EAI Endorsed Transactions
on Mobile Communications and Applications
Research Article EAI.EU

Mobility and Fault Aware Adaptive Task Offloading in Heterogeneous Mobile Cloud Environments

Abdullah Lakhan[1], Xiaoping Li[1,*]

[1]School of Computer Science and Engineering, Key Laboratory of Computer Network and Information Integration, and Ministry of Education, Southeast University, Nanjing 211189, China
abdullah@seu.edu.cn,xpli@seu.edu.cn

Abstract

Nowadays, Mobile Cloud Computing (MCC) has become a predominant prototype for fetching the benefits of cloud computing to mobile devices' propinquity. Service availability in addition to performance enhancement and mobility features is a preliminary goal in MCC. This paper proposes a mobility aware adaptive offloading framework, known as Mob-Cloud, which includes a mobile device as a thick client, ad-hoc networking, cloudlet DC, and remote cloud services, to augment the performance and availability of the MCC services. However, the impact of dynamic changes in a mobile content (e.g., network status, bandwidth, latency, and location) for the task offloading model observes through proposing a mobility aware adaptive task offloading algorithm (MATOA), which makes a task offloading decision at runtime on selecting optimal wireless network channels and suitable resources for offloading. In this paper, we are formulating the decision problem, and it is well-known as an NP-hard problem. Nonetheless, MATOA has the following phases for the entire Mob-Cloud model: (i) adaptive offloading decision based on real-time information, (ii) workflow task scheduling phase, (iii) mobility model phase to motivate end-user invoke cloud services seamlessly while roaming, and (iv) fault-tolerant phase to deal with failure (either network or node). We carry out actual real-life experiments at the implemented instruments to evaluate the overall performance of the MATOA algorithm. Evaluation results prove that MATOA adopts dynamic changes on offloading decision during run-time, and meet an enormous reduction in the total response time with the improved service availability whilst in comparison with the baseline task offloading strategies.

Keywords: Task Offloading, Software Defined Network (SDN), MATOA, Mobility, DHEFT, Edge server (cloudlet), DC (data center), Workflow Task Scheduling.

Received on 09 January 2019, accepted on 28 January 2019, published on 31 January 2019

doi:10.4108/eai.3-9-2019.159947

*Corresponding author. Email: xpli@seu.edu.cn

1. Introduction

Recently, mobile business workflow applications, such as E-commerce, E-healthcare, Augmented Reality, 3D-Games, and Augmented Reality are increasingly progressively [1]. However, the latest mobile technologies are faced by resource-constrained issues (e.g., limited storage, lower processing speed, bandwidth utilization, and battery), and can not execute prior applications locally inside the mobile device[2]. To alleviate the resource limitation issue of mobile devices, a promising paradigm mobile cloud computing (MCC) is introduced. MCC allows to transfer compute intensive tasks of a mobile program to the cloud for processing [3]. Since, task offloading is a method in MCC, which divides the into fine-grained small tasks and makes a decision where to offload them for execution. In more general terms, task offloading computationally divides application execution between the mobile devices and cloud computing networks [4]. In the conception of task offloading, cloud computing networks is classified into the following paradigms, such as ad-hoc cloud network (WANET) paradigm, decentralized cloudlet computing paradigm, and centralized cloud computing paradigm[5]. Whereas big giants, particularly Amazon, IBM, Google, and Microsoft provide pay-as-you-go public cloud services via internet technology for the mobile workflow applications[6]. A decentralized cloudlet is a mobility-enhanced and latency optimal computing paradigm that is placed at the edge of the mobile networks. [7]. Nowadays, mobility is a commonly used feature which allows the mobile-end user to invoke the cloud services for running application during roaming from any place within network range [8]. Since WANET paradigm is supportive when offloading performing

between different devices (i.e., mobile phones or vehicle devices).

In this paper, we are formulating mobility and fault aware adaptive task offloading problem in the heterogeneous MCC environments (i.e., local device, WANET, cloudlet, public cloud). The is to minimize the application response time as well as satisfies the deadline requirements. The response time includes communication time and processing during processing in the mobile cloud environment. Since each workflow application has deadline constraint. We model the workflow application into the fine-grained tasks with different characteristics. All workflow applications have stringent requirements and all tasks must be completed under deadline constraint. In the heterogeneous MCC, the fundamental requirements of mobile-end users are their mobility based cloud services, therefore there are numerous factors that can antagonistically influence task offloading performance of an application. Those are the network bandwidth impediment between mobile phones and cloud servers during offloading/downloading and exchange data. However, it is most important to design task offloading strategy that must be adaptive and adapt any environmental changes while end-user roaming among different networks.

In the literature, many studies have already been proposed task offloading techniques in the MCC environment for mobile workflow applications. For instance [5], [6], [7], [8], [9], [10], [11], [12], [13] devised a task offloading techniques assuming the wireless communication network and cloud services are always remained stable. However, the earlier supposition is rather impractical. As in daily practice a wireless network context intermittently changes due to traffic at different time. Whereas, fluctuation in the cloud services often happens in different time zones. Recently, with the advance in the wireless technology a mobile device has access to the many wireless channels, for example WiFi, cellular network and Bluetooth. Every connection performs in a different ways in terms of speed and strength. For mobile workflow applications, we have different communication channels and different cloud paradigms, thus mobility and fault aware task offloading in a dynamic environment is an important problem to solve. The following questions that needed to be addressed in an adaptive environment.

- Adaptive Task Offloading: Since many factors, such as network bandwidth, connection type, end to end latency, cloud services availability after task offloading decision during time. However, static task offloading algorithms proposed through preceding works with a set bandwidth assumption are flawed for mobility feature in the MCC environment [14]. The partitioning algorithms ought to be adaptive to easily adopt network changes and should have an awareness of the mobility model. Since the network condition is merely measurable at runtime, the task offloading must be an adaptive and mobility aware of the MCC domain [15].

- Connection Failure: To get access to the cloud services is normally stimulated with the aid of uncontrollable factors, such as the instability and intermittent of wireless networks. Due to end-user mobility and weaker bandwidth of wireless connection can also be caused of connection failure. When a network connection unexpectedly breaks down, an offloaded computation will suffer extremely, overall performance of an application. Question how to deal with this obstacle?

- Node Failure: cloud services sometime may be unreachable due to either due to node failure or service out of communication of range. Question how to fix this issue?

- Fault Recovery Mechanism: Due to either connection failure or node, how to fix them without compromising the entire application performance.

To answer the above-cited questions, this work differs from the preceding work by means of focusing on the MCC mission offloading hassle, wherein there are four key issues to be addressed. We formulated mobility aware task offloading problem as an NP-hard optimization problem. We used a dynamic programming technique to divide the considered problem into sub problems. For instance mobility aware adaptive task offloading problem involves following steps to make the optimal offloading decision: (i) task offloading process, (ii) task scheduling process, (iii) mobility mechanism, and (iv) fault-recovery management. To cope up with the issue of dynamic change in the environment due to the mobility we proposed a novel mobility aware task offloading algorithm (MATOA), which adopt dynamic changes at run-time due to the mobility features in order to manage application performance.

- Adaptive Task Offloading: To deal with the first question,we formulated adaptive task offloading decision dilemma as a Multi-criteria decision making (MCDM) problem. It refers to making choice of the ideal alternative (e.g., 3G/4G, WiFi, Cellular, mobile device, WANET, cloudlet DC, and public cloud) from amongst a finite set of decision alternatives in phrases of more than one with conflicting criteria. The offloading system examines a few parameters along with network bandwidth,

execution time, connection type, service availability, latency, and round-trip time to make an optimal offloading selection for application partitioning between mobile and cloud. Although the aforementioned parameters appear to be virtually adequate to adopt any environmental changes. To cope with circumstances we modified a Technique for Order Preference by Similarity to an Ideal Solution (MTOPSIS) method [16] to take offloading action in the dynamic MCC environment.

- Connection Failure: to cope up with the question, we propose a novel mob-cloud architecture, which handles both failures (network and node) with the assist of the fault-awareness algorithm. This algorithm employs two primary techniques such as, dynamic task re-clustering and failover Mechanisms for Distributed SDN Controllers (FMDSC) to deal with both failure.

- Task Scheduling: All considered applications are directed acyclic graph (DAG) based and has different size of fine-grained tasks. However, cloud resources are organized in heterogeneous approach. For task scheduling, we modified traditional modified heterogeneous earliest finish time (MHEFT) heuristic [17] to deal workflow task scheduling problem in mob-cloud domain, where workflow scheduler map application tasks on the local mobile device resource, wireless network resource and cloud resources.

- Mobility Model: We introduce a mobility algorithm based on Markovian random waypoint model [18] which allows us to create diverse mobility location pattern inside the given movement network domain. The mobility model allows adjusting random pause times and the velocity speed in a given network. The algorithm tries to choose an optimal state (offloading values) in order to quality of the application remain stable.

Summary, with the best knowledge, mobility aware adaptive task offloading has not been studied yet in a dynamic environment. We proposed a novel MCC mob-cloud architecture with heterogeneous cloud services availability in order to support any kind of application from any place. Due to mobility feature, it is difficult to maintain application performing when user roaming among different base stations. Most importantly, mob-cloud support any kind of application requirements (i.e., security, compression-technique for huge data, big data processing before offload in the remote cloud) with some additional component. However, in this paper mob-cloud has primary two components to deal with the considered problem. It is noteworthy, local device and WANET has a different mechanism for task execution, local device is only a single client machine, whereas WANET is made of different devices design cloud-device in a temporary network.

