MULTI-TASK COMPUTATION OFFLOADING AND SCHEDULING IN MOBILE EDGE COMPUTING SYSTEMS

A PROJECT REPORT

Submitted by

B.BALAJI 2015115009

M.MOHAMMED HASIF 2015115029

K.SARGURU 2015115049

submitted to the Faculty of

INFORMATION AND COMMUNICATION ENGINEERING

in partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY

in

INFORMATION TECHNOLOGY

DEPARTMENT OF INFORMATION SCIENCE AND TECHNOLOGY

COLLEGE OF ENGINEERING, GUINDY

ANNA UNIVERSITY

CHENNAI 600 025

APRIL 2019

ANNA UNIVERSITY

CHENNAI - 600 025

BONAFIDE CERTIFICATE

Certified that this project report titled MULTI-TASK COMPUTATION OFFLOADING AND SCHEDULING IN MOBILE EDGE COMPUTING SYSTEMS is the bonafide work of B. BALAJI (2015115009), M. MOHAMMED HASIF (2015115029) and K. SARGURU (2015115049) who carried out project work under my supervision. Certified further that to the best of my knowledge and belief, the work reported herein does not form part of any other thesis or dissertation on the basis of which a degree or an award was conferred on an earlier occasion on this or any other candidate.

PLACE: CHENNAI

DATE:

Dr. M. VIJAYALAKSHMI
ASSOCIATE PROFESSOR
DEPARTMENT OF IST, CEG
ANNA UNIVERSITY
CHENNAI 600025

COUNTERSIGNED

Dr. SASWATI MUKHERJEE
HEAD OF THE DEPARTMENT
DEPARTMENT OF INFORMATION SCIENCE AND TECHNOLOGY
COLLEGE OF ENGINEERING, GUINDY
ANNA UNIVERSITY
CHENNAI 600025

திட்டப்பணிச் சுருக்கம்

கணக்கீட்டு ஆஃப்லோடிங் என்பது கணினி முறைமைகளை அதிகப்படுத்துவதற்கான ஒரு தீர்வாகும், மேலும் மேலதிக கணினிகள் (அதாவது, சர்வர்கள்) கணக்கீடுகளை நகர்த்துவதன் மூலம். பிணைய அலைவரிசை, சேவையக வேகம், விளிம்பின் வேகம், கிடைக்கும் நினைவகம், சேவையக சுமை மற்றும் மொபைல் அமைப்புகள், மொபைல் விளிம்பிற்கும் சேவையகத்திற்கும் இடையில் பரிமாற்றப்பட்ட தரவு அளவு ஆகியவற்றைப் பொறுத்து, கணக்கீட்டு ஆஃப்லோடிங், குறிப்பிட்ட கணினி மேடை அல்லது ஒரு கிளவுட் போன்ற வெளிப்புற தளத்திற்கு சில கணக்கீட்டு பணிகளை மாற்றுவதை குறிக்கிறது. மொபைல் எட்ஜ் கிளவுட் கம்ப்யூட்டிங் (எம்.சி.சி.சி), மொபைல் சாதனங்கள் (எம்.டி.எஸ்) கம்ப்யூட்டிங் மற்றும் ஸ்டேன்ஜிங் திறனை நெட்வொர்க் விளிம்பில் பயன்படுத்தக்கூடிய வளங்களை சுரண்டுவதன் மூலம் ஒரு விளிம்பு முனையிலோ அல்லது தொலை தூரத்திலோ சர்வர்.

இந்த செயல்திட்ட வேலை மொபைல் லேட் கிளவுட் கம்ப்யூட்டிங் அமைப்பை உருவாக்கி உள்ளூர் மங்கல்கள் மற்றும் இரண்டு மடிக்கணினிகளை விளிம்பு முனையங்களாகவும், டிஜிட்டல் பெருங்கலை கிளவுட் சேவையகமாகப் பயன்படுத்துவதற்கும் இரண்டு மடிக்கணினிகளை ஒருங்கிணைத்து வழங்குகிறது. உள்ளூர் பக்கத்தில் உள்ள ஆஃப்லோடிங் கட்டமைப்பானது விவரக்குறிப்பான் தொகுதியைக் கொண்டுள்ளது, இதில் சாதன கூறுபவர், பயன்பாட்டு பேஸ்புக் மற்றும் நெட்வொர்க் பேராசிரியர் ஆகியோர் அடங்கும். சாதன விவரக்குறிப்பானது பேட்டரி நிலை, CPU கோர்கள், CPU கோர் பயன்பாடு மற்றும் CPU அதிர்வெண் பற்றிய விவரங்களை வழங்குகிறது. ஒரு பிணைய விவரக்குறிப்பானது வயர்லெஸ் இணைப்பு நிலை மற்றும் கிடைக்கக்கூடிய அலைவரிசையைப் பற்றிய தகவல்களை சேகரிக்கிறது. செயல்பாட்டு நேரம், ஹிட் விகிதம், முதலியன போன்ற பயன்பாடுகளின் சிறப்பியல்புகளை ஒரு நிரல் நிபுணர் சேகரிக்கிறார். முறை சார்ந்த பயன்பாடு பகிர்வுகளில், பயன்பாடு தனி பணிகளாக பிரிக்கப்படுகிறது. அந்த முடிவுகளை உள்ளூர் கணு அல்லது விளிம்பு முனையிலோ அல்லது கிளையன் சேவையகத்திலோ, பயனாளர்களின் அளவுருக்கள் அடிப்படையில் தீர்மானிக்க முடிகிறது. பல உள்ளூர் பணிகளை விளிம்பு முனையிலிருந்து வரும் பல பணிகளை திட்டமிட ஒரு விநியோகிக்கப்பட்ட பேராசை அதிகபட்ச திட்டமிடல் பயன்படுத்தப்படுகிறது.

ABSTRACT

Computational Offloading is a solution to augment system's capabilities by migrating computations to more resourceful computers (i.e.,servers). Offloading decision depends on many parameters such as the network bandwidth, server speed, edge speed, available memory, server load and the amount of data exchanged between mobile systems, mobile edge and server. Computational offloading refers to the transfer of certain computation tasks to an external platform, such as a mobile edge or a Cloud. Mobile Edge Cloud Computing (MECC) has becoming an attractive solution for augmenting the computing and storage capacity of Mobile Devices (MDs) by exploiting the available resources at the network edge thereby eliminating the intensive tasks to be offloaded to a edge node or a remote cloud server.

This project work presents the formation of mobile edge cloud computing setup by integrating two laptops as local nodes and two laptops as edge nodes and using Digital Ocean as Cloud server. The offloading framework in the local side contains profiler module which consists of three components namely device profiler, application profiler and network profiler which are necessary in making decision for offloading. Device profiler gives the details about battery level, number of CPU cores, CPU core utilisation and CPU frequency. A network profiler collects information about wireless connection status and available bandwidth. A program profiler collects characteristics of applications such as execution time, hit ratio, etc. In method based application partitioning, the application is splitted into separate tasks. The Decision Maker decides the location of execution of those tasks either in local node or in edge node or in Cloud server based on the parameters of profilers. A Distributed Greedy Maximal scheduling is used to schedule the multiple tasks coming from multiple local nodes to the edge nodes.

