(Received: 27-May-2020, Revised: 7-Aug.-2020, Accepted: 26-Aug.-2020)

ABSTRACT

The application of computing resources through mobile devices (MDs) is called Mobile Computing; between cloud datacentres and devices, it is known as (Mobile) Fog Computing (MFC). We ran Cloudsim simulator to offload tasks in suitable Fog Devices (FDs), cloud or mobile. We stored the outputs of the simulator as a dataset with features and a target class. A target class is a device in which tasks are offloaded and features of tasks are authentication, confidentiality, integrity, availability, capacity, speed and cost. Decision Tree (DT), Random Forest (RF), Extra-trees and AdaBoost classifiers were classified based on attribute values and the plot of trees was drawn. According to the plot of these classifiers, we extracted each sequential condition from root to leaves and inserted it into the simulator. What these classifiers do is to improve the conditions that should be inserted in the corresponding section of the simulator. We improved the response time of offloading by Random Forest, Extra-trees and AdaBoost over Decision Tree.

KEYWORDS

Fog computing, Decision tree classifier, Random forest classifier, Extra-trees classifier, AdaBoost classifier, Offloading, Machine learning.

1. INTRODUCTION

The application of computing resources through mobile devices (MDs) is called Mobile Computing. The Mobile Computing environment includes four properties of mobility; diversity of network access types, frequent network disconnection, poor reliability and poor security [1]. Between cloud datacentres and Internet of Things (IoT) devices is (Mobile) Fog Computing nods. MFC acts as an intermediate layer between IoT devices/sensors and cloud datacentres. So, it is closer to the IoT devices to handle real-time services and latency-sensitive services and provide better Quality of Service (QoS). Routers, switches, set top boxes, proxy servers, Base Stations (BS), …etc. are in the Fog Computing environment. They can support application execution [2].

Mobile Edge Computing provides services and computing capabilities at the edge of the mobile network and can optimize existing mobile infrastructure services. MEC servers are deployed at multiple locations at the edge of the mobile network to implement the MEC environment [3]. Mobile Cloud Computing extends the computing capabilities to constrained resource mobile devices and benefits from a combination of different technologies (e.g. service-oriented computing, virtualization and grid computing). Mobile devices, communication technology and cloud servers are three main portions of it. Storage, processing, computing and security mechanism for mobile devices are provided by a cloud server through communication technologies [4]. In Table 1, the differences between the three methods including Mobile Cloud Computing, (Mobile) Edge Computing and (Mobile) Fog Computing were presented [5].

Decision Tree can be defined as a non-parametric supervised learning method. It is trained on labeled data to classify it and an acyclic directed graph is built using top-down recursive partitioning of the dataset [6]. In this paper, the dataset has some features and labels that specify the target class. Decision Tree predicts the value of a target inferred from the data features. Iterative Dichotomiser 3 (ID3),

E. Darbanian, D. Rahbari, R. Ghanizadeh and M. Nickray are with the Department of Computer Engineering and Information Technology, University of Qom, Alghadir Ave., Qom, Iran. Emails: edarbanian@gmail.com, d.rahbari@stu.qom.ac.ir, ghanizadehroghayeh1@gmail.com and m.nickray@qom.ac.ir

C4.5, C5.0 and Classification and Regression Trees (CARTs) are various Decision Tree algorithms. We use the Decision Tree classifier code in scikit-learn website which uses an optimized version of the CART algorithm [7].

Table 1. The difference between the three methods Mobile Cloud Computing, (Mobile) Edge Computing and (Mobile) Fog Computing.

	Rich in computing and/or storage resources	Rich in energy and/or power resources	Computation is mainly at the network edge	Data storage is mainly at the network edge	Interaction with remote infrastructure (cloud)	Context (location, activity, …etc.) awareness	Supporting real-time control and interactive services	Support throughput applications	Augmenting cloud data centre services	Augmenting mobile device performance
Mobile Cloud Computing	yes	yes	no	yes	yes	no	yes	no	no	yes
(Mobile) Edge Computing	yes	no	yes	yes	no	yes	yes	no	yes	yes
(Mobile) Fog Computing	yes	yes	yes	yes	no	yes	yes	no	yes	yes

Random Forest fits 100 Decision Tree classifiers by default on the dataset's different sub-samples and improves the predictive accuracy using averaging [8]. We used the Random Forest classifier code in the scikit-learn website. It is applied to the dataset and trees are constructed while the resultant individuals are combined to predict the class label. The word random is for two reasons: first, random sampling for drawing samples and second, selecting attributes or features for generating Decision Trees randomly. AdaBoost and Bootstrapping techniques are used in Random Forest to construct multiple classifiers [9].

An Extra-trees classifier fits some randomized Decision Trees on various sub-samples of the dataset, improves the predictive accuracy and controls over-fitting implementing a meta-estimator and averaging, respectively. It is similar to the Random Forest [10].

Another classifier is AdaBoost. It is a meta-estimator that begins by fitting a classifier on the original dataset. Then, additional copies of the classifier are fitted on the same dataset. However, the weights of wrong classified instances are adjusted such that subsequent classifiers focus more on difficult cases [11]. Boosting methods train predictors consecutively and try to improve their predecessors. AdaBoost is similar to Random Forest at a high level, because it collects the predictions made by each Decision Tree within the forest. Some differences between them are in AdaBoost; the Decision Trees have a depth of 1 and the final prediction made by the model is impacted by the predictions made by each Decision Tree [12]. All these classifiers are popular tools in machine learning.

Key contributions of our paper are:

1- Each FD has its features and parameters based on its internal structure on which the best FD is chosen for the module placement. The parameters that were used in this paper are authentication, confidentiality, integrity, availability, capacity, speed and cost. In this paper, we had four FDs that were called FD1, FD2, FD3 and FD4. The Cloudsim simulator that we used assigns values between 0 and 1 to features at random. Depending on values, tasks are offloaded in suitable FDs and otherwise in cloud or mobile. We stored the outputs of the simulator as a dataset.

2- Decision Tree, Random Forest, Extra-trees and AdaBoost classifiers classify based on feature values and draw the plot of a tree. According to the plot of these classifiers, we extracted each sequential condition from root to leaves and inserted them into the simulator in the corresponding section. This reduced the number of conditions that should be inserted in the corresponding section of the simulator and response time of offloading.

3- Since Random Forest, Extra-trees and AdaBoost classifiers consider 100 different trees by default from which we choose the best one with the highest accuracy, they had a better response time compared to that of Decision Tree.

4- In practice, some of these parameters are not used. What these classifiers do is to improve the conditions that should be inserted in the corresponding section of the simulator. These methods are suitable for tasks that require a shorter response time. In fact, in the operational environment, the values of features and target class (a device in which tasks are offloaded, such as mobile, FD or cloud) can be stored as data in a dataset. Then, by using these classifiers and dataset, we can insert the conditions effectively into the simulator for the next tasks and achieve less response time.