The rest of the paper is organized as follows. Section 2 elaborates related work and 3 explains the problem description and formalizes the problem under study. The MATOA is proposed for the considered problem in Section 4 that describes proposed methods. Section 5 evaluates the simulation part, section 6 is summarized the paper conclusion.

2. Related work

In literature, a number of comprehensive studies have already been investigated for task offloading in MCC environment. However, latest technologies IoT (Internet of Things) devices are different in computing power point of view. Nonetheless, In this paper, proposed mob-cloud is not specific to the mobile device, but this work focuses only mobile device as a IoT device, that offloaded their compute intensive tasks to the cloud for processing. Whereas, mobility management and fault tolerant based task offloading strategy would be satisfied the service requirements for IoT devices. The following studies have already been proposed different strategies to task offloading in MCC environment

Chun and at. al proposed CloneCloud offloading framework in [5]. The aim was to improve the mobile battery life and augment the application operation. Thread level granularity is used for the application partitioning for performing the computational offloading. Energy efficient computational offloading framework Think-Air is proposed in [6]. The primary objective was to minimize the energy consumption of mobile device and prolong the battery performance. The code level (binaries) computational offloading is performed in the proposed framework. In [7–9] Cuckoo, MAUI, and JADE were proposed their energy efficient frameworks for computational offloading that is whether offloads or not during run time. The primary objective was to augment the mobile battery life and reduce the burden of the application developer via run time offloading decision. Delay sensitive applications are required to execute within shorten time, however, contemporary cloud services are available on long distance WAN, it could be incurred by longer end to end delay. To cope with above problem a Cloudlet framework is proposed by satyanarayanan et.al in [10]. The primary objective is to bring cloud computing capabilities closer to the mobile user in order to minimize the total delay for delay sensitive application. However, many of above works focused on task offloading strategies without considering mobility features in an adaptive environment.

Latency sensitive tasks of application can offload near cloudlet, which is placed between mobile devices and public cloud at the edge of network[11]. The

offloading strategy is proposed based on mixed fog and cloud architecture in order to support delay sensitive application [12]. Online Task offloading strategies investigated in their respective papers [13–15], the primary goal how to do application partitioning in a dynamic environment in MCC architecture to invoke cloud services. Latency aware optimal workload assignment to cloud computing investigated in these papers[16, 17, 19], however, [19] considered heterogeneous mobile cloud environment (ad-hoc, edge cloud and remote) to make offloading decision. [20, 21] proposed decentralized mobile cloud environment in order to improve resource availability for task offloading. These efforts focused on delay minimization for real time application in MCC paradigm.

Nowadays, each IoT technology is moveable, however, mobility and fault aware adaptive task offloading would be more appropriate in a dynamic environment. With the best knowledge, mobility aware adaptive task offloading with a fault tolerant mechanism has not been studied yet. In order to extend the feature of the task offloading strategy, we proposed the MATOA algorithm framework, which deals mobile application performance during roaming and invoking suitable services in the shadow of fault-tolerant awareness.

3. Proposed Description

The mobility and fault aware adaptive task offloading in heterogeneous cloud networks is a difficult problem. Therefore, proper handling of application tasks in the dynamic environment is a challenging task. We proposed mob-cloud architecture similar to the existing architectures in [22], there are two main components in the mob-cloud architecture: client-side component and the cloud side component as shown in Fig. 1. There are four major modules on client sides: Content Agent module, Decision Engine module, Workflow Scheduler module, and Failure Manager module. Content Agent module collects real-time data of multiple parameters, such as an application program, current network bandwidth, latency, round-trip time, and cloud resource estimation at runtime [22]. Whereas, Content Agent module assists Decision Engine module to make adaptive task offloading decisions based on collecting real-time information of multiple parameters. Then the decision engine module divides the resilient workflow application in a local cluster of tasks (e.g., lightweight tasks) and a remote cluster of tasks (e.g., compute-intensive tasks). The Workflow Scheduler module is the responsible for task scheduling and has two primary jobs: first, it schedules the local cluster of tasks on the heterogeneous mobile cores second, schedule the remote cluster of tasks on a fitting available efficient wireless network channel to the cloud server. It is a noteworthy Workflow scheduler only schedule one task at a time, and able to schedule tasks either on the mobile device or a wireless channel network, it cannot schedule the tasks on the cloud server, but it can predict the task process time on the cloud server. The Failure Manager module has a more important role and identifies two preliminary sub failure modules during runtime of an application: such as a network failure module (NF) and a node or resource failure module (RF). Due to the mobility feature of network contents change due to many reasons, such as weak signals, high communication latency, and lower bandwidth could degrade overall performance. Thus, because of dynamic network context the transmission time for offloading and downloading data for running application exceed the given deadline, in this case, possibly many application tasks could be failed due to either weak network or unavailability of the network. NF module adaptively lets know to the failure manager about failed tasks and show an alternative list of wireless networks. Whereas RF module monitors the remote set of task execution process on assigned resources. Once it detects any failure due to node capacity, or unavailability it would collect tuple of information tuple information (e.g., task ID, location ID, the point of failure, and the result obtained) and sent back to the failure manager for rescheduling. It is noteworthy; failure only can be detected after scheduling.

3.1. Problem Formulation and System Model

Application Partitioning. Generally, the mobile workflow application usually consists of composite dependent components (e.g., tasks). However, some of them compute intensive components of the application will offload their data and operation to the cloud remote via partitioning method. There are many methods for application partitioning, such as programmer define a method, linear programming, profile-based, simulation based, graph based, and last but not least inference algorithm, all partitioning methods detail can be presented in [22]. In this paper, we only focus on profile-based partitioning, because of the considered problem mob-cloud has many parameters to do application partitioning, namely task execution cost, bandwidth, CPU speed, network type, cloud service availability, and round-trip time as illustrated. The propose MATOA algorithm make offloading decision based on collecting real-time information that is provided by different profilers, likewise, program, network, resource profiler individually.

Task Characterization. After partitioning process, application can be broken down into multiple types of fine-grained objects, modules, classes, bundles, threads, functions, and components [22]. Since, this paper only

Table 1. Mathematical Notation

Notation	Description
G	A mobile application
G_D	Application G Deadline
V	All tasks of a G
l_i	Location of task v_i
N	Number of mobile cores C
C_k	k^{th} core in N
f_k	CPU frequency of the k^{th} core in MIPS
ϵ_m	Capacity of a mobile device
T_i^{ws}	Offloading time of task v_i
T_i^{wr}	Receiving data time of task v_i
$T_i^l e$	Execution time of task v_i locally
$T_i^c e$	Execution time of task v_i remotely
ST_i^l	Setup time of v_i
ST_i^c	Setup time of a v_i
F_i^l	Finish time of a v_i
F_i^c	Finish time of a v_i
$\mathcal{V}$	A virtual machine
M	Number of virtual machines $\mathcal{V}$
$\mathcal{V}_j$	j^{th} virtual machine in M
ζ_j	Speed of j^th virtual machine in M
P	Number of virtual machines in cloudlet server
cp	p^{th} virtual machine in P
τ_p	The speed of p^{th} virtual machine in P
D	Number of devices in Ad-hoc
dn	The d^{th} device in the D
μ_d	The CPU speed of d^{th} device
ϵ_c	capacity of all cloud resources
$x_{i,j}$	Assignment of task v_i on virtual machine j
$y_{i,k}$	Assignment of task v_i on mobile k^{th}core

focuses on components based fine-grained for application partition. Each component can be assumed as a task, after partitioning there are two kinds tasks for execution, named remote tasks and local tasks. The Decision Engine makes a task offloading decision typically, which tasks are executed locally (a local cluster of tasks), which are offloaded to be executed on a cloud (a remote cluster of tasks). In general terms, based on real-time information provided by different profiler, Decision Engine divides application components into a local cluster of tasks V_l and a remote cluster of task V_c in order to manage application performance. Each task has input/output data, i.e., $data/data'$, that require the CPU instruction to do execution, however, local cluster of tasks can be retained inside mobile device, because they required lower CPU instruction for execution and the device has sufficient resources to execute all tasks. However, remote clusters of tasks require huge CPU cycle of instruction, thus they would be offloaded to the rich resource cloud computing, but incur extra network round-trip time RTT, such as communication time. Generally, total application response time is the combination of computation time and communication for execution.