ACKNOWLEDGEMENT

We wish to record our deep sense of gratitude and profound thanks to our project supervisor **Dr. M. Vijayalakshmi**, Associate Professor, Department of Information Science and Technology, College of Engineering, Guindy for her keen interest, inspiring guidance, constant encouragement with our work during all stages, to bring this thesis into fruition.

We are extremely indebted to **Dr. Saswati Mukherjee**, Professor and Head of the Department of Information Science and Technology, Anna University, Chennai, for extending the facilities of the Department towards our project and for her unstinting support.

We express our sincere thanks to the review committee panel members **Dr. K. Vani**, Professor, **Dr. P. Yogesh**, Professor, **Dr. S. Bama**, Assistant Professor, **Dr. N. Thangaraj**, Assistant Professor and **Mr. B. Yuvaraj**, Teaching Fellow for their critical reviews throughout the course of our project.

We thank our parents, family, and friends for bearing with us throughout the course of our project and for the opportunity they provided us in undergoing this course in such a prestigious institution.

BALAJI B

MOHAMMED HASIF M

SARGURU K

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

API	Application Programming Interface
CPU	Central Processing Unit
DGMS	Distributed Greedy Maximal Scheduling
EH	Energy Harvesting
IT	Information Technology
MECC	Mobile Edge Cloud Computing
MD	Mobile Device
MEC	Mobile Edge Computing
MU	Mobile User
QoE	Quality of Experience
QoS	Quality of Service
RAN	Radio Access Network
RPC	Remote Procedure Call
SDR	Semi Definite Relaxation
WD	Wireless Device
WiFi	Wireless Fidelity

Chapter 1

INTRODUCTION

1.1 MOBILE EDGE COMPUTING

Mobile edge computing, is a network architecture concept that enables cloud computing capabilities and an IT service environment at the edge of the cellular network [1] and more in general at the edge of any network. The basic idea behind MEC is that by running applications and performing related processing tasks closer to the cellular customer, network congestion is reduced and applications perform better. MEC technology is designed to be implemented at the cellular base stations or other edge nodes, and enables flexible and rapid deployment of new applications and services for customers. Combining elements of information technology and telecommunications networking, MEC also allows cellular operators to open their radio access network (RAN) to authorized third parties, such as application developers and content providers.

1.2 BUSINESS AND TECHNICAL BENEFITS OF MEC

By using mobile edge computing technology, a cellular operator can efficiently deploy new services for specific customers or classes of customers. The technology also reduces the signal load of the core network [2] and can host applications and services in a less costly way. It also collects data about storage, network bandwidth, CPU utilization, etc., for each application or service deployed by a third party. Application developers and content providers can take advantage of close proximity to cellular subscribers and real-time RAN information. MEC has been created using open standards and application programming interfaces (APIs), using common programming models, relevant tool chains and software development kits to encourage and expedite the development of new applications for the new MEC environment.

1.3 NEED FOR MOBILE EDGE COMPUTING

Nowadays, Mobile Devices (MDs) such as smartphones, tablet, computers, wearable devices and etc., are playing more and more important role in our daily life. Moreover, mobile applications running on MDs, such as immersive gaming, human-computer interaction, often demand stringent delay and processing requirements. Consequently, despite the tremendous improvement of MDs' processing capacity, executing applications solely on a MD is still incapable of providing satisfactory Quality of Experience (QoE) to users. Besides, MDs are energy constrained, which further increases the tension between resource limited MDs and computation intensive mobile applications. Fortunately, with the advanced networking and multitask technology, offloading part of the work load of an application (or simply put it as a task) from MDs to a remote cloud or a nearby mobile edge cloud (also known as cloudlet) provides a promising alternative to reduce the task execution time as well as prolong the lifetime of MDs. A mobile edge cloud consists of either one edge server or a set of devices that serve mobile users cooperatively. Though the mobile edge cloud is less powerful compared with a remote cloud, as it is located at the edge of the network the transmission latency between a MD and the mobile edge cloud is much lower than that of the remote cloud. Besides, a mobile edge cloud can explore the idle computing and storage resources at the network edge efficiently.

1.4 FORMATION OF MOBILE EDGE COMPUTING SETUP

In this Project work, Mobile Edge Computation system setup has been formed by integrating two laptops as local nodes and two laptops as edge nodes and the Digital Ocean as cloud server. The setup [3] for Mobile Edge Computation system is depicted in Figure 1.1. The local devices are connected to edge devices through wireless medium and the edge devices are connected to the Cloud server. The end user device is connected to cloud server through edge device.

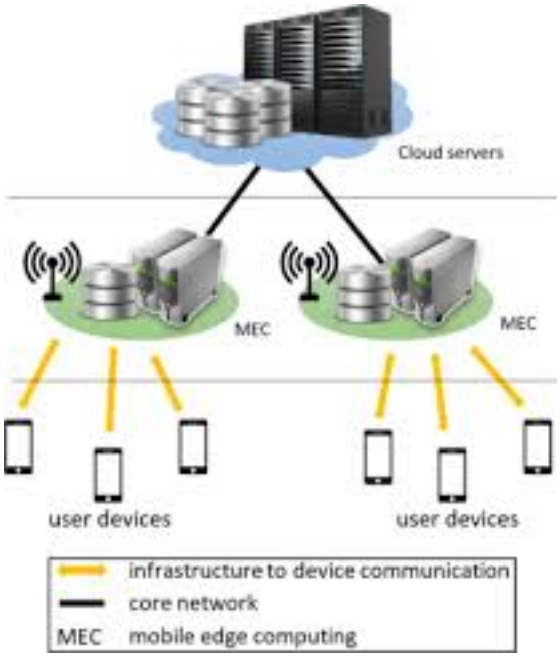

Figure 1.1: Mobile Edge Computing Setup

Any data that is uploaded to cloud from user device will reach the edge node, the edge node will transfer the task to the Cloud server.

1.5 PROBLEM STATEMENT

The limited CPU processing power and battery lifetime of mobile devices hinder the possible execution of computationally intensive applications in mobile devices. Computational offloading is a solution to augment system's capabilities by migrating computation to more resourceful computers.

1.6 OBJECTIVE OF THE PROJECT

The computationally intensive tasks of the Local devices with limited processing capacity and energy constraints are offloaded for computation to the Edge node or remote cloud to reduce the battery usage. The system also explore the idle devices and storage resources at the network edge and perform computation faster.

1.7 SCOPE OF THE PROJECT

With the development of mobile network and 5G, MEC has become a promising field in these years. It not only meets the user's more business needs, improves the QoS and QoE of MU, but also brings business benefits to service providers.

1.8 OVERVIEW OF THE PROPOSED SYSTEM

The proposed system consists of four modules, namely Profiling, Application Partitioning, Offload Decision Making and Scheduling.

In profiling, the program status is captured by the program profiler, the network status such as bandwidth, upload speed and download speed are captured by the network profiler and the device status such as battery level, battery usage are captured by the device profiler. In application partitioning, the application is partitioned into tasks by method based partitioning.