The rest of the paper is organized as follows. Related works are presented in Section 2. In Section 3, the system model is described. The proposed approach is provided in Section 4. In Section 5, the evaluation results of the simulation are described. At last, Section 6 presents the conclusions.

2. RELATED WORK

Task offloading has been noticed a lot in recent years. Related papers were categorized as follows:

Authors in [13] expressed Android Unikernel, a short run time designed for mobile computing offloading under MFC and MEC scenarios. It also has been argued that advanced unikernel is used as a runtime in MEC or MFC to support mobile quality. To this, the concept of Rich-Unikernel was considered which aims to support various applications in one unikernel while avoiding their time-consuming recompilation. In [14], a computation offloading problem was provided in a fog computing system, which uses fog computing to answer computation requests by validating requests through the fog node or central cloud increasing the performance of applications, such as power consumption and delay. Also, the game theory approach was used to minimize the running cost. Specifically, a Generalized Nash Equilibrium Problem (GNEP) was formulated and addressed with various constraints by using the exponential penalty function method and semi-smooth Newton method.

Another way to minimize energy consumption, delays and costs was provided in [15]. Researchers investigated the problem of power consumption, performance delays and costs in a mobile fog computing system. Here, queue theory was used to derive analytical results on power consumption, delay in performance and cost by assuming three different queuing models in MD, fog node and central cloud. Based on this analysis, the multi-part optimization problem has been formulated with a common goal of minimizing energy consumption, delay in execution and cost by optimizing the probability of optimal offloading and power transfer for each mobile device. Also, using an Interior point method-based algorithm, a multi-segment problem with various limitations has been developed.

Authors in [16] proposed container transfer algorithms and architectures to support moving tasks with different needs. Also, the container migration problem of mobile application tasks in large-scale FC was modeled. Then, container transfer algorithms to support moving tasks are proposed. This has significantly reduced latency, power consumption and transmission costs. In [17], the researchers investigated a problem of cost-based fairness in a min-max computation system by optimizing offloading and resource allocation decisions, which minimized the delay cost weighting and energy consumption in the system. To address the Np-hard problem, the computation offloading and resource allocation algorithm (CORA) was proposed, which has low complexity and the offloading decisions are taken at random.

In [18], the performance of SIMDOM (A framework for SIMD instruction translation and offloading in heterogeneous mobile architectures) framework was discussed from various dimensions, such as FMEC and MCC offloading, application partition and increasing input sizes. The SIMDOM framework was evaluated in terms of parameters, such as energy, time and performance of MFLOPS. Comparison with state-of-the-art Qemu-based compiled code-offloading framework was performed, where it was found that the SIMDOM framework provides better results. Paper [19] evaluates information about IoT programs in unstable channel conditions and suggests a new way to model the quality of unstable channels. An optimal programming model and a way to reduce the complexity of the algorithm were proposed, which improved the quality of the algorithm. Besides, an offloading data scheduling algorithm (DEED) was proposed aiming at reducing energy consumption.

In [20], a near-end network solution of computation offloading on the edge/fog of the mobile was presented. Mobility, heterogeneity and geographical distribution of mobile devices are challenges of computing offloading at the edge/fog. For consideration of the computational resource demand, an independent q-learning management framework was presented. The proposed method significantly improved computational offload discharge performance by minimizing computational delay. In [21], the various types of offloading techniques that have recently been introduced in fog-driven literature or edge computing in the cloud-IoT environment are discussed. Some criteria determine when offloading was performed. Finally, the research challenges related to offloading are highlighted in the fog calculations.

In [22], deep reinforcement learning was proposed to solve the problem of offloading large-scale multi-service nodes of MEC and multiple dependencies in mobile tasks. Then, the offloading strategy by each algorithm was simulated on the edge computing iFogSim simulator platform. At last, the advantages and disadvantages of each algorithm are evaluated by comparing different factors, including power consumption, cost, load balancing, delay and network usage. Paper [23] has modeled the expected time and energy cost for different options for offloading a task on the edge, cloud or the device itself. Authors in [24] raised the issue of offloading optimization and then used the metaheuristic method to find the best policy. Also, Simulated Annealing-based Offloading Algorithm (SAOA) has also been proposed to provide a node access estimation policy based on a variety of health care information. Sensitive requirements should be met related to the different roles in IoT for architectural and algorithm design. A blockchain-based Edge ABC architecture and a Task Offloading and Resource Allocation (TO-RA) algorithm have been proposed to meet the requirements [25].

In [26], an efficient resource allocation and computation offloading model for a multiuser MEC system was proposed. Also, Advanced Encryption Standard (AES) method has been introduced to protect sensitive information from cyber-attacks and an optimization problem has been developed for mobile users to minimize energy consumption and delay latency. In [27], effective and efficient services in the fog computing environment called for a decentralized management plan of mobile edge server with p2p activation. In [28], a task-offloading and resource-scheduling algorithm was proposed to solve the problems of minimizing energy consumption and processing time of task offloading in the MEC system.

Internet of connected vehicles (IoV) has been introduced as a technology to provide tracking information to drivers and transportation control systems [29]. In this paper, to reduce execution time and energy consumption while satisfying the privacy of computational tasks, an edge computing method called edge computing-enabled computation offloading (ECO) was presented [29]. In [30], a new offloading strategy based on the firefly technique was presented. The firefly method was designed to address the offloading strategy in the Fog-Cloud environment, which selects a suitable computational device for each application.

In [31], a smart energy management method was proposed to increase the lifetime of the network in a fog computing network. A clustering mechanism was introduced for a computation degradation scenario. In [32], the problem of task mapping and scheduling (TMS) in wireless sensor networks was investigated. Its main goals were to improve runtime, power consumption and network lifetime. The MODIFIED RANDOM BIT CLIMBING (λ -MRBC) method was used to obtain the optimal solution faster. In [33], the negotiation between publisher and fog node was formed as an optimization issue. In [34], to provide effective services for performing tasks sensitive to latency and computation, a positive decision-making algorithm and resource allocation based on deep learning have been developed to minimize time and energy consumption in fog. In [35], fog calculations were introduced in a three-tier architecture to minimize energy consumption. To minimize energy consumption, an energy consumption oriented offloading algorithm for fog computing has been suggested.

Paper [36] presented and analyzed a vector instruction offloading framework (SIMDOM) in heterogeneous compute architectures. In [37], a mathematical model is presented to facilitate the calculation of computational time and energy.

We compare the mentioned offloading and scheduling methods by objectives, network architecture, environment, advantages and disadvantages in Table 2.

Table 2. Comparison of offloading and scheduling methods mentioned.