Different Topologies. There are many types topologies to represent the application task structures, particularly Linear topology, Loop topology, Tree topology, Mesh topology, and call graph [18]. This paper only considers call graph as a directed acyclic task graph (DAG) topology, where a workflow application can be modeled as a DAG e.g., $G(V, E)$. Whereas, each node $v_i \in V$ represents a task and edge $e(v_i, v_j) \in E$ represents precedence constraint that a task v_i should complete its execution before a task v_j starts execution. The application G has a V number of tasks, whereas, in the graph, a task v_1 is the entry task and v_{10} is an exit task. For each task v_i we define $data_i$ and $data_i'$ input data and output data, each of the application is restricted by their deadlines G_D. $V_l \cup V_c$=V denotes the application

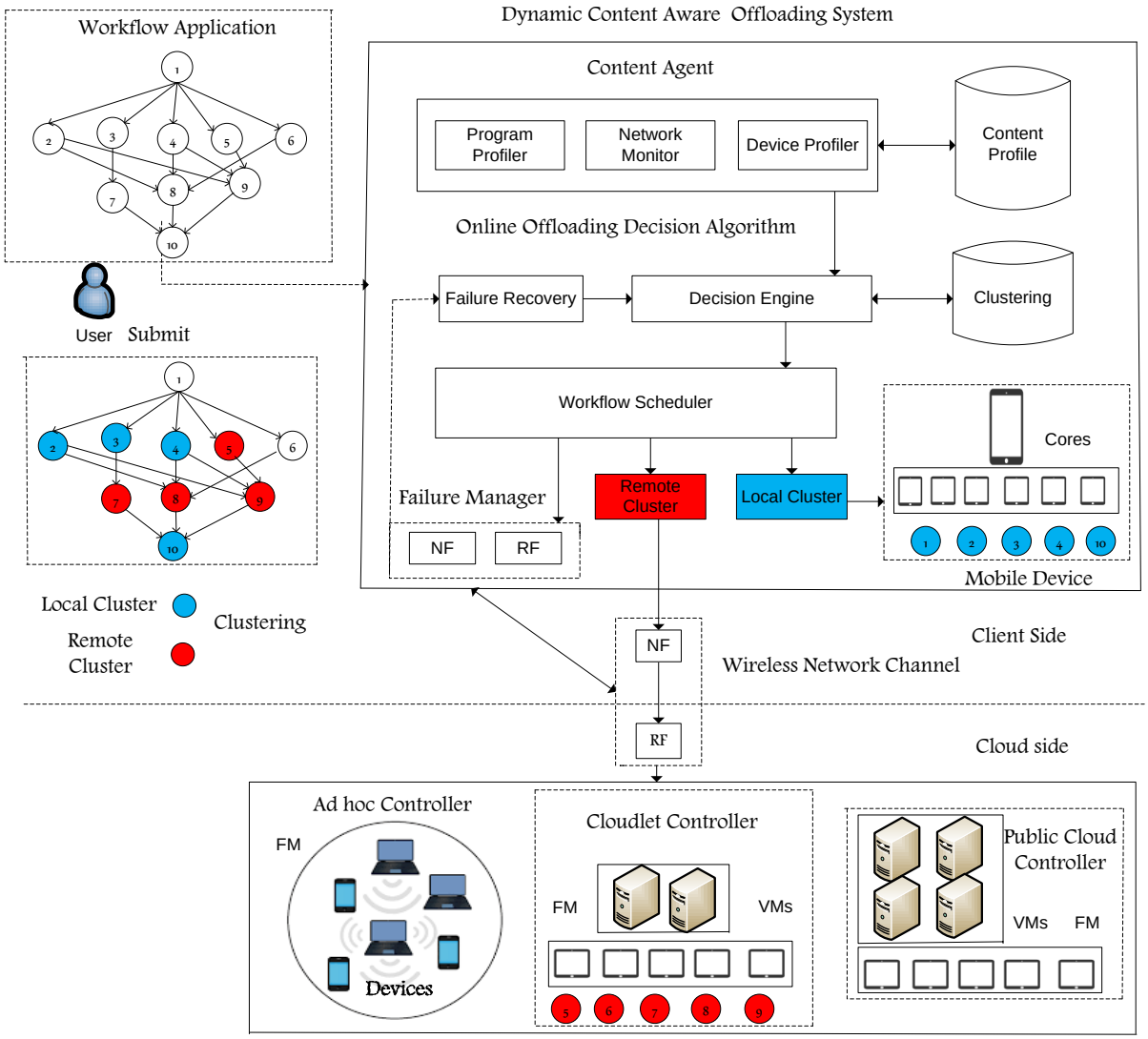

Figure 1. Modified mob-cloud Architecture

has two clusters of tasks such as the local cluster of tasks and the remote cluster of tasks.

MCC Resources. The modified MCC system model leverages with four types of computing resources for application execution, such as mobile device, local cloud servers, WANET cloud devices, and public cloud servers as shown in Fig. 1. Since, WANET has a set of heterogeneous devices, i.e. $D=\{d_1, d_2, d_3, ..., N\}$. Each device d_k has a different frequency speed, i.e., $f_k=\{k=1, k=2, ..., N\}$. Whereas, cloudlet DC has a set of heterogeneous virtual machines, i.e. $K=\{k_1, k_2, k_3, ..., N\}$. Each k^{th} virtual machine has a different frequency speed, i.e., $\zeta_k=\{k=1, k=2, ..., N\}$. Whereas, public cloud has a set of heterogeneous virtual machines, i.e. $\mathcal{V}=\{\mathcal{V}_1, \mathcal{V}_2, \mathcal{V}_3, ..., M\}$. Each $\mathcal{V}_j$ virtual machine has a different frequency speed, i.e., $\zeta_j=\{j=1, j=2, ..., M\}$.

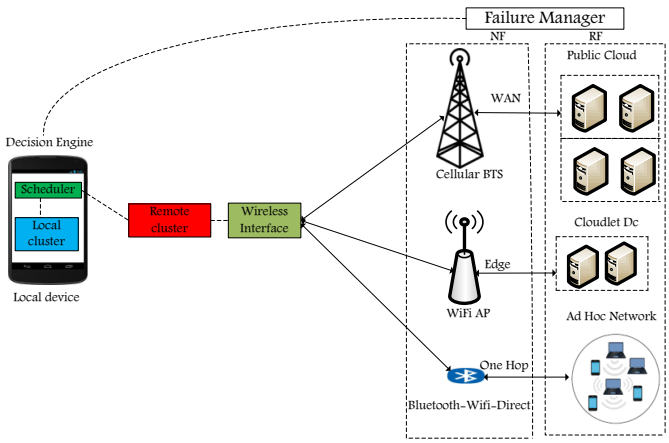

Figure 2. Fault Tolerant Offloading Mechanism

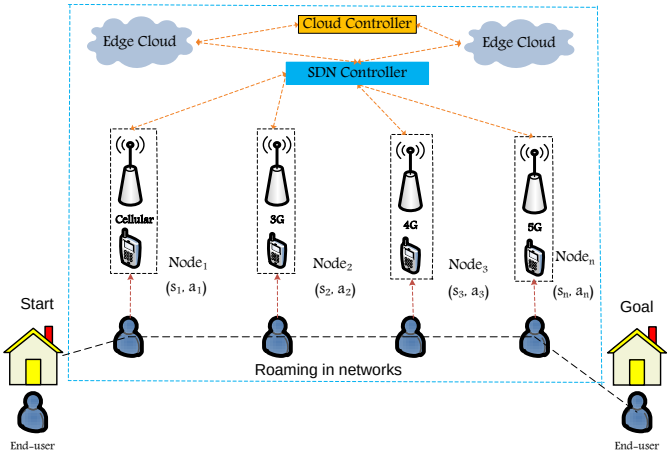

Figure 3. Mobility Model Waypoint States Scenario

Mobility Model. In the MCC, mobile users definitely roaming among multiple places while invoking different services for task execution. Furthermore, during user's roaming, network values (e.g., bandwidth, latency, upload, and download transmission rates) could be varied along with location. To find this phenomenon study proposes a modified random waypoint mobility model (MRWMM), and formulates it as a Markov decision process model (MDPM). The MDPM is defined as the navigating between numerous sets of states $\{s_1, s_2, s_3, ..., S\}$, and set of actions $\{a_1, a_2, a_3, ..., A\}$ as illustrated in Fig. 3, where at each state (s, a) has a task offloading action and it estimate either new values are aversive stimulus (e.g., penalty) or positive stimulus (e.g., reward). $P_a(s, s') = P_r(s_{t+1} = s' \mid s_t = s, a_t = a)$ as compared to previous one, thus the probability of the transition when end-user move from one state to another under the initial task offloading action a. Where $R_a(s, s')$ is the immediate stimulus reward if next state offloading decision gain some positive value rather than aversive stimulus. The system estimates pause time and pre-defined velocity speed $(\{t_min, t_{max}\}, (\{v_min, v_{max}\}$ during mobility whenever an end-user chooses a random selection at each waypoint. The mechanism can be assumed mobility as a service, where each state assume end-user offload and invoke services via base-stations (BSs) and cloudlets DC (data center). Mobility gets optimal performance due to efficient path provided via BS to BS.