In offload decision making, Random forest algorithm is used to classify the tasks. In Scheduling, Distributed Greedy Maximal algorithm is used to schedule the tasks which are to be offloaded to edge devices based on edge device's CPU utilization and network bandwidth between local device and edge device.

1.9 ORGANIZATION OF THE PROJECT

The project report is organised as follows

Chapter 2 discusses about the literature survey done for this project. This chapter discusses in detail the various literatures studied for understanding the problem domain clearly.

Chapter 3 describes the complete design of the project. It reflects how the

problem statement is addressed in a real time environment. Modules of the project have been discussed in detail in this chapter.

Chapter 4 explains the complete implementation details of the project. It discusses about comments about the results obtained from the project. It discusses about the outcome of every module of the project with necessary screen shots.

Chapter 5 discusses about the conclusion and the future enhancements that can be done.

Chapter 2

LITERATURE SURVEY

2.1 MULTIUSER COMPUTATION OFFLOADING

Xu Chen et al. [4] proposed the multiuser computation offloading to compute a centralized optimal solution, and hence he adopted a game theoretic approach for achieving efficient computation offloading in a distributed manner. He formulated the distributed computation offloading decision making problem among mobile device users as a multi-user computation offloading.

Thinh Quang Dinh et al. [5] proposed an optimization framework of offloading from a single mobile device (MD) to multiple edge devices. Both the total tasks execution latency and the MD's energy consumption are minimized by jointly optimizing the task allocation decision and the MD's central process unit (CPU) frequency. This paper considers two cases for the MD, i.e., fixed CPU frequency and elastic CPU frequency. They proposed a linear relaxation-based approach and a semidefinite relaxation (SDR)-based approach for the xed CPU frequency case, and an exhaustive search based approach and an SDR-based approach for the elastic CPU frequency case. The SDR-based algorithms achieve near optimal performance. Performance can be improved in terms of energy consumption and tasks execution latency when multiple edge devices and elastic CPU frequency are considered. And the MD's flexible CPU range can have an impact on the task allocation.

2.2 MULTITASK SCHEDULING

Yibo Yang et al. [6] created an optimization problem to minimize the combination of energy cost and packet congestion. The packet congestion of different priority packets transmitted to MEC are efficiently controlled by adopting

a promoted-by-probability scheme. The total overhead of MEC in terms of energy consumption and queuing congestion are minimized.

Hamed Shah-Mansouri et al. [7] has addressed the issues in selection of tasks to be offloaded to cloud servers and the optimal price of cloud services. The scheduler effectively balances the tradeoff between the energy consumption and delay.

2.3 GREEN MOBILE EDGE CLOUD COMPUTING

Weiwei Chen et al. [8] has discussed about the multi-user multi-task offloading scheduling schemes in a renewable mobile edge cloud system. As the mobile edge cloud is composed with relatively low processing ability (the processing ability is not as high as that of a server), scheduling schemes needs to map the workload from a MD to multiple WDs. Moreover, scheduling schemes also needs to handle uncertain energy supply at each WD properly so as to make the best of the mobile edge cloud system. In detail, the scheduling algorithms determine the energy harvesting strategy, a set of offloading requests to be admitted, and a sub-set of WDs to compute the workload for the admitted offloading request so as to maximize the overall system utility. It is NP-hard to compute a centralized optimal solution, and hence a game theoretic approach achieved efficient computation offloading in a distributed manner.

Guanglin Zhang et al. [9] proposed an online dynamic tasks assignment scheduling to investigate the trade off between energy consumption and execution delay for an MEC system with EH capability. Optimal scheduling about the CPU-cycle frequencies of mobile device and transmit power for data transmission can be obtained. Besides, the dynamic online tasks ofoading strategy is developed to modify the data backlogs of queues. The battery energy level and the trade off between energy consumption and execution delay are stabilised.

2.4 CONCLUSION FROM LITERATURE SURVEY

The above cited papers were helpful in visualizing and understanding the flow of the entire project and also in designing of its main architecture. Inputs and its format to be given for each module along with expected output were identified. The Scheduling algorithm should match the local device's offloading request to the Edge node's utilization capacity. It is observed that faster the computation and lower the delay and better the user experience. Mobile Cloud Systems can be efficiently utilised by reducing energy consumption, exploring the idle devices and storage resources at the network Edge which leads to increasing the revenue based on the offloading of the task to the Edge or Cloud. Distributed Greedy Maximal Scheduling algorithm can be preferred over Centralized Greedy algorithm because centralized Greedy algorithm requires a centralized controller which will incur implementation complexity when the network size grows.

Chapter 3

DESIGN

3.1 ARCHITECTURE DIAGRAM

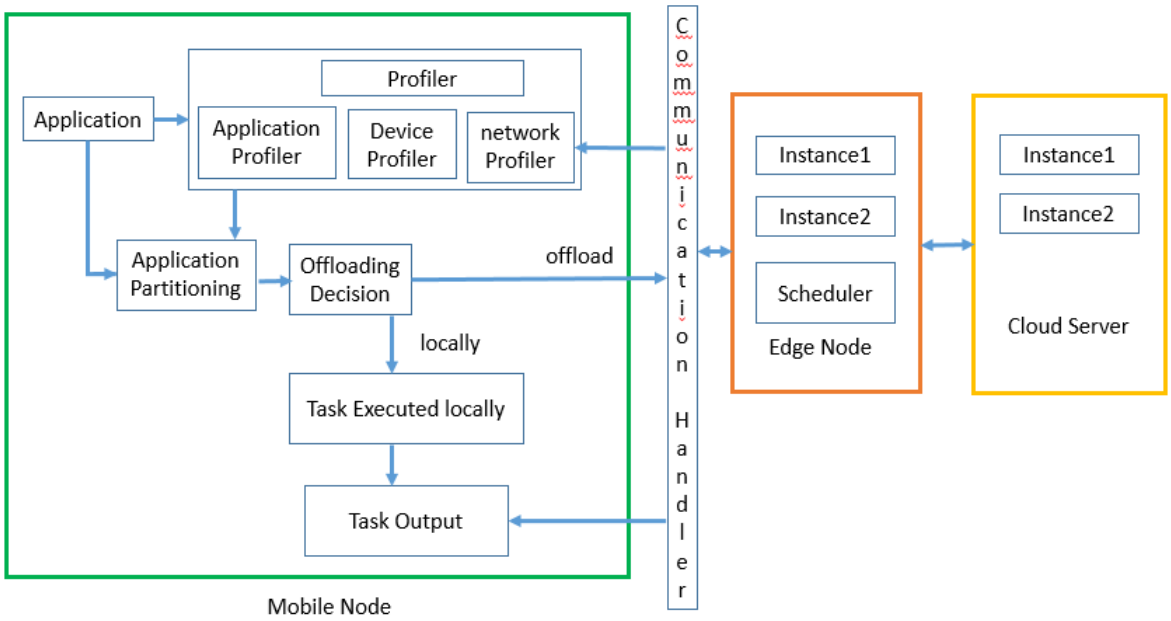

Figure 3.1: Overall Architecture Diagram for Multitask Computation Offloading and Scheduling in MEC

Figure 3.1 shows the overall architecture. It is comprised of profiler, partitioner, decision maker, scheduler, edge node and cloud server. The profilers consists of device profiler, network profiler and program profiler. The profilers send various parameters to the offloading decision maker. After the application is profiled, method based partitioning takes place followed by decision making. Decision maker gives the list of methods to be executed in local device, edge node and cloud server. The methods to be offloaded are sent to the communication handler. The communication handler transfers the method from the local device to the edge node or cloud. The offloaded methods are executed in the edge node or cloud and the output is integrated in the local node.