Algorithm	**Objectives**	**Network and Environment (N and E)**	**Advantages**	**Disadvantages**
Unikernel [13]	Time, memory and energy	N: MFC E: Android- x86	Compared to when running Android VM or Android container, it has the advantages of being small, fast and secure.	- Multi-process applications are not allowed. - Those codes need to fork new processes and cannot run in Android Unikernel.
MOIPM [15]	Energy, delay and cost	N: MFC E: Simulation	Low algorithm complexity.	Not suitable for delay-sensitive applications.
GNEP [14]	Execution cost	N: MFC E: Simulation	The proposed algorithm improves performance and accuracy.	Not suitable for delay-sensitive applications.
DQLCM [16]	Computational delay and power consumption	N: MFC E: Real	Among the many deep learning reinforcement algorithms, it has the advantage of fast decision-making.	May lead to a long delay in processing the work.
CORA [17]	Energy, delay and cost	N: MFC/ MCC E: MATLAB	Unlike previous work, they emphasize influential decision-making [39], [40] or resource allocation [41], with consideration of cloud and fog in common.	Has not considered the queue length and delay of user equipment request.
SIMD [18]	Energy, MFLOPS and execution time	N: MFC/ MEC/ MCC E: Real	Ability to reload from a server and execute SIMD instructions while saving energy and reducing runtime.	The simdom framework does not provide energy efficiency for metrics that have less computation.
OFFLOADING [21]	Reducing energy consumption	N: MFC/ MEC E: Real	Offers complete classification of offloading schemes.	Failure to explain how to do the review.
DEED [19]	Reducing energy consumption	N: MFC/ MEC/ MCC E: Simulation	Providing innovative work for optimal energy consumption.	Too much interpretation
IDRQN [22]	latency and network load	N: MEC E: Simulation	Better performance in power consumption, load balance, latency and average runtime.	Locking in scalability and limited to relatively few problems.
Deep Q-learning [20]	Minimizing delay	N: MFC/ MEC E: MATLAB	Ability of parallel execution.	Lack of expression of the future work.
Modeling time and energy cost [23]	Time and energy cost	N: MEC/ MCC E: Simulation	Dynamic offloading decision-making	Edge devices should be adjacent to IoT devices and their resources are limited.
SAOA [24]	Minimizing delay	N: MFC E: MATLAB	Quick response to a user request.	Privacy may not be protected.
TO-RA [25]	Reduction of delay	N: MEC E: Simulation	Compared to other algorithms, more stable and higher user	- Limitation on computing resources and storage of smart devices.

			scarification.	- Impossibility to perform computational tasks with high complexity for a long time.
Multi-users Computation Offloading Decision [26]	Minimizing time and energy consumption	N: MEC E: MATLAB	Low delay	Increasing the number of Mus causes severe interference.
Decentralized mobile edge server management plan [27]	Minimizing runtime and reducing energy consumption	N: MEC E: simulation	The proposed method is effective and practical.	The search space of this method is very large and time-consuming.
Distributed [28]	Minimizing energy consumption and processing time	N: MEC E: simulation	First work on dynamic task offloading and resource scheduling.	Lack of expression of future work
ECO [29]	Optimizing time and reducing energy consumption	N: MEC E: simulation	Preservation privacy	The smaller the scale of the vehicle, the longer the transmission time.
Firefly [30]	Minimizing computation time and energy consumption	N: MFC/ MCC E: simulation	- Automatic split - Easily finding the best solution	It is difficult to efficiently allocate IoT applications between the fog node and the cloud datacenter.
Prediction-based Energy Harvesting Scheme and Clustering [31]	Increasing network lifetime	N: MFC E: MATLAB	Saving energy and reducing latency	Management complexity
λ -MRBC [32]	Improvement of execution time, energy consumption and network lifetime	N: Wireless E: simulation	The network lifetime is prolonged through using the proposed algorithm.	Lack of expression of future work
Design of an incentive mechanism [33]	Reducing energy consumption and delay	N: MFC E: simulation	Increasing transfer speed	There may be an asymmetry between publisher and fog node.
DLJODRA [34]	Reducing energy consumption and delay	N: MFC E: Tensorflow and MATLAB	Increasing network efficiency	Deep learning-based computation offloading scheme does not consider the optimization allocation of network resources.
Energy consumption-oriented offloading algorithm [35]	Minimizing energy consumption	N: MFC/ MCC E: MATLAB	Better performance	It would have been better if it had introduced a multi-user model.
A framework for translating pre-compiled vector instructions [36]	Saving energy and time	N: MEC/ MCC E: Real	Leading to increased translation training efficiency.	Lack of expression of future work
Mathematical model for calculating the time and energy	Reducing computational time and energy	N: MCC E: simulation	Boosting performance and energy efficiency	It would have been better if authors also talked about the energy consumption coefficients of the

consumption of application models [37]				prominent parameters for smartphones.

3. SYSTEM MODEL

The system architecture is shown in Figure 1 [41]. There are mobile devices at the lowest level. For transferring data to routers, access points or base stations are used and routers send data to the closest FDs for task processing. When the sum of memory power and CPU power consumption is less than Wi-Fi's power consumption, tasks run in mobile; otherwise, they are offloaded in FDs according to their properties. If tasks do not execute in FDs, send them finally to the highest level in the cloud.

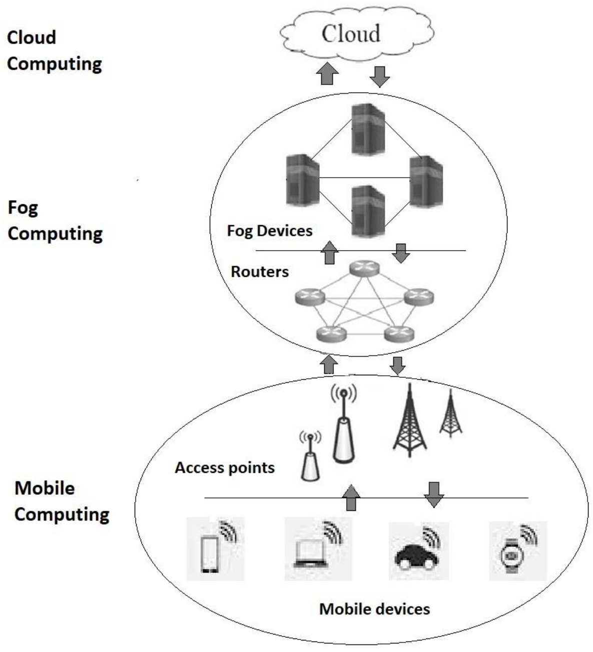

Figure 1. System architecture.

When computing is performed by mobile or portable devices (e.g. laptops, tablets or mobile phones), it is called mobile computing or nomadic computing. It is not suitable for many recent computational challenges, because of the requirements of connected consumer devices. Therefore, fog computing and cloud computing are used for more advanced calculations, because they have more resource-rich hardware [42].

3.1 Calculating Power

We used three resources for power consumption as CPU, RAM and Wi-Fi in our model, which are explained as follows [41].

3.1.1 Power Consumption of CPU

Frequency and performance are the factors on which power consumption of CPU depends. The power consumption of CPU is:

$$P_{CPU} = f_{base} + \sum_{i=0}^{n-1}(f_i * U_i) \tag{1}$$

where f_{base} and f_i are frequency-dependent coefficients, U_i is the utilization of the ith CPU and n is the number of CPUs.