Fault Tolerant. Fig. 2 elucidates that failure manager component traces and monitor the failure either due to the communication failure or node failure, it annotates these reports to the decision to recover failure tasks from effecting point not from scratch based fault recovery algorithm. Since, the Failure Manager (NT) has NF module and RF module to detect failure. The tracing information by a NT module is shown in the following expression:

$$NT = \begin{cases} NF & Cellular, WiFi, Bluetooth \\ ,RF & local, WANET, cloudlet, cloud \end{cases}$$

Task Scheduling. The binary variable y, represents the assignment of a task i to a k^{th} mobile core $y_{ik} = \{0, 1\}$ namely:

$$y_{i,k} = \begin{cases} 1, & \text{Assigned task to core} \\ 0, & \text{otherwise,} \end{cases} \quad (1)$$

if a task v_i is scheduled locally, then the average execution time of a task v_i is calculated by:

$$T_i^e = \sum_{k=1}^{N} y_{i,k} . \frac{data_i}{f_k}, \quad (2)$$

T_i^e is the task execution time at locally, the average execution time of local cluster describes in the following:

$$T_i^{le} = \sum_{k=1}^{N} \sum_{i=1}^{V_l} T_i^e, \quad (3)$$

if a task v_i is scheduled for the cloud, the execution time of the v_i it depends on the location:

$$T_i^c = \begin{cases} \sum_{j=1}^{M} x_{i,j} \frac{data_i}{\zeta_j} & if \;\; l_i = 1 \\ , \sum_{p=1}^{P} x_{i,p} \frac{data_i}{\zeta_k} & if \;\; l_i = 2 \\ , \sum_{d=1}^{D} x_{i,d} \frac{data_i}{f_k} & if \;\; l_i = 3, \end{cases}$$

since T_i^c is the average execution time, if a task V_i immediate predecessor of a task v_j then communication time depends upon the service location, i.e., (either local mobile cores or offloaded to the cloud servers). Whenever, a task v_i offloads to the cloud server, the communication time can express in the following way:

$$T_i^s = \begin{cases} \frac{data_i}{S^B} & if \;\; l_i = 3 \\ , \frac{data_i}{S^B} & if \;\; l_i = 2 \\ , \frac{data_i}{S^B} & if \;\; l_i = 1. \end{cases}$$

T_i^s is communication time for offloaded task v_i and it depends upon the site that determines by a variable l_i. The average execution time the remote cluster express in the following way:

$$T_i^{ce} = \sum_{i=1}^{V_c} T_i^c, \quad (4)$$

The communication time of task v_i when it returns results back to the mobile device and it is similar to the equation (3).

$$T_i^r = \frac{data_i'}{R^B}, \quad (5)$$

All its immediate predecessors must schedule before a task v_i. However, if a task v_i schedule on a mobile core, the setup-time can calculate in the following way:

$$ST_i^l = \max_{v_j \in pred(v_i)} \max\{FT_j^l, FT_j^{wr}\}. \quad (6)$$

FT_j^l, FT_j^{wr} denotes before schedule a task v_i predecessor v_j finished its execution or FT_j^{wr} result received back from cloud server after v_j finish its execution. The setup time of a remote cluster of tasks is scheduled on the wireless network channel:

$$ST_i^{ws} = \max_{v_j \in pred(v_i)} \max\{FT_j^l, FT_j^{ws}\}. \quad (7)$$

The setup time of a task v_i onto the cloud server:

$$ST_i^c = \max\{FT_i^{ws}, \max_{v_j \in pred(v_i)} FT_j^c\}, \quad (8)$$

setup time for the cloud to send back the results of a task v_i to the mobile device:

$$ST_i^{wr} = FT_i^c. \quad (9)$$

Where if a task schedule v_i on the cloud server, the round-trip time calculate in the following way:

$$RTT = T_i^s + T_i^{ce} + T_i^r. \quad (10)$$

Since, the total average response time of the application is expressed in the following way:

$$T^{total} = \sum_{i=1}^{V_l} ST_i^l + T_i^{le} + \sum_{i=1}^{V_c} RTT. \quad (11)$$

The considered problem is mathematically modelled as below, whereas, notations are explained in Table 1.

$$\min T^{total} \quad (12)$$

subject to

$$\sum_{i=1}^{V_l} y_{i,k}.data_i \leq \epsilon_m, \forall k \in K, \forall v_i \in V_l, \quad (13)$$

$$\sum_{i=1}^{V_c} data_i \leq \epsilon_c, \forall \epsilon_c \in M \cup P \cup D, \forall v_i \in V_c, \quad (14)$$

$$\sum_{i=1}^{V} y_{i,k} = 1, \forall k \in C, \quad (15)$$

$$\sum_{k=1}^{N} y_{i,k} = 1, \forall i \in V, \quad (16)$$

$$T^{total} \leq G_D, \quad (17)$$

$$y_{i,k}\{0,1\}. \quad (18)$$

However, equation (12) shows the average response time of an application G. Where equation (13) and equation (14) show that the size of the local cluster less than the total capacity of the mobile device and similar requested size of the remote cluster less than the total capacity of all types of cloud resource. Since, equation (15) and (16) that every resource is assigned to exactly one task, every task is assigned exactly one resource. Application average execution time less than a given deadline is defined in equation (17). The binary variable is defined in equation (18).

4. Proposed MATOA Algorithm

For the considered problem minimizes total response is determined mobility aware adaptive task offloading for a workflow application in a heterogeneous mobile cloud environment can be solved via multiple steps. The problem that we are investigating is the decision problem, and it is a well-known NP-hard problem. We apply dynamic programming approach to this problem, to divide the main problem into subproblems. The problem mobility aware adaptive task offloading problem has the following phases: (i) task offloading engine phase, (ii) task scheduling phase, (iii) mobility model phase, and (iv) fault aware phase and calculated via Algorithm 1. To deal with all phases of the problem we have proposed mobility aware adaptive task offloading algorithm (MATOA). For phase (i) to make potential and optimal task offloading decision we propose a modified MTOPSIS algorithm, because there are multiple parameters and alternative involve in the task offloading decision, it takes DAG as an input, and generate the result as two clusters (i.e., blue local cluster and red remote cluster). Whereas, phase (ii) to map resource to the clusters we proposed a modified MHEFT method because resources are heterogeneous and clusters have task-precedence need while invoking cloud services for running applications. Since phase (iii) allows the benefit of mobility features while end-user roaming on a given network (base-station to a base-station efficient way to roam with edge cloud services and for WANET network, not public cloud). To deal with the mobility feature system proposed mobility algorithm based on Markovian mobility model, where each base-station assume as a state, performance can be improved via reward value to reach at last destination. The final phase (iv) is a fault awareness for running application, that might be rise due to the mobility feature when uploading and downloading speed for task offloading could be changed, the fault aware algorithm employs two strategies, such as network failure strategy (NF) and node failure strategy (RF), to detect either task fail due to wireless connection or node

capacity constraint. We will discuss all phases in detail in the following sections.

4.1. Adaptive Task Offloading Phase

The latter offloading orchestrate policies only considered two core threshold values network speed and energy consumption when they fabricate offloading decision. However, in dynamic environment many factors can degrade application performance (i.e., bandwidth, latency, signal strength, and delay), whereas the previously mentioned threshold criteria is not suitable in MCC environment when do application partitioning. To deal with the above situation our study considers to fabricate offloading decision, for suppose available network type, bandwidth, latency, resource availability, execution cost, and network congestion. mob-cloud follow a modified multi-criteria decision making (MCDM) method [23] in the MATOA framework to make a task offloading decision as illustrated. For MCDM, system employs modified Technique in order-of Preference by Similar to Ideal Solution (TOPSIS) for appropriate offloading decision. Traditional TOPSIS only considers parameters, however, we need to modify and some additional module which involves an application partitioning module because only light tasks are executed locally, and compute intensive tasks offloaded to the cloud server. A modified MTOPSIS has the following steps (i) collect the device context information and application scale down (i.e., task size), (ii) real time information for selecting appropriate network connection (offloading round-trip time, congestion, latency, and bandwidth), (iii) availability of cloud resources (ad-hoc devices, cloudlet, and public cloud), system partition the application into local cluster of tasks and remote cluster of tasks based on above mentioned steps by using a MTOPSIS method. We obtained some relative weight for criteria $\omega = \{0.1, 0.4, 0.3, 0.5, 0.7, 0.2\}$ for task size, latency, round-trip time, congestion, inference, and execution time, respectively, by using the analytic hierarchy process (AHP) [23]. It is a pair-wise approach, and their results are shown in the matrix X.

$$X = \begin{bmatrix} a_{11}a_{12}a_{13}...a_{16} \\ a_{21}a_{22}a_{23}...a_{26} \\ a_{31}a_{32}a_{33}...a_{36} \\ a_{41}a_{42}a_{43}...a_{46} \\ a_{51}a_{52}a_{53}...a_{56} \\ a_{61}a_{62}a_{63}...a_{66} \end{bmatrix}, a_{nn} = 1.a_{mn} = \frac{1}{a_{nn}}. \tag{19}$$

The $m \times n(3 \times 6)$ shows three alternative cloud resources (e.g., local devices could be ad-hoc, local cloudlet edge server, and public cloud) and six criteria as mentioned above. The pairwise comparisons of giving criteria are produced based on the normalized comparison scale on nine levels as illustrated in Table 2. MTOPSIS employs matrix X to compute the weight of the criteria by getting a eigenvector ω which associated with addition to the prime eigenvalue λ_{max}. As generally, the outcome of the AHP technique is stringently associated with the reliability of the pairwise comparison, so it is compulsory to estimate the reliability index (RI) in the following way:

$$\begin{aligned} RI &= \frac{\lambda_{max} - n}{(n-1)}, \\ RR &= \frac{RI}{(n-1) \times Random - Index}, \end{aligned} \tag{20}$$

RR is the reliability index ration of RI, and it should be less than given threshold 0.1, then it has a suitable relative weight output. Successive relative weights are produced, the possibility of multi-criteria derive via $N_{M(x_{mn})^k}\ x$, since m is a fascinating alternative (local devices, cloudlet, and public cloud) with n multi-criteria (available network, resource availability, bandwidth, task size, latency, congestion) can be further normalized in the following way:

$$N_{M(x_{mn})^k} = \frac{M(x_{mn})^k}{\sum_{n=1}^{6} M_{mn}^2} \tag{21}$$

x is a any real number R to explore the choices may be criteria and alternative can be varied, where k be a k^{th} decision maker. It is noteworthy, it could be possible system has a set decision making choices $\{k_1, k_2, k_3, ..., K\}$.