3.2 OVERALL FLOW DIAGRAM

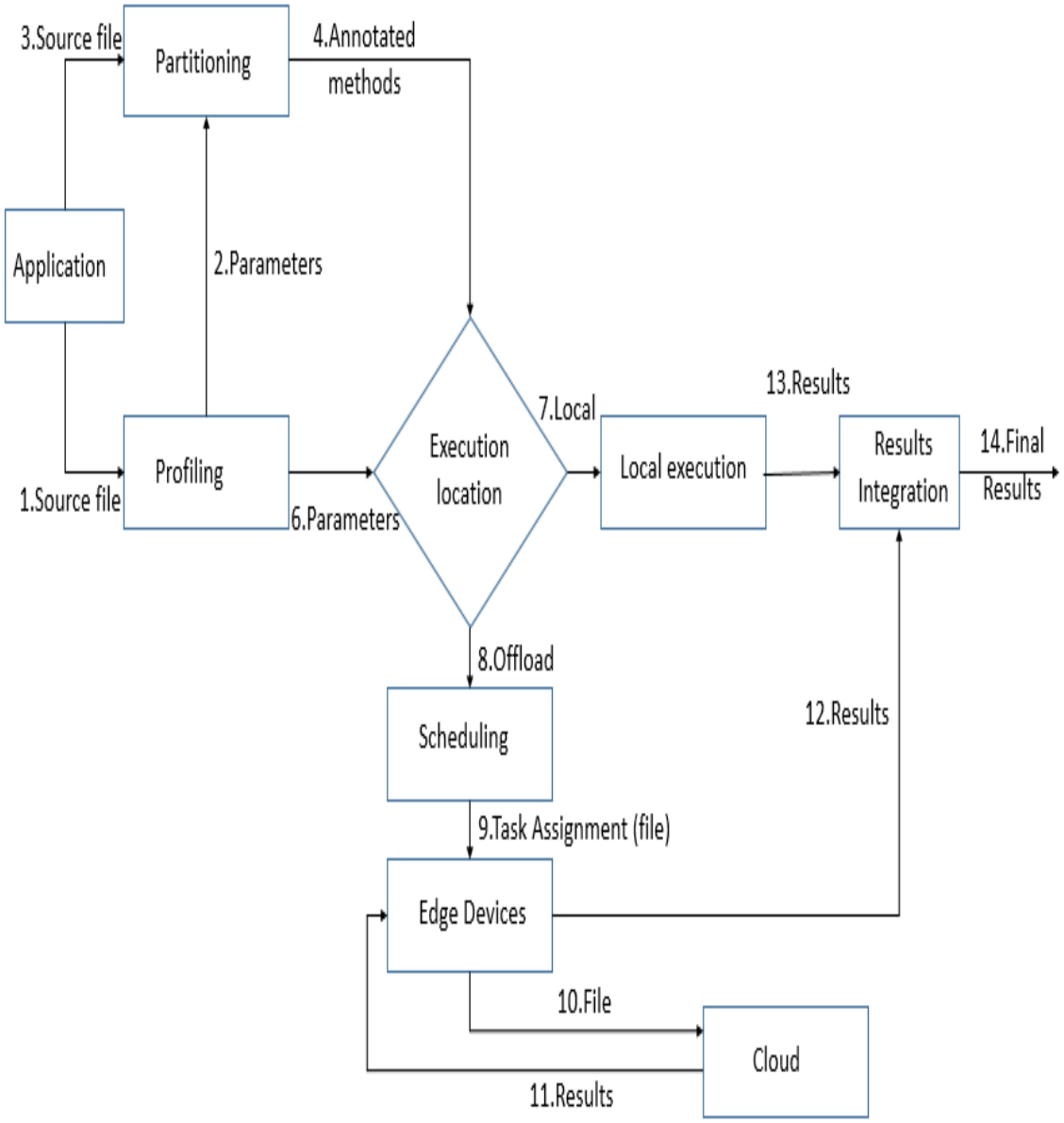

Figure 3.2: Overall Flow Diagram for Multitask Computation Offloading and Scheduling in MEC

Figure 3.2 shows the overall flow representing the various interactions between the modules. Profiling (Step 1) gives program parameters such as execution time, hit rate (step 2) for partitioning(step 3) the application and local device's parameters (step 6) for deciding the location for executing the annotated methods (Step 4). Based on the parameters given by the Profiler, the offloading decision maker decides

the methods which are to be executed locally (Step 7) and the methods which are to be offloaded (Step 8). For Offloading, the methods are transferred by to the Edge devices(step 9) or Cloud server (step 10) and the method execution output is returned to the local device (Step 12). If the execution is local, the method is executed locally (step 13). The execution result of both local and cloud/edge is integrated in Step 14.