3.1.2 Power Consumption of RAM

The power consumption of RAM depends on the type of modules and is calculated as:

$$P_{RAM} = P_{s1} * U + P_{s2} \tag{2}$$

where P_{s1} and P_{s2} are coefficients of power. U is the aggregated CPU utilization that is:

$$U = \sum_{j=0}^{n-1} Uj \tag{3}$$

U_j is the utilization of jth CPU and n is the total number of CPUs.

3.1.3 Power Consumption of Wi-Fi

Another source of energy consumption in MDs is the power consumption of Wi-Fi. Idle, initial, send, receive and tail are states of the Wi-Fi model, with the total Wi-Fi power consumption for the MDs being the sum of their power consumption as shown in Equation 4. Coefficients are based on quad-core Galaxy S3. T_{State} is the state's time, where T_{Send} and T_{Idle} are calculated in runtime. Finally, N is the number of packets sent or received per second which considered more than 20.

$$P_{Wi\text{-}Fi} = P_{Init} + P_{Send} + P_{Receive} + P_{Tail} + P_{Idle} \tag{4}$$

where

$$P_{Init} = (0.8613 * N + 98.612) * T_{Init}$$

$$P_{Send} = (0.4049 * N + 686.93) * T_{Send}$$

$$P_{Receive} = 0.0211 * N + 15.628$$

$$P_{Tail} = 195 * T_{Tail}$$

$$P_{Idle} = 20 * T_{Idle}$$

4. The Proposed Approach

Each FD has its features and parameters based on its internal structure on which the best FD is chosen for the module placement. The parameters that were used in this paper are: authentication, confidentiality, integrity, availability, capacity, speed and cost. In this paper, we had four FDs that were called FD1, FD2, FD3 and FD4. For example, if the authentication, confidentiality, integrity, availability, capacity, speed and cost of the task that has arrived for processing are > 0.7, > 0.5, < 0.3, < 0.8, < 0.7, < 0.8 and < 0.9, respectively, then the task is performed on FD1. Checking these conditions for allocating the appropriate device can be carried out by Decision Tree or other trees. The features and values of each of the FDs are shown in Table 3.

Table 3. The features of FDs.

	Authentication	**Confidentiality**	**Integrity**	**Availability**	**Capacity**	**Speed**	**Cost**
FD1	> 0.7	> 0.5	< 0.3	< 0.8	< 0.7	< 0.8	< 0.9
FD2	<= 1	< 0.4	> 0.1	< 0.7	< 0.6	< 0.7	< 0.8
FD3	> 0.8	> 0.5	< 0.6	> 0.7	> 0.8	< 0.8	> 0.7
FD4	< 0.9	< 0.7	< 0.8	> 0.9	> 0.7	< 0.8	> 0.6
Cloud	Other values						

Table 4. Part of the dataset.

No.	**Authentication**	**Confidentiality**	**Integrity**	**Availability**	**Capacity**	**Speed**	**Cost**	**Target Classes**
1	0.1	0.16	0.32	0.94	1	0.51	0.69	FD4
2	0.91	0.61	0.04	0.18	0.26	0.17	0.53	FD1
3	0.86	0.18	0.64	0.55	0.05	0.66	0.28	Cloud
4	0.61	0.33	0.28	0.24	0.4	0.13	0.13	FD2
5	0.79	0.45	0.26	0.39	0.17	0.68	0.51	Mobile
6	0.86	0.81	0.12	0.88	0.94	0.59	0.94	FD3
…	…	…	…	…	…	…	…	…

Our base article is [41]. The simulator assigns values between 0 and 1 to features at random. Tasks are offloaded in suitable FDs and otherwise in cloud or mobile, so that when the sum of memory power consumption and CPU power consumption is less than Wi-Fi's power consumption, tasks run in mobile; otherwise, according to Table 3, they are offloaded in an FD or cloud.

We stored the outputs of the simulator as a dataset with 457 samples with 7 features and 1 target class, part of which is shown in Table 4.

In general, with more data, classifiers are better trained to classify. The target class is the device in which the task is offloaded. Hence, we had sex targets, FD1, FD2, FD3, FD4, cloud and mobile. Decision Tree, Random Forest, Extra-trees and AdaBoost classifiers classify based on feature values and draw the plot of the tree. In the plot of the tree, each node is divided into two branches and leaves are the target classes.

For example, in this code that runs by Decision Tree, availability is in the root. If its value in the sample is greater than 0.875, it goes to the left branch; otherwise, it goes to the right branch and so on. In Random Forest, Extra-trees and AdaBoost classifiers, 100 different trees are considered by default and we choose the best one with the highest accuracy. According to the plot of them, we extracted each sequential conditions from root to leaves and insert them into the simulator in the corresponding section. This reduced the number of conditions and response time.

In practice, some of these parameters are not used. For example, the authentication parameter in FD2 is not required for classification. What these classifiers do is to improve the conditions in practice. These methods can be used for tasks that require a shorter response time. In fact, in the operational environment, the values of 7 features and 1 target class can be stored as data in a dataset. Then, by using these classifiers, we can insert the conditions effectively into the simulator for the next tasks and achieve less response time.

In Algorithm I, the steps are performed. In lines 1 to 3 FDs are created with time complexity O(k), so that k is the number of FDs and VMs created with time complexity O(t), where t is the number of VMs in line 4. Then, n tasks are taken from MDs with O(n) in line 5. The broker is created with O(1) and VMs and tasks are submitted to it in lines 6 and 7 with O(n+t). So time complexity until this stage is equal to O(k+n+t).

Algorithm I

```
Input: VMs, Tasks, FDs, cloud
Output: VMs and tasks in the broker
1: for each i ϵ FDs do
2:        Create micro DCs in $FD_i$
3: end for
4: Create VMs
5: Get tasks from MDs
6: Create broker
7: Submit VMs and tasks to the broker
```

In Algorithm II, input includes dataset according to Table 4. By executing Decision Tree, Random Forest, Extra-trees and AdaBoost classifiers, its plot is obtained.

Algorithm II

```
Input: dataset
Output: Plot of Trees
1: Running Decision Tree  classifier by Python code to draw its plot
2: Running Random Forest, Extra-trees and AdaBoost classifiers and choosing the best one with the
heights accuracy by Python code to draw its plot
```

In Algorithm III, tasks are offloaded in one of six modes that include mobile, FD1 to FD4 and cloud. Placement is prepared by calling Algorithm I in line 1. Then, in line 2, new conditions are inserted in the corresponding section of the simulator by running Algorithm II. In lines 3 to 10, tasks are offloaded in one of the six modes with O(n), where n is the number of tasks. In Equations 1 to 4 it is shown, how to calculate memory power, CPU power and Wi-Fi’s power consumption. When the sum of memory power consumption and CPU power consumption is less than Wi-Fi’s power consumption, tasks run in mobile; otherwise, they are offloaded in an FDs or cloud.