$$M_w = \omega_n \times N, \tag{22}$$

where ω_n be a weight which is already initialized above. Find out the affirmative best solution and the aversive solutions for each decision maker k from weight matrix.

$$\begin{aligned} A^{k+} &= \langle \min t_{mn} \mid m = 1, 2, 3, .., 6 \mid n \in J^+ \rangle, \\ &\quad \langle \min t_{mn} \mid m = 1, 2, 3, .., 6 \mid n \in J^- \rangle \\ A^{k-} &= \langle \min t_{mn} \mid m = 1, 2, 3, .., 6 \mid n \in J^- \rangle, \\ &\quad \langle \min t_{mn} \mid m = 1, 2, 3, .., 6 \mid n \in J^+ \rangle \end{aligned} \tag{23}$$

where J^+ is a positive solution of the decision maker for application objective, where J^- is aversive (negative) solution to the objective. It is natural to consider the real time information related to the available wireless network via network profiler, system uses Euclidean distance matrix among all possible alternatives and can be calculated in this way:

$$\begin{aligned} D_m^+ &= \sqrt{\sum_{n=1}^{6} (t_{mn} - t_{mn}^+)^2}, \\ D_m^- &= \sqrt{\sum_{n=1}^{6} (t_{mn} - t_{mn}^-)^2}, \end{aligned} \tag{24}$$

Table 2. Scale & Definition

Definition	Strength of importance
Uniformly significant	1
Fairly supplementary important	3
Robustly supplementary important	5
Very robustly supplementary important	7
Extremely robustly supplementary important	9
Intermediary	2 4 6 8

since D_m^+ and D_m^- show best and worst solution, respectively, for each alternative. The task offloading algorithm chooses highest rank solution H_m solution from all alternative as shows follow:

$$H_m = \frac{D_m^-}{D_m^+ + D_m^-}. \tag{25}$$

$$\sum v_i = 1^{V_l} + \sum v_i = 1^{V_c} = H_m. \tag{26}$$

Algorithm 2 optimally partition the application into

Algorithm 1: MATOA

Input : $v_i \in V$;
Output: min T^{total}, s;
1 **begin**
2 $T \leftarrow 0$;
3 $NT[] \leftarrow 0$;
4 $S[] \leftarrow 0$;
5 Application Partitioning based on equation (22)
Scheduling $v_i \in V_l$,$v_i \in V_c$;
6 Mobility Model;
7 **foreach** $(v_i \in V_l)$ **do**
8 $lsum = \sum_{i=1}^{V_l} ST_i^l + T_i^{le}\ NT[] \leftarrow v_i$;
9 T=$T + NT[] + lsum$
10 **foreach** $(v_i \in V_c)$ **do**
11 $csum = \sum_{i=1}^{V_c} + RTT.\ T$=$T + NT[] + csum$
$NT[] \leftarrow v_i$;
12 **foreach** $(s \in S)$ **do**
13 Apply Mobility Model;
14 $s[] \leftarrow T$;
15 **return** T, s;

local cluster and remote cluster. Whereas, local cluster consists of tasks which have to be retained locally on a mobile device, remote cluster of tasks offloaded to the remote cloud via wireless network. In Algorithm 2, Line 2 shows a decision function which collects information via real-time profile technologies. Line 4 illustrates application partition clusters (local and remote cluster). Lines 6-8 calculate the task execution time at local and remote cloud. Lines 10-19 it checks where to offload (WANET, cloudlet DC, and public) on what wireless network (WiFi, Bluetooth, Cellular network). Line 20 return optimal T after entire task assignment.

Algorithm 2: Adaptive Task Offloading

Input : $G(V, E)$;
Output: min T^{total};
1 **begin**
2 $func \leftarrow getDecision(context, V)$;
3 $context \leftarrow profile$;
4 $V \leftarrow V_l \cap V_c$;
5 $T \leftarrow 0$;
6 **foreach** $(v_i \in V_l)$ **do**
7 T_i^{le}
8 **foreach** $(v_i \in V_c)$ **do**
9 T_i^s
10 **if** $NT = 1$ **then**
11 **if** l_i=*1* **then**
12 **return** $T \leftarrow MinCost(\sum T_i^{le}, \sum T_i^c)$;
13 **else**
14 **return** $T \leftarrow MinCost(local, null)$;
15 **if** $NT = 2\ Or\ NT = 3$ **then**
16 **if** l_i=*1,2,3* **then**
17 **return** $T \leftarrow MinCost(\sum T_i^{le}, \sum T_i^{cl}, \sum T_i^c)$;
18 **else**
19 **return** $T \leftarrow MinCost(\sum T_i^{le}, \sum T_i^d)$;
20 **return** T;

4.2. Task Scheduling Phase

The clustering based task scheduling is a promising and efficient method when there are more than one resource types (local device, WANET, cloudlet DC, public cloud) [24]. However, for the considered problem, we have different fine-grained size of tasks and heterogeneous resource in different types, so that HEFT is an appropriate heuristic to map application tasks on heterogeneous resources [25]. In our case, we modified MHEFT heuristic, because traditional

HEFT only schedule tasks on cloud resources, but mob-cloud schedule tasks on the local mobile device, wireless network, and cloud computing. We calculate the execution time of local as well as a remote cluster of tasks based on equation (3) and equation (4), respectively. The task priority is identical to HEFT algorithm, if a task v_i remote cluster task the average computation time is:

$$w_i = T_i^c + T_i^s + T_i^r, \tag{27}$$

similar the average computation of a task v_i when schedule locally:

$$w_i = \underset{1<k<K}{AVG} T_i^{le}, \tag{28}$$

the priority shows the task precedence requirements:

$$Priority(v_i) = w_i + \max_{v_j \in succ(v_i)} Priority(v_j) \tag{29}$$

$$Priority(v_i) = w_i, v_i \in exist - task, \tag{30}$$

equation (29) and equation (30) show that the priority level of all tasks recursively computed from the starting point of the task graph until exist tasks [25].

4.3. Fault Aware Phase

Fault awareness phase is very important and an essential component of mob-cloud, which significantly embedded between client and server. Since, Fig. 4 demonstrates two scenarios, for instance stable scenario and unstable scenario. Whereas in unstable scenario, it starts from the task offloading phase, which transmits computation and data of a task to the cloud server via wireless network. Nonetheless, data transmission and receiving their results from the cloud can be interrupted by connection failure. In another case, data transmission would be interrupted by cloud node failure. However, in stable phase node failure occur while data transmission, processing in the cloud, and receiving their results to the mobile. To deal with the above mentioned scenarios mob-cloud has an efficient fault aware Algorithm, which employs two preliminary strategies for node failure and connection failure. For instance, the Dynamic Reclustering (DR) strategy and Failover Mechanisms for Distributed SDN Controllers (FMDSC) as shown in Algorithm 3. Which determines the failed tasks either from a local cluster or a remote cluster along with tuple information (e.g., TaskID, LocationID, Resource ID, failure point, the result obtained), and failed task due to the wireless connection. Algorithm 3 effectively finds failure detection, and sends their information to the failure manager, and then failure manager passes this information to the failure recovery module which implemented by a dynamic Reclustering method and

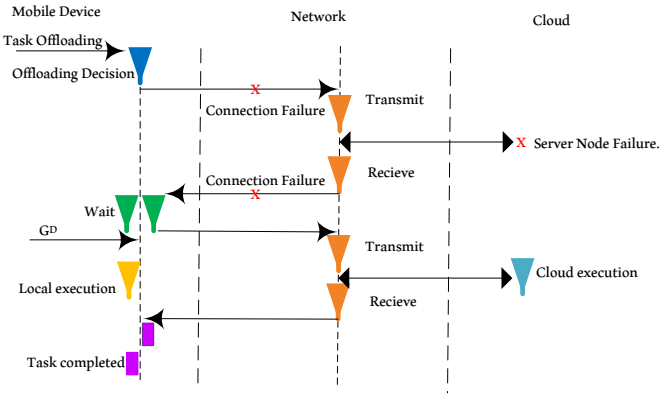

Figure 4. Failure Recovery Strategy

a FMDSC method to fix failure issue. Once, these failures are fixed offloading decision would drive these information to the task scheduler for re-scheduling if still application deadline not missed.