3.3 PROFILER

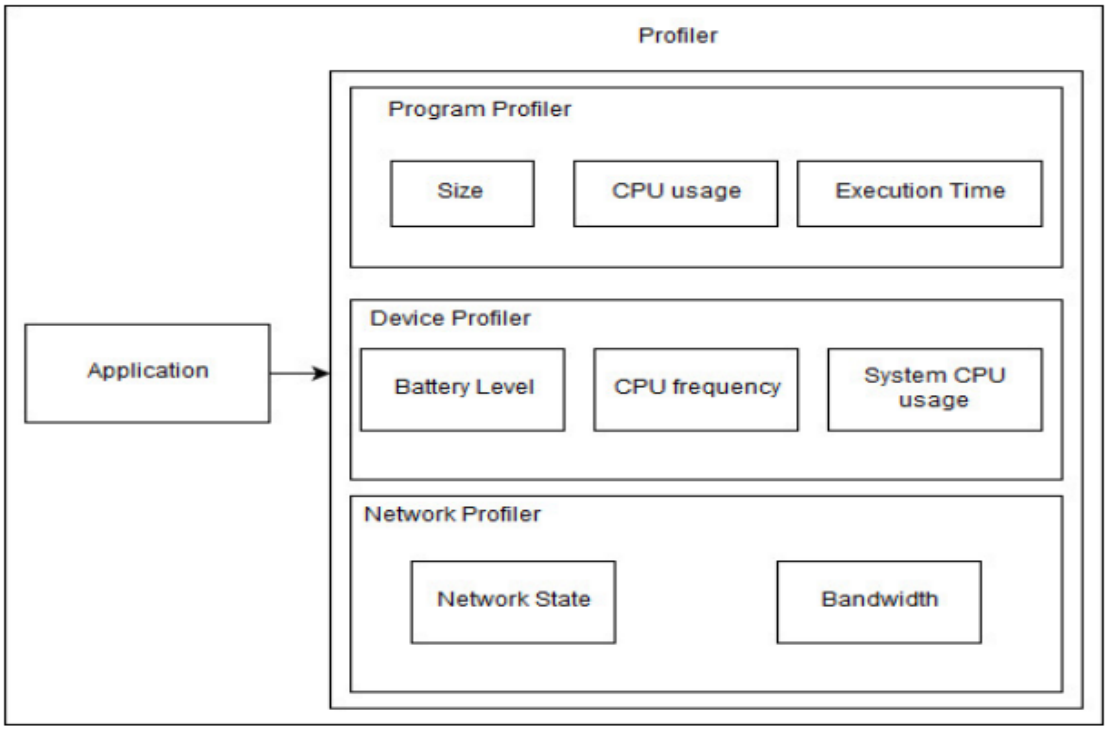

Figure 3.3: Profiler in Mobile Device

Figure 3.3 shows the components of Profiler in Mobile Device. The Profiler consists of three profilers namely Device Profiler, Application Profiler and Network profiler which are necessary in making decision for offloading and scheduling.

3.3.1 Device Profiler

Device profiler contains information about battery level, number of CPU Cores, CPU core utilisation and the CPU frequency. It also tells whether the battery is recharging or not, total number of cores available in local device and Edge node. It also provides the information on average utilisation in Edge node.

3.3.2 Application Profiler

A program profiler collects characteristics of applications, e.g., the execution time, the memory usage and the size of data. It gives information on hit ratio and percentage of time taken for each methods. It also provides the dependency between methods. It provides these information in graph based structure by representing methods as vertices and dependency between methods as edges. The graph structure is formed by importing the package 'pycallGraph'.

3.3.3 Network Profiler

A network profiler collects information about wireless connection status and available bandwidth. The profiler tracks several parameters for the WiFi and mobile data interfaces including the receiving and transmitting data rate. These measurements enable better estimation of the current network. It provides information such as uplink and downlink speed from local device to mobile edge by establishing connection to mobile edge. It also give the response time from mobile edge to cloud server by pinging the cloud.

3.4 PARTITIONING

Static analysis obtain the control flow graph of an application by analyzing the byte code with nodes representing methods and edges representing relations between objects. The methods and the relations between them are obtained by traversing the graph. Application partitioning is a technique of splitting the application into separate tasks, while preserving the semantics of the original application. Method based partitioning is performed here. Each method in the application is considered as a separate partition. Certain methods that cannot be offloaded to the cloud as they can be only done in local device. e.g. the methods which require input from sensors in the devices.

3.4.1 Partitioning Flow

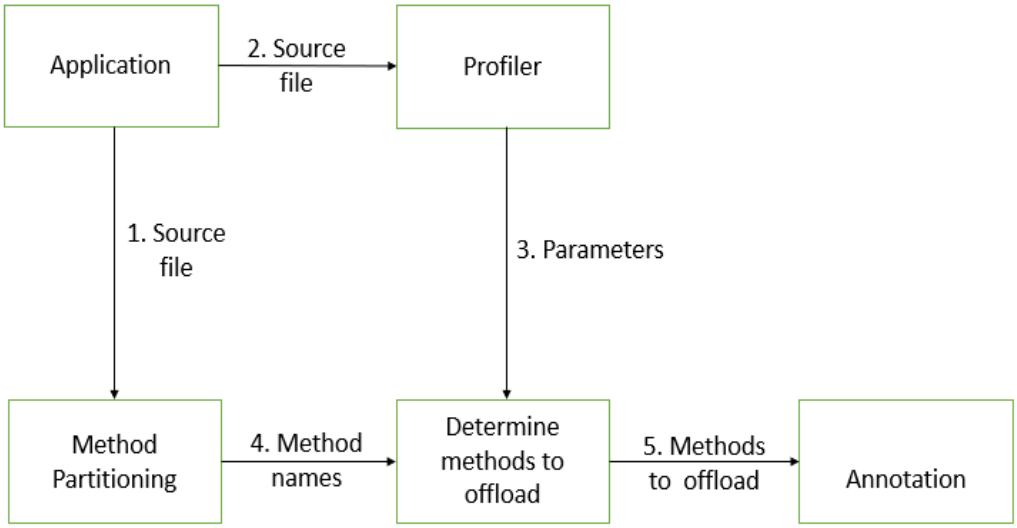

Figure 3.4: Partitioning Flow Diagram

Figure 3.4 shows the flow of application partitioning where the source files are given to partitioner (step 1) where the methods which may be offloade are annotated as "@ may be offload". Now the annotated methods from partitioner (step 4) and network profiler (step 3) helps to determine the location of execution of that annotated methods. The methods which are to be offloaded are annotated as "@offload" (step 5).

3.5 OFFLOAD DECISION MAKING

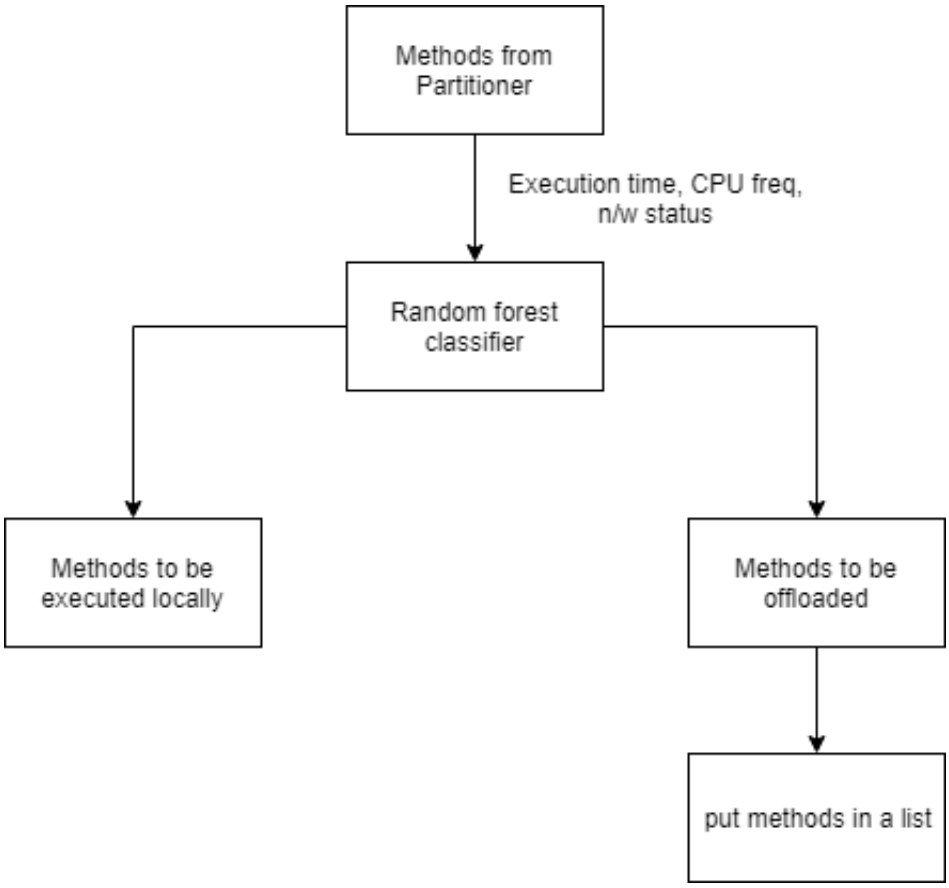

Figure 3.5: Decision maker flow diagram

Figure 3.5 shows the flow of decision maker. The decision maker use Random forest classifier to predict the location of execution of methods. It takes input as methods execution time, network status, CPU frequency from Profiler and gives the two list of methods. One list contain the methods to be executed locally and another contain the methods to be offloaded.