Algorithm III

```
Input: VMs, Tasks, FDs, cloud
Output: The places of Tasks
1: Preparing placement by calling Algorithm I
2: Insert new conditions by running Algorithm II to the simulator in the corresponding section
3: for each n ϵ Tasks do
4:         Calculate $PC_{Wi\text{-}Fi}$, $PC_{CPU}$ and $PC_{RAM}$
5: if  $PC_{CPU} + PC_{RAM} < PC_{Wi\text{-}Fi}$ then
6:         Execute task in MD
7: else
8:         Place task in suitable $FD_j$ (j from 1 to 4) or cloud
9: end if
10: end for
```

5. Evaluation

In this part, we compare three methods: mobile, Decision Tree, Random Forest, Extra-trees and AdaBoost classifiers. We used Cloudsim simulator. In the local mobile processing method, the tasks are executed in MD and don't offload to FDs and cloud. These classifiers were executed and the plot of trees is drawn by Python code in the sci-kit learn website [7]. Then, new if statements are added to the relevant section in the simulator. Response time, power consumption of CPU, power consumption of RAM and performance were compared in these three methods. In the simulator, each of them has been run 30 times individually and the average values were presented.

5.1 Configurations of the Simulator

DC and micro DC configurations are shown in Table 5.

We executed the simulation in different states of number of VMs and tasks, as shown in Table 6. In our simulation, the main classes are Cloudlet, Datacenter, DatacenterBroker and VM. Task offloading is carried out in the DatacenterBroker class. New conditions resulting from the implementation of the Decision Tree, Random Forest, Extra-trees and AdaBoost classifiers were inserted into Cloudlet class.

Table 5. DC and Micro DC configurations.

Name	DC	Micro DC
CPU	Octa-core	Quad-core
Memory size	8192	2048 GB
Memory cost	0.015	0.005
Storage size	1 TB	100 GB
Storage cost	0.05	0.01
Bandwidth rate	100 MB/S	10 MB/S
Bandwidth cost	0.1	0.01

Table 6. Different states of the number of VMs and tasks in Cloudsim simulator.

No.	VMs	Tasks	No.	VMs	Tasks
1	10	10	6	50	100
2	10	20	7	100	100
3	20	20	8	100	200
4	40	50	9	200	200
5	50	50	10	500	500

5.2 Offloading Frequency

Figure 2 shows the offloading frequency that is the frequency of offloading tasks to FDs or cloud. When the sum of memory power consumption and CPU power consumption is more than Wi-Fi's power consumption, tasks are offloaded in FDs and cloud. According to Table 1 and the simulator assign values of seven features between 0 and 1 at random, most tasks are offloaded in cloud, FD2, FD1, FD4 and FD3, respectively.

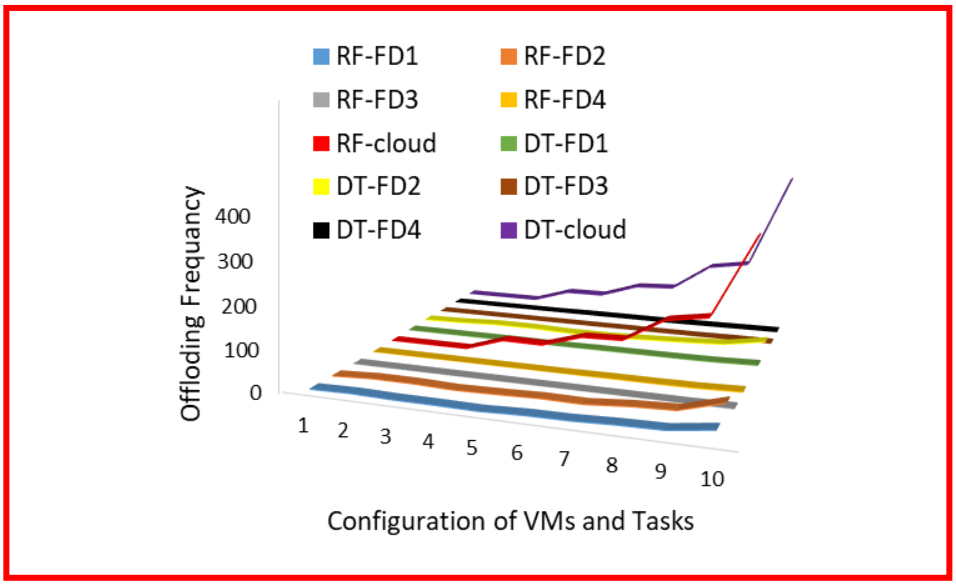

Figure 2. Comparison of offloading frequency to cloud and FDs by decision tree, random forest, extra-trees and AdaBoost and mobile methods.

5.3 Response Time

In Figure 3, the response time is shown. Response time of mobile is less than in Decision Tree, Random Forest, Extra-trees and AdaBoost classifiers, because there is no offloading. As can be seen in the Figure, the response time of Random Forest, Extra-trees and AdaBoost methods is better than that of Decision Tree, because they consider 100 different trees by default, where we choose the best one with the highest accuracy. The average response time in Random Forest, Extra-trees and AdaBoost methods is 649.361, 646.61 and 643.452ms, respectively, while in the Decision Tree method, it is 696.363ms.

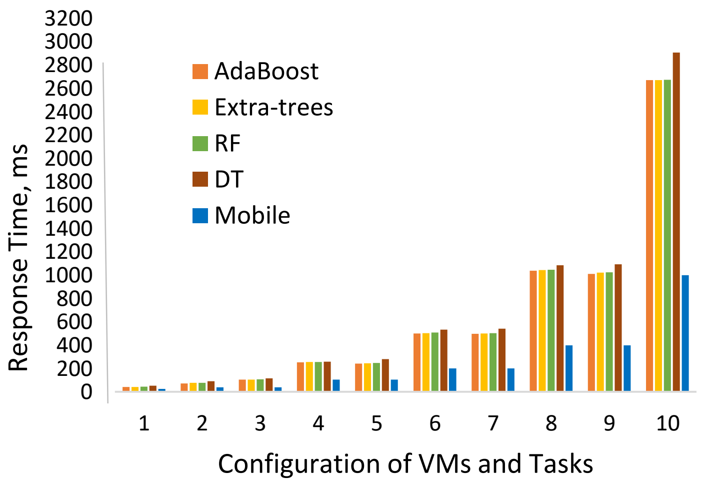

Figure 3. Comparison of response time of decision tree, random forest, extra-trees and AdaBoost and mobile methods.

5.4 The Power Consumption of CPU and RAM

Total power consumption of CPU and RAM is presented in Figures 4 and 5. On average, they are almost the same in Decision Tree, Random Forest, Extra-trees and AdaBoost methods. In the mobile method, it is higher than in the other methods. The average of power consumption of CPU in Decision Tree, Random Forest, Extra-trees and AdaBoost methods is 1874.174, 1875.074, 1877.667 and 1878.147W, respectively, while in the mobile method, it is 1898.796W. The average of the power consumption of RAM in Decision Tree, Random Forest, Extra-trees and AdaBoost methods is 112.45, 112.503, 113.034 and 113.038W, respectively, while in the mobile method, it is 113.842W.