Failure Detection Strategy. In Algorithm 3, Line 2 shows a detection function, which involves (NF and RF module). Lines 2-7 illustrate initializations. Lines 8-13 evaluate that checkpointing technique for all entire nodes that handles data transmit and receiving from the cloud. Whereas, lines 14-19 if a failure detects on a node and still would be recovered based dynamic re-clustering technique within the application deadline, then it fetch checkpointing information from a failure manager other it re-schedule a failed task from scratch. Line 20-23 will fix network failure task based on FMDSC technique.

$$RTT = \sum_{i=1}^{V_c} T_i^s + T_i^{ce} + T_i^r, \tag{31}$$

4.4. Mobility Model Phase

This study formulates a mobility model phase based on Markov random waypoint paradigm, where the each waypoint works as a state, and each state has a set of parameters (e.g., offloading results and decision action), transition function is used to select the next waypoint which may be possibly depend on the recent state location and result. After all waypoints (states) of the entire network is shown via a visibility graph. It is remarkable that the study only consider finite states (sequence) for mobility model. There are some factors like pause time and transitory velocity need to adjust while end-user roaming in a given network range environments.

$$\pi : P \times A \leftarrow [0, 1] \tag{32}$$

$$\pi(a \mid s) = P(a_t = a \mid s_t = s), \tag{33}$$

Algorithm 3: Fault Aware Algorithm

```
1  begin
2      Detection (node, network);
3      state[] ← connected;
4      NF[] ← initialized;
5      RTT[] ← intialized;
6      RF[] ← initialized;
7      T_{i,k}^{avail} ←0;
8      for (checkpointing i to n) do
9          if (NF[i]==true) then
10             for (node i in NF[]) do
11                 RF ← true;
12             RTT[i] = NF[i] + 1;
13         return RTT[i];
14     for (node i in RF[]) do
15         if (RTT[i] > G_D)& (V_c > ε_c) then
16             state[i] ← Failure;
17             RTT[] ← 0;
18         updateRF[i];
19         Dynamic − reclustering(state[i], NF[i])
20     for (node i ∈ to state[]==connected) do
21         if (NF[i]==false) then
22             FMDSC(RF[i], NF[i])
23         Termination;
24     End;
```

equation (19) and equation (20) show map policy provides the probability of acquiring the action a in specific state s. Since, the value function $V_\pi(s)$ described as expected return and starting with s i.e., s_o successively following policy π. It also estimates how good it to be in the given state s, and explain in the following way:

$$V_\pi(s) = E[R] = E[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_o = s], \tag{34}$$

R is a random variable and return expected value and described as a future of discount rewards. Where, $R = \sum_{t=0}^{\infty}$ shows reward r_t at step t and $\gamma \in [0,1]$ is discount rate. The mobility Algorithm 4 typically works with the entire offloading system in order to take the optimal action in a dynamic environment. Algorithm 4 assists Algorithm 1 to take optimal action in any given state in order to maintain application performance during mobility. In Algorithm 4 lines 4-8 initialize visibility graph of entire network, and their variables. Line 9 surf all areas which involve in visibility graph. Line 9 traces user existing movement either it is in moving state or stop for some reason. Lines 13-21, Algorithm detect user location coordinates and movement state it checks round-trip for data uploading/ downloading is under given threshold and exceeds their limits. If until it is under given threshold and transition to the next state, the policy will be updated based on current state results. Lines 22-26 illustrates that, if an end-user still in same state and did not transit to next state will be carry in addition to same policy. Whereas, lines 27-29 search another value based on update policy, once next state has better offloading action results and round-trip time as compared to existing state, the transition probability will add reward in result as illustrated in Lines 31-38. The update Q-learning based policy always optimize entire states in a given waypoint visibility graph. Whenever, Q-value is updated by the entire waypoints it calculate optimal t value (e.g) minimum delay between mobile clients and cloud service will be returned as shown by lines 39-42.

4.5. Time Complexity

The complexity of the proposed algorithm MATOA $O(KNlogN)$, where K is the number of WANET devices, whereas N is a number of workflow application tasks. We modified the DHEFT algorithm for task scheduling between mobile and cloud computing in terms of time complexity. For task scheduling, all tasks are organized with some sequences (e.g., apply sorted queue), it gets $O(KNlogN)$ to build task queue. For resource mapping, each iteration uses $O(KN)$ complexity of comparing the head of the queue. Therefore the entire complexity of for local devices is $O(KNlogN)$. Since, mob-cloud has cloudlet DC resources, and public cloud resources, initially they used $O(N)$ to estimate the task execution at cloud servers, thus the total complexity of the cloud is $O(KNlogN + N) = O(KNlogN)$. The decision engine uses $O(1)$ time complexity of clustering the application into local and remote cluster. Hence the entire time complexity of the MATOA for offloading system is $O(KNlogN)$.

5. Performance Evaluation

5.1. Experiments settings

To the best we know, for the performance evaluation existing MCC workflow simulator must be modified, where both computational and network resources should be measured simultaneously. Therefore, we added some additional components such as mobility model, network model and edge component to the mob-cloud workflow simulator. Thus, to evaluate the effectiveness and efficiency of MATOA as compared to existing offloading schemes on different kinds of mobile applications, we carry out 4 android workflow applications as shown in Table 4. Furthermore, MCC setup parameters are listed in Table 3. We designed the simulator mob-cloud for the considered problem which

Table 3. Simulation Parameters

Simulation Parameters	Values
λ_i user arrival time	5 seconds
Languages	JAVA, XML, Python
Application fine-grained	Methods
Simulation Time	6 hours
Experiment Repetition	14
Location user Mobility	Waypoint model
Standard task size	1500 to 2000 MI
Upload/download data size	2000/150 KB
Ad-hoc-devices	HTC G17 and Samsung 1997
Cloudlet	Intel 5 laptop, AndroidX86
Public Cloud	AndroidX86 Amazon t2.medium

Table 4. Practical Application Scenarios

No	Applications	Scenario	User Preference
I	Healthcare, AG	Stable Env.	Time Sensitive
II	Business, 3D-Gaming	Stable Env.	Time Sensitive
III	Healthcare, AG	Unstable Env.	Time Sensitive
IV	Business, 3D-Gaming	Unstable Env.	Time Sensitive

Table 5. Workload Analysis

Workload	data size(byte)	C.Ins. (MI)	No. of tasks
Health, AG	825	5.8	200~500
Business, Game	735	7.8	500~1000

Table 6. Resource Availability Cases

Case	Cloud VM	Cloudlet	Devices
I	4	2	2~4
II	0	2	2~4
III	2	0	2~4
IV	0	0	2~6

Table 7. Cases for Network Speed

Case	Wifi(MB/s VM	3G(MB/s)	Bluetooth(MB/s)
I	14.3	4	2~3
II	0	3.5	2~3.3
III	0	0	2~2.2
IV	1.7	3.5	2.5~3

consists of three offloading scenarios, such as a device to device (e.g., No Offloading) scenario, and single-tier mobile to cloudlet offloading (i.e., Partial Offloading) scenario, and direct cloud offloading (Full Offloading). We exploited all scenarios as baseline approaches when compared them with the proposed architecture mob-cloud. Based on the simulation parameters as illustrated in Table 3, we divided the application into distributed tasks among heterogeneous networks. The GenyMotion android client is exploited as a user interface for testing purpose in the mobile device, and GenyMotion as the server-side for the cloud services

[26]. The existing simulators, for instances cloudsim simulator [27] device to device offloading simulator[28], and cloudlet simulator [29] devised different modeling models. The mob-cloud composed of the heterogeneous cloud environment and exploited GenyMotion with different network models. However, mob-cloud is integrated mobility, and fault aware features in order to adapt any environment changes during application execution.

5.2. Baseline Approaches

We will evaluate the efficiency and effectiveness of MATOA algorithm by comparing it with existing task offloading strategies from a different perspective, such as task offloading, task scheduling, fault awareness, and mobility scenarios for workflow applications. The following existing approaches are under consideration in comparison.

- Full offloading (FUL): It is a task offloading strategy, adopted by many papers, such as [10–12], the primary goal is to offload entire mobile application to the cloud server.
- NO offloading (NOF): This strategy allows offload the application tasks among a set of mobile devices when there are lightweight and required less network requirement.
- Partial Offloading (PAR): It is a widely employed strategy, adopted by many papers, likewise ThinkAir [6], MAUI [8], and JADE [9]. The main goal is to be partitioned the application between the mobile device and cloud computing. However, above-mentioned papers based on PAR strategy proposed their efficient and unique framework to optimize user preferences (e.g., mobile energy consumption, application response time). The paper and frameworks detailed you can find in their respective references.

5.3. Parameter and Components Calibration

In the MATOA has four preliminary components, namely task offloading, task scheduling, failure manager, and mobility for each workflow application. At the same time, in order to verify the performance of the algorithm at different deadline, the performance of the MATOA and the state-of-art algorithms (local offloading techniques) are evaluated under the deadlines from strict to lose. The RPD (Relative Percentage Deviation) is employed to evaluate the performance of the algorithm, the calculation of RPD is defined as follows:

$$RPD\% = \frac{local - T^{total}}{T^{total}} \times 100\%, \qquad (35)$$

we compare $RPD\%$ based on existing local offloading with our proposed partial offloading MATOA.

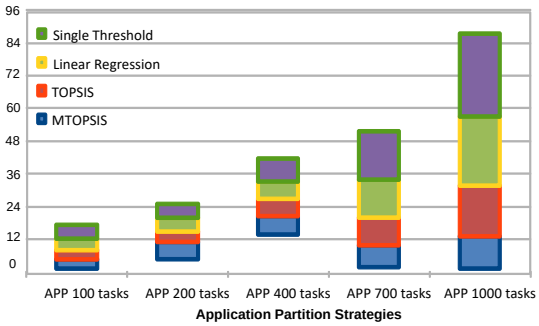

Figure 5. Application Partitioning Accuracy

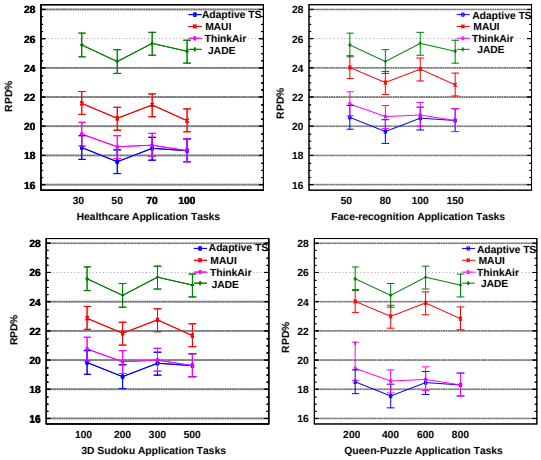

Figure 6. Application Response Time (ms)

5.4. Adaptive Task Offloading Phase

We analyze different application performance in two real scenarios such as stable environment where both network and cloud resource always remain stable. Another scenario where both network and cloud resources values change due to user mobility and cannot remain stable. To evaluate the performance of phase (i), adaptive task offloading strategy uses a MTOPSIS method for application partitioning based on multi-criteria decision engine. Whereas, Fig. 5 shows that the partitioning accuracy during time is an efficient and optimal as compared to the single threshold value based method and others baseline methods. It is noticed in Fig. 6 that the task offloading strategy (adaptive TS) in a dynamic environment for all applications partitioning performance better as compared existing offloading strategies (they assume stable environment remains ever) in terms of response time.