3.6 PARTITION OFFLOADING

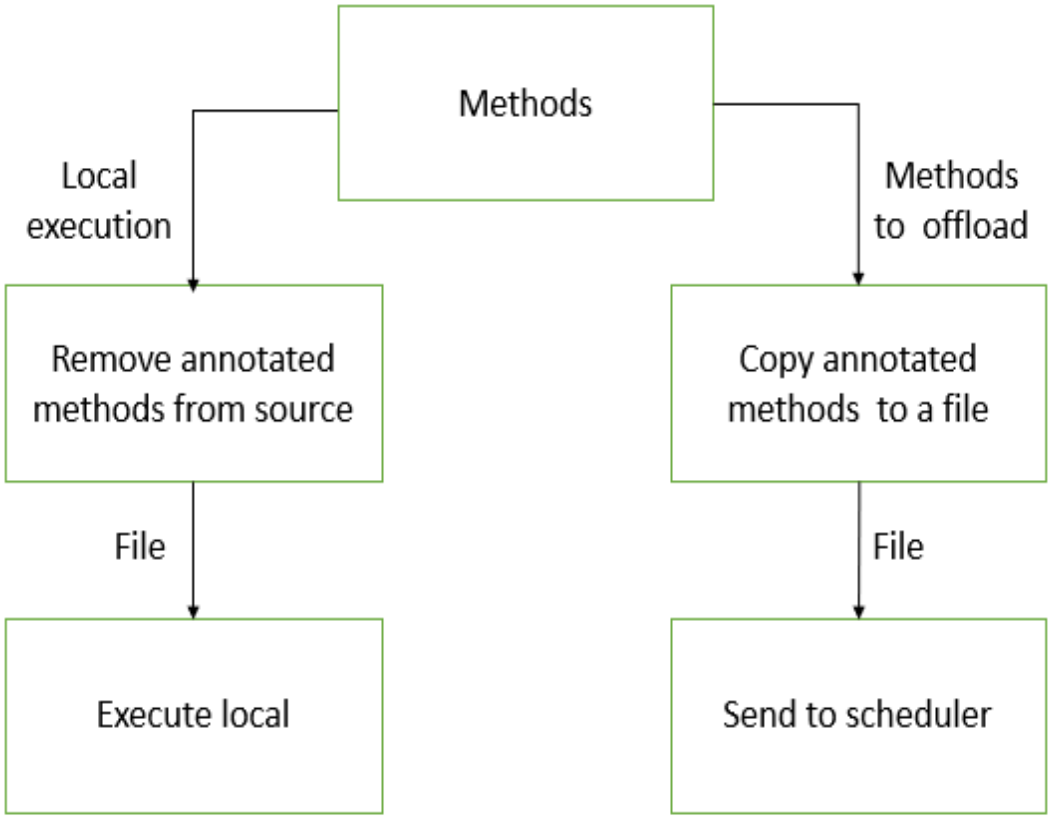

Figure 3.6: Partitioning Offloading Flow Diagram

Figure 3.5 shows the offloading of the partitioned methods to edge node/cloud. The source code of the methods which are to be offloaded are copied to a file and send to scheduler. The remaining methods are executed locally by removing the annotated methods from the source file.

3.7 SCHEDULING

DGMS is used to schedule the tasks which are to be offloaded to edge devices based on edge device's CPU utilization and network bandwidth between local device and edge device.

3.7.1 Four way Handshaking

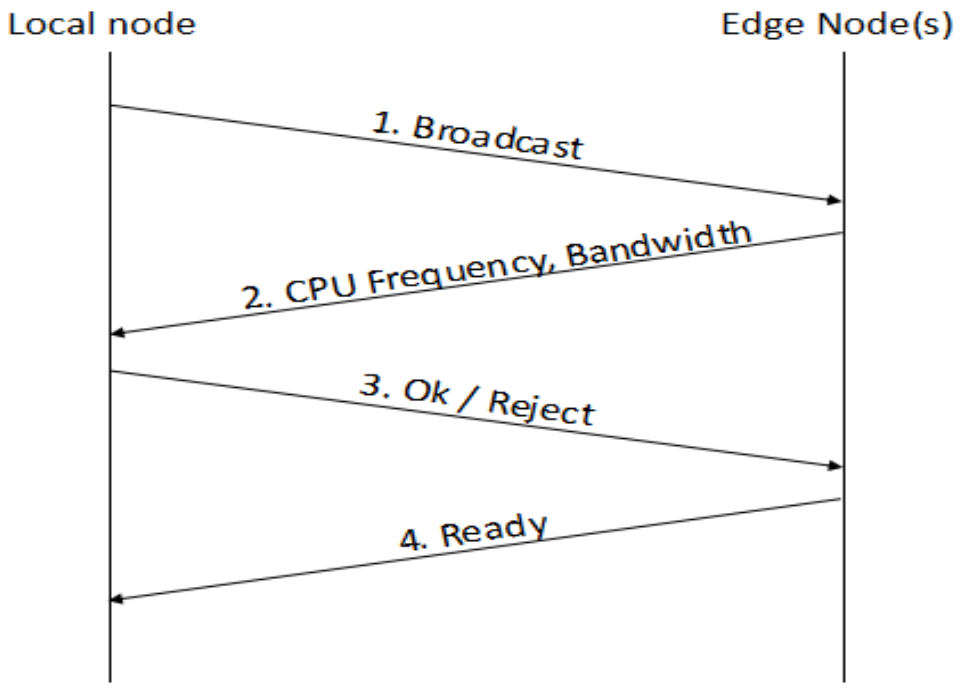

Figure 3.7: Four way Handshaking Diagram

Figure 3.7 shows the flow of interaction between the local node and edge nodes. The local node broadcasts the offloading request to nearby edge devices. The edge nodes which received the request reply with its CPU frequency, bandwidth, etc. The local node selects a edge node based on the highest parameter and send "ok" message. It also send "reject" message to remaining edge devices. The edge device

which received “ok” message will send “Ready to receive” message to that local node.