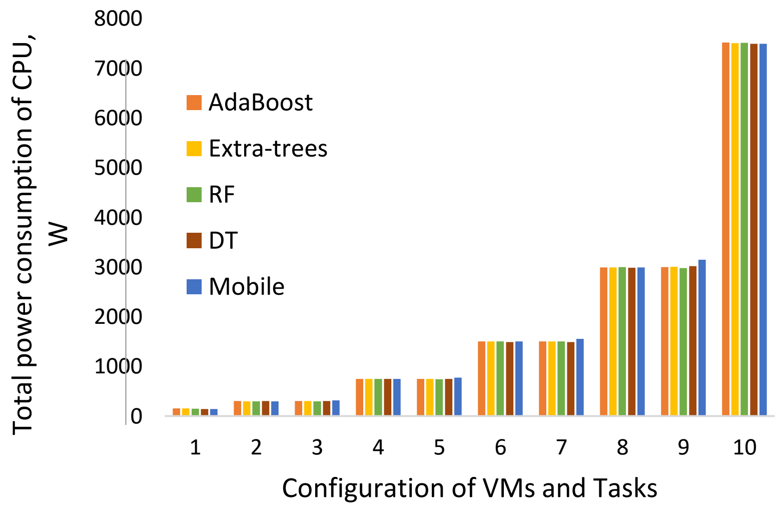

Figure 4. Total power consumption of CPU in decision tree, random forest, extra-trees and AdaBoost and mobile methods.

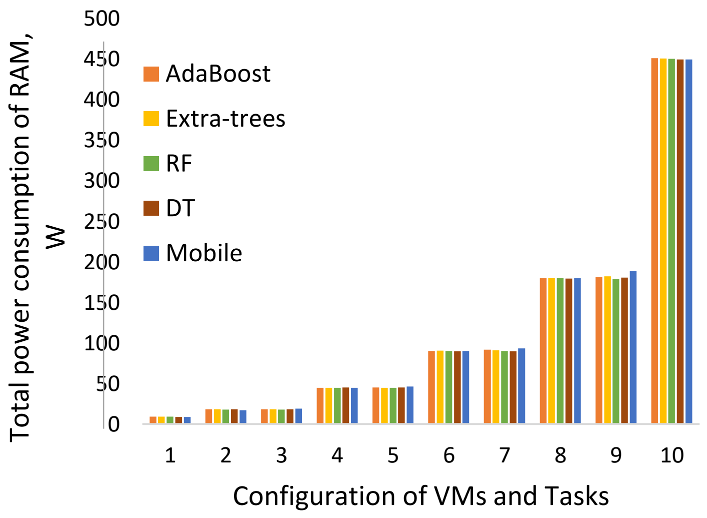

Figure 5. Total power consumption of RAM in decision tree, random forest, extra-trees and AdaBoost and mobile methods.

5.5 Performance

In Figure 6, performance is presented. Its calculation is as follows:

$$\text{performance} = 1 - (P_{\text{Receive}} + P_{\text{Idle}}) / P_{\text{CPU}} \tag{5}$$

where P_{Idle} and P_{Receive} are the power consumption of Wi-Fi in the idle and receive states and P_{CPU} is the power consumption of CPU in MD (see Eq. 4) [42]. As a result, in Figure 5, the performance of Decision Tree, Random Forest, Extra-trees and AdaBoost is better than in the mobile method.

5.6 Comparison of Algorithms

Our simulation results show that the response time of the mobile method is less than in Decision Tree,

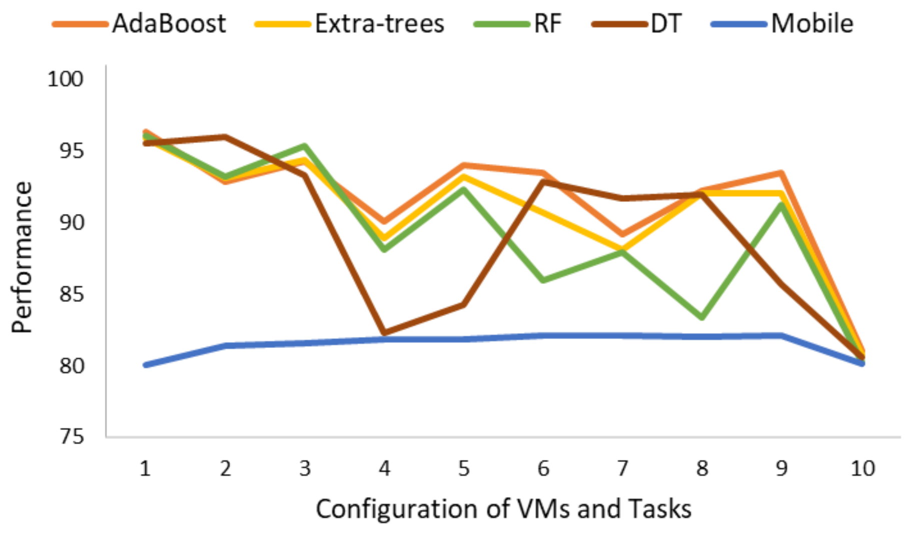

Figure 6. Performance comparison of decision tree, random forest, extra-trees and AdaBoost and mobile methods.

Random Forest, Extra-trees and AdaBoost classifiers, because there is no offloading and the response time of Random Forest, Extra-trees and AdaBoost methods is better than that of the Decision Tree method. AdaBoost is a meta-estimator that begins by fitting a classifier on the original dataset. Then, additional copies of the classifier are fitted on the same dataset. However, the weights of wrong classified instances are adjusted such that subsequent classifiers focus more on difficult cases [11]. Boosting methods train predictors consecutively and try to improve its predecessor, because they collect the predictions made by each Decision Tree within the forest. Some differences between them are in AdaBoost; the Decision Trees have a depth of 1 and the final prediction made by the model is impacted by the predictions made by each Decision Tree [12]. So, the tree of AdaBoost performs better in terms of the conditions that should be inserted in the corresponding section of the simulator compared to Decision Tree. Response time improved by 7.6 percent in this case. An Extra-trees classifier fits some randomized Decision Trees on various sub-samples of the dataset, improves the predictive accuracy and controls over-fitting implementing a meta-estimator and averaging, respectively [10]. Thus, the tree of Extra-trees performs better in terms of the conditions that should be inserted in the corresponding section of the simulator compared to Decision Tree. In this case, response time improved by 7.14 percent. Random Forest fits 100 Decision Tree classifiers by default on the dataset's different sub-samples and improves the predictive accuracy using averaging [8]. So, the tree of Random Forest performs better in terms of the conditions that should be inserted in the corresponding section of the simulator compared to Decision Tree. Response time improved by 6.75 percent in this case.