5.5. Algorithm Comparison

Based on the ANOVA technique, Fig. 7 demonstrates that the mean plot of T_{total} by 95.0% Tukey HSD intervals, reduces RPD% of T_{total} by using MATOA algorithm.

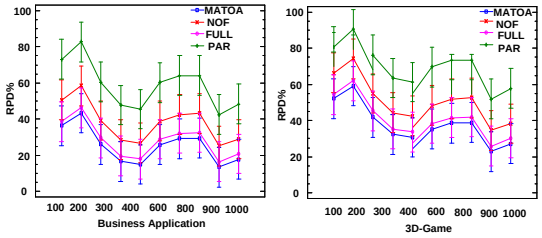

Figure 7. The mean plot of ęÓ with 95.0% Tukey HSD intervals and The mean plot of workflow Application rules with 95.0% Tukey HSD intervals

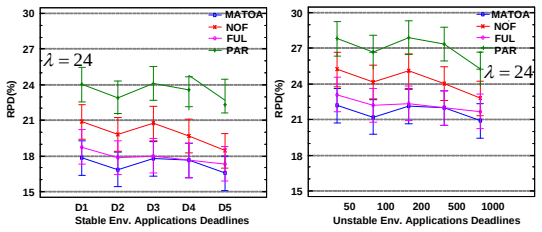

Figure 8. Workload Analysis and Deadlines

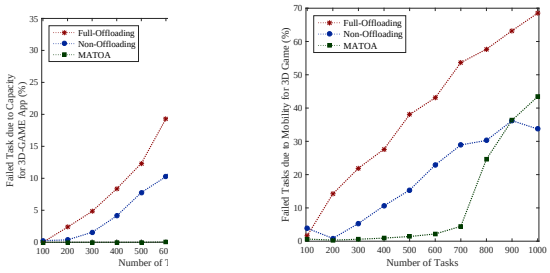

Figure 9. 3D-Game Application performance without and with mobility

5.6. Task Scheduling Phase & Fault Aware Phas

However, user defined deadline constraint for application, Fig. 8 confirms that in both stable unstable environment despite many parameters cha during application offloading and invoking serv proposed algorithm MATOA outperform as comp to baseline approaches. Whereas, the results are sh in the Fig. 9, 10, 11, and 13 for both stable environment without mobility and with unstable environment with mobility, MATOA efficiently adopt dynamic change, and maintain application performance without any comprising as compared to existing approaches. It is noticed that, baseline approaches did not adopt any dynamic changes and degraded application performance during mobility. Table 5 shows workload analysis for different application for execution. To support the mobility and interactivity of an end-user application, we set the multiple cases for network values and cloud resources as shown in Table 6 and Table 7, it can be observed from Fig. 14 in all cases, the

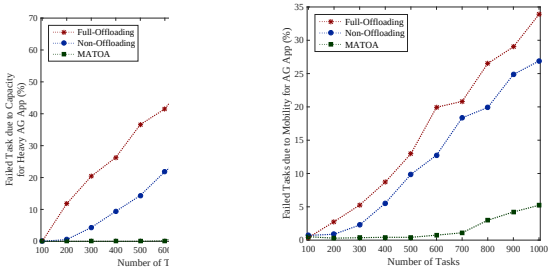

Figure 10. Augmented Application performance without and with mobility

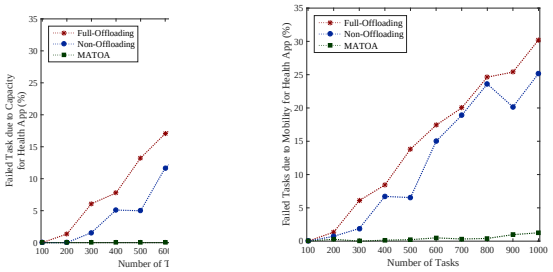

Figure 11. Healthcare Application performance without and with mobility

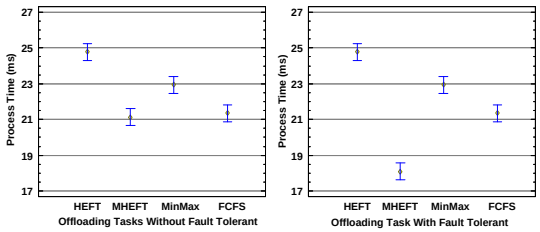

Figure 12. Task Scheduling Heuristics Performance

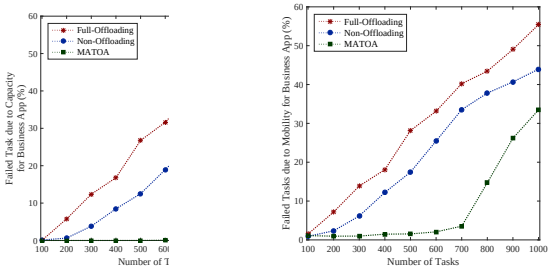

Figure 13. Business Application Performance Without and With Mobility

performance of the MATOA more fairly as compared to literature task offloading approaches. Fig. 12 shows that our modified heuristics are getting lower overhead as compared to existing heuristics, this is because mobile cloud application cannot directly accept the traditional policies.

Table 8. Mobility Dataset

Parameter	Value	View
F (Area of simulated env.)	$17 \times 17Km$	V_G (Googlemap)
V_G (Map-resolution)	1000×1000	r 17m
$v \leftarrow (V_{min}, V_max)$	(3,6,20 Km/h)	speed on foot,car
$p \leftarrow (w_{min}, w_{max})$	(1,10 minutes)	Average wait

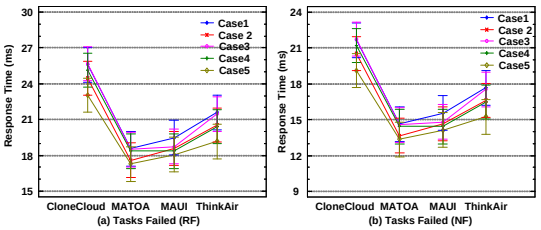

Figure 14. Adaptive Environment with respect to resource and network

5.7. Mobility Model Phase

For mobility scenario, we employ the end-user movement trace mechanism offered by the Every-Where-lab. Since, the trace method offers end-user mobility movement in the given scenario (road network of Milan). The entire road boundary is 17× 17 km. Dataset provides 100,000 end-users in the one waypoint for invoking the cloud services based on adaptive task offloading strategy, whereas, location coordinates of each end-user are traced for every 25 seconds. We already mentioned that, during mobility the whole network area coverage by base-stations (BSs), whereas, each BS is linked with edge cloudlet DC as shown in Figure 4. It is a notable public cloud performance become very poor during mobility due to longer delay. Typically, the mobility feature performs well with edge cloudlet DC and WANET. Each workflow application has different properties and task characterization. We measure the application execution performance without mobility and with mobility features in context of task failure. Since, Figure 13, 14, and 16 illustrate that application tasks failed during mobility and without mobility environment, however, MATOA outperforms as compared to existing approaches in context task failure. MATOA has less failure ratio, while applications are offloading or invoking services while roaming between networks. The mobility movement dataset has many parameters and values since, we have shown a few values in Table 8 for understanding the scenario, however you can find the full detail in [30].

6. CONCLUSION and Future Work

This work studies the dynamic content aware task offloading problem in heterogeneous mobile cloud environments. To the best knowledge, failure aware task offloading in an adaptive environment is useful for real life application. We modified existing MCC architecture with additional components that one may support the user mobility, fault tolerant and task scheduling mechanism for a workflow application. We proposed a novel Mobility Aware Task Offloading Algorithm (MATOA) that minimizes the average response time of the workflow application and delay sensitive application under task precedence constraint and hard constraint deadline. MATOA has three phases: first, it performs the task offloading mechanism based on multi-criteria as mentioned above, second task scheduling based on modified DHEFT heuristic in order to schedule local and remote cluster tasks on appropriate resources, third failure mechanism in order to support mobility and interactivity features of the end user in a dynamic environment.

For future work, we will develop a complex task offloading algorithm and mob-cloud architecture, because existing modified mob-cloud provides cloud services as infrastructure as a service which are wrapped in a virtual machine box. However, virtual machine based services are heavyweight and need pre-allocation of cloud resources (e.g., RAM, storage, libraries, host OS) in advance, but might be requested tasks of an application would use a small amount of resources, thus remaining resources are wastage. Most of the IoT devices invoking microservices to dealing their tasks, however, single code virtual machine services are difficult to recover or redeploy one they failed due to the any reason. In future work, we will design mob-cloud based on container microservices in order to support IoT devices data in a more optimal way and augment the resource utilization of the cloud server. With the development of advance wireless technology and cloud computing mobility as a service, function as a service, and offloading as a service will effective architecture for smart devices, it is difficult to directly deploy complex workflow application into smart devices, in the future we will extend mob-cloud and add fog nodes for preprocessing the IoT on the edge nodes before offloading to remote cloud. There

are many questions need to be addressed, such mobile cloud architecture in the context of Big Services, Service Composition, and Big Data.