3.7.2 Scheduling Flow

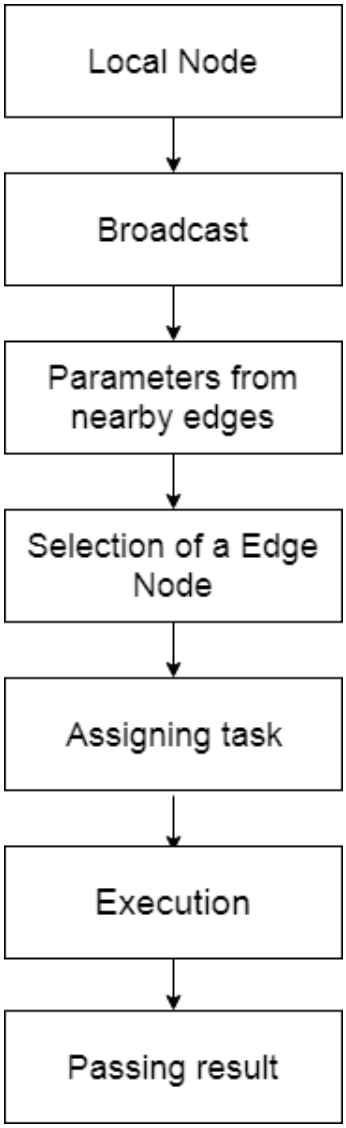

Figure 3.8: Scheduling Flow Diagram

Figure 3.8 shows the flow diagram of scheduling the tasks to the edge devices. The local device send broadcast to edge devices. It receives parameters from nearby edge devices. It selects a edge node based on the CPU frequency and bandwidth and assigns task to the selected edge device. The offloaded task is executed and the result is sent back to the local device.

Chapter 4

IMPLEMENTATION AND RESULTS

4.1 PROFILING ALGORITHM

Algorithm 4.1 Profiling

Input: None
Output: Bandwidth, Network type, Battery, CPU usage, No of cores, CPU frequency

```
1:  psutil.sensors_battery();
2:  battery.power_plugged;
3:  str(battery.percent);
4:  battery.secsleft;
5:  psutil.cpu_times();
6:  psutil.cpu_count();
7:  psutil.cpu_percent(interval=5,percpu=True);
8:  for i=0 to 2 do
9:      last_up_down = up_down;
10:     upload=psutil.net_io_counters(pernic=True)['wlan0'].bytes_sent;
11:     download=psutil.net_io_counters(pernic=True)['wlan0'].bytes_recv ;
12:     up_down = (upload,download);
13: end
```

Algorithm 4.1 shows the step by step procedure for getting various profiler parameter such as CPU cores, CPU utilization, CPU frequency, network speed and response time.

```
PROFILER OUTPUT


NUMBER OF CPU CORES :  2
CPU UTILISATION AVERAGE :  3.0
CPU CURRENT FREQUENCY :  797.969
CPU MAXIMUM FREQUENCY :  2001.0

NETWORK UPLINK SPEED :  0.0
NETWORK DOWNLINK SPEED :  0.0
EDGE NODE RESPONSE TIME :  0.00705999999999994
```

Figure 4.1: Profiler Information

Figure 4.1 shows the profiler parameter that are added to the application in order to provide the required input in the decision making of partition offloading. The program profiler collect the characteristics of application by analyzing the control flow graph. Network profiler collects information about wireless connection and the available bandwidth. Device profiler contains information about the CPU frequency and CPU utilisation average.

4.2 PARTITIONING ALGORITHM

Algorithm 4.2 Application Partitioning

Input: File(s)
Output: Graph G(V,E)

```
1: f=open(li,'r')
2: l=f.readlines()
3: s='@profile'
4: for line in enumerate(l)
5:         if(line.startswith('def'))
6:                 line=s+line
7:                 del l[j]
8:                 l.insert(j,line)
9: end
10: f.close()
11: f=open(li[i],'w')
12: for line in l
13:         f.write(line)
14: end
15: f.close()
```

Algorithm 4.2 shows the step by step procedure for partitioning the application as individual methods.

```
PARTITIONING OUTPUT


THE METHODS IN THE PROGRAM :

print_grid
find_empty_location
used_in_row
used_in_col
used_in_box
check_location_is_safe
solve_sudoku


THE METHODS THAT CAN BE OFFLOADED :
find_empty_location
check_location_is_safe
used_in_row
used_in_col
used_in_box
```

Figure 4.2: Partition output

Figure 4.2 shows the output of the partition module. It gives the list of methods in the Sudoku program and also the list of methods that may be offloaded.

4.3 OFFLOADING DECISION MAKING ALGORITHM

Algorithm 4.3 Offloading Decision

Input: parameters of methods, status of CPU and Network parameters
Output: list of methods to be offloaded

1: Randomly select “k” features from total “m” features, Where k less than m.

2: Among the “k” features, calculate the node “d” using the best split point.

3: Split the node into daughter nodes using the best split.

4: Repeat 1 to 3 steps until “l” number of nodes has been reached.

5: Build forest by repeating steps 1 to 4 for “n” number times to create “n” number of trees.

Algorithm 4.3 (Random forest algorithm) starts with randomly selecting “k” features out of total “m” features. In the next stage, randomly “k” features are selected to find the root node by using the best split approach. Then, calculate the daughter nodes using the same best split approach. The first 3 stages will run until the tree is formed with a root node and having the target as the leaf node. Finally, 1 to 4 stages are repeated to create “n” randomly created trees. This randomly created trees forms the random forest.

```
DECISION MAKING(RANDOM FOREST) RESULT :

METHODS AS INPUT TO ALGO:
find_empty_location
check_location_is_safe
used_in_row
used_in_col
used_in_box

Execution time  : 2.05s
CPU frequency   : 798

METHODS SHOULD BE OFFLOADED:
find_empty_location
check_location_is_safe
used_in_row
used_in_col
used_in_box
```

Figure 4.3: Decision Making output

The Figure 4.3 shows the output of the decision making module. It runs random forest algorithm. The algorithm takes execution time, CPU frequency, uplink and downlink as input and gives the separate list of methods to be executed in local or to be offloaded to Edge or Cloud. It gives result as either 0 or 1. If result for a method is 1, the method is offloaded to Edge or Cloud, else the method is executed locally.

4.4 PARTITION OFFLOADING ALGORITHM

Algorithm 4.4 Offloading

Input: File(s), List
Output: None

1: for methods in offload list

2: copy method to a fie

3: remove method from original file

4: send file to the scheduler

5: execute file locally

Algorithm 4.4 shows step by step procedure of offloading the partitioned methods to edge node/cloud. The methods which are to be offloaded are copied to a file and send to scheduler. The remaining methods are executed locally by removing the annotated methods from the source file.

```
OFFLOADING OUTPUT :

@localdevice(192.168.43.175):
METHODS TO BE COPIED TO FILE : sudokusolver2.py
find_empty_location
check_location_is_safe
used_in_row
used_in_col
used_in_box
code to be sent


code to be sent :sudokusolver2.py

file sent for execution...

@192.168.43.2:
file received : sudokusolver2.py

sudokusolver2.py running as subprocess 3564

connected from
192.168.43.175 -- "POST / HTTP/1.1" 200 -

execution finished killing :3564

listening for new connections...
```

Figure 4.4: Offloading output

Figure 4.4 shows the output of the Offloading module of Sudoku program. The methods to be offloaded are copied into a file and sent to scheduler.