Total power consumption of CPU and RAM is almost the same in these methods and in the mobile method, it is higher than in the other methods. Also, the performance of them is better than that of the mobile method, because there is no offloading on the mobile. Thus, our offloading methods of using Random Forest, Extra-trees and AdaBoost classifiers have a better response time than Decision Tree on MFC.

6. CONCLUSIONS

The application of computing resources through mobile devices (MDs) is called Mobile Computing. Between cloud datacentres and devices is (Mobile) Fog Computing (MFC). Tasks are offloaded in suitable FDs and otherwise in cloud or mobile. We used Decision Tree, Random Forest, Extra-trees and AdaBoost classifiers for task offloading on MFC, where Random Forest, Extra-trees and AdaBoost classifiers had a better response time than previous methods. Our simulation results showed that the response time of the mobile method is less than these classifiers, because there is no offloading and the response time of Random Forest, Extra-trees and AdaBoost methods was better than in the Decision Tree method. Thus, our offloading methods of using Random Forest, Extra-trees and AdaBoost classifiers had a better response time than that of Decision Tree on MFC. Total power consumption of CPU and RAM was almost the same in these methods and in the mobile method, it was higher than in the other methods. Also, the performance of Decision Tree, Random Forest, Extra-trees and AdaBoost was better than in the mobile method.

For future work, we will try to implement an algorithm that gives better results in the simulator. Also, machine learning methods can be a great way to task offloading.

REFERENCES

[1] T. H. Noor, S. Zeadally, A. Alfazi and Q. Z. Sheng, "Mobile Cloud Computing: Challenges and Future Research Directions," Journal of Network and Computer Applications, vol. 115, pp. 70-85, 2018.

[2] R. Mahmud, R. Kotagiri and R. Buyya, "Fog Computing: A Taxonomy, Survey and Future Directions," Internet of Everything, pp. 103-130, Springer, Singapore, 2018.

[3] R. Roman, J. Lopez and M. Mambo, "Mobile Edge Computing, Fog et al.: A Survey and Analysis of Security Threats and Challenges," Future Generation Computer Systems, vol. 78, pp. 680-698, 2018.

[4] F. Gu, J. Niu, Z. Qi and M. Atiquzzaman, "Partitioning and Offloading in Smart Mobile Devices for Mobile Cloud Computing: State-of-the-art and Future Directions," Journal of Network and Computer Applications, vol. 119, pp. 83-96, 2018.

[5] C. Li, Y. Xue, J. Wang, W. Zhang and T. Li, "Edge-oriented Computing Paradigms: A Survey on Architecture Design and System Management," ACM Computing Surveys (CSUR), vol. 51, no. 2, pp. 1-34, 2018.

[6] S. Fletcher and Md. Z. Islam, "Decision Tree Classification with Differential Privacy: A Survey," ACM Computing Surveys (CSUR), vol. 52, no. 4, pp. 1-33, 2019.

[7] Scikit-learn, "Decision Trees (DTs)," [Online], Available: https://scikit-learn.org/stable/modules/tree. html#tree-algorithms.

[8] Scikit-learn, "Random Forest Classifier," [Online], Available: https://scikit-learn.org/stable/modules/ generated/sklearn.ensemble.RandomForestClassifier.html?highlight=random%20forest#sklearn.ensemb le.RandomForestClassifier.

[9] A. B. Shaik and S. Srinivasan, "A Brief Survey on Random Forest Ensembles in Classification Model," Proc. of the International Conference on Innovative Computing and Communications, pp. 253-260, Springer, Singapore, 2019.

[10] Scikit-learn, "Extra Trees Classifier," [Online], Available: https://scikit-learn.org/stable/modules/ generated/sklearn.ensemble.ExtraTreesClassifier.html

[11] Scikit-learn, "AdaBoost Classifier," [Online], Available: https://scikit-learn.org/stable/modules/ generated/sklearn.ensemble.AdaBoostClassifier.html

[12] Towards Data Science, "AdaBoost Classifier Example in Python," [Online], Available: https://towardsdatascience.com/machine-learning-part-17-boosting-algorithms-adaboost-in-python-d00faac6c464

[13] S. Wu, C. Mei, H. Jin and D. Wang, "Android Unikernel: Gearing Mobile Code Offloading Towards Edge Computing," Future Generation Computer Systems, vol. 86, pp. 694-703, 2018.

[14] L. Liu, Z. Chang and X. Guo, "Socially Aware Dynamic Computation Offloading Scheme for Fog Computing System with Energy Harvesting Devices," IEEE Internet of Things Journal, vol. 5, no. 3, pp. 1869-1879, 2018.

[15] L. Liu, Z. Chang, X. Guo, S. Mao and T. Ristaniemi, "Multi-objective Optimization for Computation Offloading in Fog Computing," IEEE Internet of Things Journal, vol. 5, no. 1, pp. 283-294, 2017.

[16] Z. Tang, X. Zhou, F. Zhang, W. Jia and W. Zhao, "Migration Modeling and Learning Algorithms for Containers in Fog Computing," IEEE Trans. on Services Computing, vol. 12, no. 5, pp. 712-725, 2018.

[17] J. Du, L. Zhao, J. Feng and X. Chu, "Computation Offloading and Resource Allocation in Mixed Fog/Cloud Computing Systems with Min-max Fairness Guarantee," IEEE Transactions on Communications, vol. 66, no. 4, pp. 1594-1608, 2018.

[18] J. Shuja, A. Gani, K. Ko, K. So, S. Mustafa, S. A. Madani and M. K. Khan, "SIMDOM: A Framework

for SIMD Instruction Translation and Offloading in Heterogeneous Mobile Architectures," Transactions on Emerging Telecommunication Technologies, vol. 29, no. 4, p. e3174, 2018.

[19] H. Yan, X. Zhang, H. Chen, Y. Zhou, W. Bao and L. T. Yang, "DEED: Dynamic Energy-efficient Data Offloading for IoT Applications under Unstable Channel Conditions," Future Generation Computer Systems, vol. 96, pp. 425-437, 2019.

[20] Md. G. R. Alam, M. M. Hassan, Md. ZIa Uddin, A. Almogren and G. Fortino, "Autonomic Computation Offloading in Mobile Edge for IoT Applications," Future Generation Computer Systems, vol. 90, pp. 149-157, 2019.

[21] M. Aazam, S. Zeadally and K. A. Harras, "Offloading in Fog Computing for IoT: Review, Enabling Technologies and Research Opportunities," Future Generation Computer Systems, vol. 87, pp. 278-289, 2018.

[22] H. Lu, C. Gu, F. Luo, W. Ding and X. Liu, "Optimization of Lightweight Task Offloading Strategy for Mobile Edge Computing Based on Deep Reinforcement Learning," Future Generation Computer Systems, vol. 102, pp. 847-861, 2020.