References

[1] M.-H. Chen, M. Dong, and B. Liang, "Resource sharing of a computing access point for multi-user mobile cloud offloading with delay constraints," *IEEE Transactions on Mobile Computing*, 2018.

[2] W. Chen, D. Wang, and K. Li, "Multi-user multi-task computation offloading in green mobile edge cloud computing," *IEEE Transactions on Services Computing*, 2018.

[3] H.-S. Lee and J.-W. Lee, "Task offloading in heterogeneous mobile cloud computing: Modeling, analysis, and cloudlet deployment," *IEEE Access*, 2018.

[4] R. Yu, J. Ding, S. Maharjan, S. Gjessing, Y. Zhang, and D. H. Tsang, "Decentralized and optimal resource cooperation in geo-distributed mobile cloud computing," *IEEE Transactions on Emerging Topics in Computing*, vol. 6, no. 1, pp. 72–84, 2018.

[5] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: elastic execution between mobile device and cloud," in *Proceedings of the sixth conference on Computer systems*. ACM, 2011, pp. 301–314.

[6] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, "Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading," in *Infocom, 2012 Proceedings IEEE*. IEEE, 2012, pp. 945–953.

[7] R. Kemp, N. Palmer, T. Kielmann, and H. Bal, "Cuckoo: a computation offloading framework for smartphones," in *International Conference on Mobile Computing, Applications, and Services*. Springer, 2010, pp. 59–79.

[8] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "Maui: making smartphones last longer with code offload," in *Proceedings of the 8th international conference on Mobile systems, applications, and services*. ACM, 2010, pp. 49–62.

[9] H. Qian and D. Andresen, "Jade: An efficient energy-aware computation offloading system with heterogeneous network interface bonding for ad-hoc networked mobile devices," in *Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), 2014 15th IEEE/ACIS International Conference on*. IEEE, 2014, pp. 1–8.

[10] M. Satyanarayanan, Z. Chen, K. Ha, W. Hu, W. Richter, and P. Pillai, "Cloudlets: at the leading edge of mobile-cloud convergence," in *2014 6th International Conference on Mobile Computing, Applications and Services (MobiCASE)*. IEEE, 2014, pp. 1–9.

[11] M. Chen, Y. Hao, C.-F. Lai, D. Wu, Y. Li, and K. Hwang, "Opportunistic task scheduling over co-located clouds in mobile environment," *IEEE Transactions on Services Computing*, vol. 11, no. 3, pp. 549–561, 2018.

[12] 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.

[13] B. Zhou, A. V. Dastjerdi, R. N. Calheiros, and R. Buyya, "An online algorithm for task offloading in heterogeneous mobile clouds," *ACM Transactions on Internet Technology (TOIT)*, vol. 18, no. 2, p. 23, 2018.

[14] C.-K. Tham and B. Cao, "Stochastic programming methods for workload assignment in an ad hoc mobile cloud," *IEEE Transactions on Mobile Computing*, vol. 17, no. 7, pp. 1709–1722, 2018.

[15] Z. Kuang, S. Guo, J. Liu, and Y. Yang, "A quick-response framework for multi-user computation offloading in mobile cloud computing," *Future Generation Computer Systems*, vol. 81, pp. 166–176, 2018.

[16] G. Zhang, W. Zhang, Y. Cao, D. Li, and L. Wang, "Energy-delay tradeoff for dynamic offloading in mobile-edge computing system with energy harvesting devices," *IEEE Transactions on Industrial Informatics*, 2018.

[17] Y.-H. Kao, B. Krishnamachari, M.-R. Ra, and F. Bai, "Hermes: Latency optimal task assignment for resource-constrained mobile computing," *IEEE Transactions on Mobile Computing*, vol. 16, no. 11, pp. 3056–3069, 2017.

[18] H. Wu, W. Knottenbelt, K. Wolter, and Y. Sun, "An optimal offloading partitioning algorithm in mobile cloud computing," in *International Conference on Quantitative Evaluation of Systems*. Springer, 2016, pp. 311–328.

[19] B. Zhou, A. V. Dastjerdi, R. N. Calheiros, S. N. Srirama, and R. Buyya, "mcloud: A context-aware offloading framework for heterogeneous mobile cloud," *IEEE Transactions on Services Computing*, vol. 10, no. 5, pp. 797–810, 2017.

[20] T. G. Rodrigues, K. Suto, H. Nishiyama, and N. Kato, "Hybrid method for minimizing service delay in edge cloud computing through vm migration and transmission power control," *IEEE Transactions on Computers*, vol. 66, no. 5, pp. 810–819, 2017.

[21] R. Yu, J. Ding, S. Maharjan, S. Gjessing, Y. Zhang, and D. H. Tsang, "Decentralized and optimal resource cooperation in geo-distributed mobile cloud computing," *IEEE Transactions on Emerging Topics in Computing*, vol. 6, no. 1, pp. 72–84, 2018.

[22] B. Zhou and R. Buyya, "Augmentation techniques for mobile cloud computing: A taxonomy, survey, and future directions," *ACM Computing Surveys (CSUR)*, vol. 51, no. 1, p. 13, 2018.

[23] G. Baudry, C. Macharis, and T. Vallée, "Range-based multi-actor multi-criteria analysis: A combined method of multi-actor multi-criteria analysis and monte carlo simulation to support participatory decision making under uncertainty," *European Journal of Operational Research*, vol. 264, no. 1, pp. 257–269, 2018.

[24] Z. Zhou, Z. Cheng, L.-J. Zhang, W. Gaaloul, and K. Ning, "Scientific workflow clustering and recommendation leveraging layer hierarchical analysis," *IEEE Transactions on Services Computing*, vol. 11, no. 1, pp. 169–183, 2018.

[25] N. Chopra and S. Singh, "Heft based workflow scheduling algorithm for cost optimization within deadline in hybrid clouds," in *Computing, Communications and Networking Technologies (ICCCNT), 2013 Fourth International Conference on*. IEEE, 2013, pp. 1–6.

[26] J. ECHESSA, "Improved android emulation with genymotion."
[27] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya, "Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and experience*, vol. 41, no. 1, pp. 23–50, 2011.
[28] G. Nardini, A. Virdis, and G. Stea, "Simulating device-to-device communications in omnet++ with simulte: scenarios and configurations," *arXiv preprint arXiv:1609.05173*, 2016.
[29] K. Ha and M. Satyanarayanan, "Openstack++ for cloudlet deployment," *School of Computer Science Carnegie Mellon University Pittsburgh*, 2015.
[30] "User movement simulations project, accessed on apr. 6, 2017. [online]. available: http://everywarelab.di.unimi.it/lbs-datasim."

Algorithm 4: mobility model

Input : t=(D, F, RTT), duration d;
Output: min t with waypoint for each state;

begin
  $V_G \leftarrow$compute visibility graph with RTT and area F;
  $t \leftarrow 0$ minimum delay;
  $S_n[] \leftarrow 0$ initial state;
  $v \leftarrow 0$ velocity;
  $p \leftarrow 0$ pause time;
  $R[] \leftarrow 0$ reward;
  $a[] \leftarrow 0$ action;
  foreach $(r \in F)$ **do**
    while $(r <\mid N_r^{move} \mid + \mid N_r^{stop} \mid)$ **do**
      $r[] \leftarrow n$;
      if $r[] <\mid N_r^{move} \mid$ **then**
        $n \leftarrow\mid N_r^{move} \mid$;
        calculate v, p, and RTT based on equation (33), (34);
        $v \leftarrow rand(V_{min}, V_{max})$;
        $p \leftarrow rand(V_{min}, V_{max})$;
        $W_n[] \leftarrow v + RTT + p$;
        if $RTT < W_n[]$ **then**
          $R[]$++;
          $\pi =$
          $\pi(a[] \mid s[]) = P([]a_t = a[] \mid s_t = s)$;
          update $\pi[] \leftarrow RTT$;
      else
        $n \leftarrow\mid N_r^{stop} \mid$;
        $R[]$–;
        $\pi = \pi(a[] \mid s[]) = P([]a_t = a[] \mid s_t = s)$;
        carry same policy $\pi[] \leftarrow RTT$;
    while $t < d$ **do**
      if $n \in N_r^{stop}$ **then**
        $W_n \leftarrow$ path based on movement cycle Z_r and VE using Modified value function;
      else
        $W_n[] \leftarrow$ get random position inside P_r;
      if $RTT < W_n[]$ **then**
        $R[]$++;
        $\pi = \pi(a[] \mid s[]) = P([]a_t = a[] \mid s_t = s)$;
        update $\pi[] \leftarrow RTT$;
        $v \leftarrow rand(V_{min}, V_{max})$;
        $p \leftarrow rand(V_{min}, V_{max})$;
        add waypoint for $W_n[]$ to the trace based on V and P;
        $Q^*(s[], a[]) \leftarrow t + W_n[]$
        employed generate modified-q-value;
      $t \leftarrow Q^*(s[], a[])$;
  return t;