4.5 SCHEDULING ALGORITHM

Algorithm 4.5 Distributed Greedy Maximal Scheduling

Input: file F, list L
Output:node n

```
1: initialise current_parameter, n=null
2: for node in list l
3:     send broadcast to m
4:     receive parameters from m
5:     if current_parameter is less than received parameter
6:         current_parameter= received parameter
7:         update n
8:     end if
9: end for
10: send ok to node n
11: receive ready message from node n
12: assign file f to node n
13: exceute f in n
14: receive result of f from n
```

Algorithm 4.5 shows the step by step procedure of scheduling the tasks to the edge devices. The local device send broadcast message to all nearby edge devices. It receives parameters such as CPU frequency and bandwidth and select a edge device which has highest CPU frequency and bandwidth. It assign the task to the selected edge device and receives result.

```
SCHEDULER OUTPUT :

@localdevice(192.168.43.175):
broadcast message sent

edge device information received:
192.168.43.28->['0.3', '600.0', '1200.0']
192.168.43.2->['1.8', '798.2955', '2700.0']

EDGE DEVICE SELECTED :
192.168.43.2

EDGE DEVICES REJECTED :
['192.168.43.28']

sending reject message to 192.168.43.28
sending select message  to 192.168.43.2

@192.168.43.28
Listening for new connections ...
broadcast from 192.168.43.175
sending information to 192.168.43.175
rejected

Listening for new connections ...
@192.168.43.2
Listening for new connections ...
broadcast from 192.168.43.175
sending information to 192.168.43.175
selected for execution
waiting for file ...
```

Figure 4.5: Scheduler output

Figure 4.5 shows the output of scheduler for Sudoku program. Four way handshaking happens between local and edge devices. A edge device is selected based on the Distributed greedy maximal scheduling algorithm. Methods are copied into a file and sent to the selected edge device.

4.6 RESULT INTEGRATION

Algorithm 4.6 RPC Code for Result integration

Input: File(s), list
Output: File(s)

```
1:  f=open(fname,'r')
2:  l=f.readlines()
3:  l1='import xmlrpc.client'
4:  l2='proxy=xmlrpc.client.ServerProxy("http://ip:8000/")'.format(ip=ipaddress)
5:  l.insert(0,l1 and l2)
6:  f.close()
7:  f=open(server,'w')
8:  l1='from xmlrpc.server import SimpleXMLRPCServer'
9:  l.insert(0,l1)
10: l2='= SimpleXMLRPCServer(("",8000))'
11: l.append(l2)
12: for i in offload_list:
13:         l1='server.register_function(func,"func")'.format(func=i)
14:         l.append(l1)
15: l1='server.serve_forever()'
16: l.append(l1)
```

Algorithm 4.6 shows the step by step procedure of adding the required RPC Code before the methods which are to be offloaded.

```
INPUT TO SUDOKU SOLVER :

5 3 0 0 7 0 0 0 0
6 0 0 1 9 5 0 0 0
0 9 8 0 0 0 0 6 0
8 0 0 0 6 0 0 0 3
4 0 0 8 0 3 0 0 1
7 0 0 0 2 0 0 0 6
0 6 0 0 0 0 2 8 0
0 0 0 4 1 9 0 0 5
0 0 0 0 8 0 0 7 9
```

Figure 4.6: Input of a Sudoku puzzle

Figure 4.6 shows the input of a Sudoku puzzle. The file containing the Sudoku puzzle to be solved and the source code of the Sudoku solver is given as input to the proposed system (Edge Computing Systems).

```
OUTPUT :

5 3 4 6 7 8 9 1 2
6 7 2 1 9 5 3 4 8
1 9 8 3 4 2 5 6 7
8 5 9 7 6 1 4 2 3
4 2 6 8 5 3 7 9 1
7 1 3 9 2 4 8 5 6
9 6 1 5 3 7 2 8 4
2 8 7 4 1 9 6 3 5
3 4 5 2 8 6 1 7 9
```

Figure 4.7: Output of a Sudoku puzzle

Figure 4.7 shows the output of the solved Sudoku puzzle given to the Sudoku solver. The system executes this by offloading some of the methods to the edge devices and finally integrate the results from all the utilized edge devices.

Chapter 5

CONCLUSION AND FUTURE WORK

5.1 CONCLUSION

This proposed system helps in executing the complex tasks of an application in a edge node and considerably reduces the execution time and the battery usage for local device. The mobile edge computing system setup is formed by integrating two laptops as local devices and two laptops as edge devices. They are connected through wireless medium. Profiler gives necessary information which helps in making offloading decision. Partitioner separates the application as different methods and also annotates the methods which can be offloaded. The Decision maker tells whether the task is to executed locally or somewhere else. The tasks which are to be offloaded are sent to the scheduler by transferring file. Distributed Greedy maximal scheduling is used to schedule the tasks to the nearby edge devices. The assigned tasks are executed at edge device or cloud server. The results are passed back to the local device and integrated. The offloading of computationally intensive tasks reduces the battery usage of local machine. It also increases the CPU utilization of the edge devices.

5.2 FUTURE WORK

The work can be further enhanced by monitoring the network status (loss of connection between local device and edge device, packet loss) continuously for making appropriate decision at the run time.

REFERENCES

[1] A. Ahmed and E. Ahmed, "A survey on mobile edge computing," in *2016 10th International Conference on Intelligent Systems and Control (ISCO)*, vol. 57, pp. 1–8, Jan 2016.

[2] B. Garvelink, "Mobile edge computing: a building block for 5G," *Telecom Paper*, vol. 2, pp. 183–184, July 2015.

[3] D. Satria, D. Park, and M. Jo, "Recovery for overloaded mobile edge computing," *Future Generation Computing System*, vol. 70, pp. 138–147, 2017.

[4] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Transactions on Networking*, vol. 24, pp. 2795–2808, October 2016.

[5] T. Q. Dinh, J. Tang, Q. D. La, and T. Q. S. Quek, "Offloading in mobile edge computing: Task allocation and computational frequency scaling," *IEEE Transactions on Communications*, vol. 65, pp. 3571–3584, Aug 2017.

[6] Y. Yang, Y. Ma, W. Xiang, X. Gu, and H. Zhao, "Joint optimization of energy consumption and packet scheduling for mobile edge computing in cyber-physical networks," *IEEE Access*, vol. 6, pp. 15576–15586, 2018.

[7] H. Shah-Mansouri, V. W. S. Wong, and R. Schober, "Joint optimal pricing and task scheduling in mobile cloud computing systems," *IEEE Transactions on Wireless Communications*, vol. 16, pp. 5218–5232, Aug 2017.

[8] W. Chen, D. Wang, and K. Li, "Multi-user multi-task computation offloading in green mobile edge cloud computing," *IEEE Transactions on Services Computing*, vol. 18, pp. 1–1, 2018.

[9] 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*, vol. 14, pp. 4642–4655, Oct 2018.