[23] A. Jaddoa, G. Sakellari, E. Panaousis, G. Loukas and P. G. Sarigiannidis, "Dynamic Decision Support for Resource Offloading in Heterogeneous Internet of Things Environments," Simulation Modeling Practice and Theory, vol. 101, p.102019, [Online], Available: https://doi.org/10.1016/j.simpat.2019. 102019, 2020.

[24] C. Zhang, H.-H. Cho and C.-Y. Chen, "Emergency-level-based Healthcare Information Offloading over Fog Network," Peer-to-Peer Networking and Applications, vol. 13, no. 1, pp. 16-26, 2020.

[25] K. Xiao, Z. Gao, W. Shi, X. Qiu, Y. Yang and L. Rui, "EdgeABC: An Architecture for Task Offloading and Resource Allocation in the Internet of Things," Future Generation Computer Systems, vol. 107, pp. 498-508, 2020.

[26] I. A. Elgendy, W. Zhang, Y.-C. Tian and K. Li, "Resource Allocation and Computation Offloading with Data Security for Mobile Edge Computing," Future Generation Computer Systems, vol. 100, pp. 531-541, 2019.

[27] W. Tang, X. Zhao, W. Rafique, L. Qi, W. Dou and Q. Ni, "An Offloading Method Using Decentralized P2P-enabled Mobile Edge Servers in Edge Computing," Journal of Systems Architecture, vol. 94, pp. 1-13, 2019.

[28] Q. Wang, S. Guo, J. Liu and Y. Yang, "Energy-efficient Computation Offloading and Resource Allocation for Delay-sensitive Mobile Edge Computing," Sustainable Computing: Informatics and Systems, vol. 21, pp. 154-164, 2019.

[29] X. Xu, Y. Xue, L. Qi, Y. Yuan, X. Zhang, T. Umer and S. Wan, "An Edge Computing-enabled Computation Offloading Method with Privacy Preservation for Internet of Connected Vehicles," Future Generation Computer Systems, vol. 96, pp. 89-100, 2019.

[30] M. Adhikari and H. Gianey, "Energy Efficient Offloading Strategy in Fog/Cloud Environment for IoT Applications," Internet of Things, vol. 6, p. 100053, 2019.

[31] A. Bozorgchenani, S. Disabato, D. Tarchi and M. Roveri, "An Energy Harvesting Solution for Computation Offloading in Fog Computing Networks," Computer Communications, vol. 160, pp. 577-587, 2020.

[32] Y. E. M. Hamouda, "Modified Random Bit Climbing (Λ-MRBC) for Task Mapping and Scheduling in Wireless Sensor Networks," Jordanian Journal of Computers and Information Technology (JJCIT), vol. 5, no. 01, pp. 17-33, 2019.

[33] M. Zeng, Y. Li, K. Zhang, M. Waqas and D. Jin, "Incentive Mechanism Design for Computation Offloading in Heterogeneous Fog Computing: A Contract-based Approach," Proc. of IEEE International Conference on Communications (ICC), pp. 1-6, Kansas City, MO, USA, 2018.

[34] X. Zhu, S. Chen and G. Yang, "Energy and Delay Co-aware Computation Offloading with Deep Learning in Fog Computing Networks," Proc. of the 38th IEEE International Performance Computing and Communications Conference (IPCCC), pp. 1-6, London, UK, 2019.

[35] X. Zhao, L. Zhao and K. Liang, "An Energy Consumption Oriented Offloading Algorithm for Fog Computing," Proc. of the International Conference on Heterogeneous Networking for Quality, Reliability, Security and Robustness, pp. 293-301, Springer, Cham, 2016.

[36] J. Shuja, S. Mustafa, R. W. Ahmad, S. A. Madani, A. Gani and M. K. Khan, "Analysis of Vector Code

Offloading Framework in Heterogeneous Cloud and Edge Architectures," IEEE Access, vol. 5, pp. 24542-24554, 2017.

[37] M. Othman, A. N. Khan, J. Shuja and S. Mustafa, "Computation Offloading Cost Estimation in Mobile Cloud Application Models," Wireless Personal Communications, vol. 97, no. 3, pp. 4897-4920, 2017.

[38] J. Shuja, A. Gani, M. Habib ur Rehman, E. Ahmed, S. A. Madani, M. K. Khan and K. Ko, "Towards Native Code Offloading Based MCC Frameworks for Multimedia Applications: A Survey," Journal of Network and Computer Applications, vol. 75, pp. 335-354, 2016.

[39] X. Chen, "Decentralized Computation Offloading Game for Mobile Cloud Computing," IEEE Transactions on Parallel and Distributed Systems, vol. 26, no. 4, pp. 974-983, 2014.

[40] S. Sardellitti, G. Scutari and S. Barbarossa, "Joint Optimization of Radio and Computational Resources for Multicell Mobile-edge Computing," IEEE Transactions on Signal and Information Processing over Networks, vol. 1, no. 2, pp. 89-103, 2015.

[41] D. Rahbari and M. Nickray, "Task Offloading in Mobile Fog Computing by Classification and Regression Tree," Peer-to-Peer Networking and Applications, vol. 13, pp. 104-122, 2019.

[42] A. Yousefpour, C. Fung, T. Nguyen, K. Kadiyala, F. Jalali, A. Niakanlahiji, J. Kong and J. P. Jue, "All One Needs to Know about Fog Computing and Related Edge Computing Paradigms: A Complete Survey," Journal of Systems Architecture, vol. 98, pp. 289-330, 2019.

ملخص البحث:

إنّ تطبيق مصادر الحوسبة من خلال الأجهزة النّقّالة يُعرف بالحوسبة النّقّالة؛ وبين مراكز البيانات السّحابية والأجهزة يُسمى ذلك الحوسبة الضّبابية النّقّالة. لقد قمنا بتشغيل المحاكي () لتفريغ المهام في أجهزة حوسبة ضّبابية مناسبة أو في السّحابة أو في أجهزة نقّالة. وقمنا بتخزين مخرجات المحاكي في شكل مجموعة بياناتٍ ذات سِمات مميزة، بالإضافة الى درجة هدف. والأخيرة هي جهاز يتم فيه تفريغ المهام. أما سِمات المهام فهي: الأصالة، والسِّرية، والسَّلامة (الكمال)، والتّوافر، والسِّعة، والتكلفة.

تم استخدام عددٍ من المصنفات (DT; RF; Extra-trees; AdaBoost) ومقارنتها بناءً على قيم السِّمات، الى جانب رسم مخططات الشجرة لكل منها. ووفقاً للرسومات التي تم الحصول عليها من تلك المصنفات، تم استخلاص الوضع التتابعي لكلٍّ منها من "الجذر" وحتى "الأوراق" وإدخالها الى المحاكي. والجدير بالذكر أن ما تقوم به هذه المصنفات هو تحسين الشروط التي يتعين إدخالها الى الجزء المناظر من المحاكي. وقادت النتائج الى تحسين زمن الاستجابة للتفريغ باستخدام مصنفات RF و Extra-trees و AdaBoost على نحوٍ يفوق التحسين المتحقق باستخدام مصنف DT.