A Major Project Report

On

COMPUTE WITH NEARBY MOBILE DEVICES USING WORKSTEALING ALGORITHM – CLOUD TECHNOLOGY

Submitted in partial fulfillment of the requirements for the award of degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

Under the Guidance of

Mr.L.Manikandan

Assistant Professor

By

M. Rachana (18D21A05L3)

K. Supraja (18D21A05J6)

P. Siri (18D21A05M1)

ESTD: 2001

SRIDEVI WOMEN'S ENGINEERING COLLEGE

Department Of Computer Science and Engineering

(Approved by AICTE, affiliated to JNTU, HYD and Accredited by NBA and NAAC
An ISO 9001:2015 Certified Institution)

V.N.PALLY, Gandipet, Hyderabad-107

2021-2022

Department of Computer Science and Engineering

SRIDEVI WOMEN'S ENGINEERING COLLEGE

(Approved by AICTE, affiliated to JNTUH and Accredited by NBA and NAAC

An ISO 9001:2015 Certified Institution)

V.N.PALLY, Gandipet, Hyderabad-107

2021-2022

CERTIFICATE

This is to certify that the MAJOR PROJECT report entitled "**COMPUTE WITH NEARBY MOBILE DEVICES USING WORKSTEALING ALGORITHM – CLOUD TECHNOLOGY"** is being submitted by **M.Rachana(18D21A05L3), K.Supraja (18D21A05J6), P.Siri (18D21A05M1)** in partial fulfillment for the award of degree of **Bachelor of Technology in Computer Science and Engineering** is a record of bonafide work carried out by them.

UNDER THE GUIDANCE OF	**COORDINATOR**	**HEAD OF THE DEPARTMENT**
Mr. L. Manikandan	Dr.M.RamaSubramanian	Dr. A. Gautami Latha
Assistant Professor	Professor	Professor & HOD

EXTERNAL EXAMINER

TruProjects
Find Your Dream Projects

CERTIFICATE OF COMPLETION

DATE: 30/05/2022

This is to certify that **RACHANA MIRIYALA, SUPRAJA KONGARA, SIRI PATLOLLA** bearing with **REGNOS** : **18D21A05L3, 18D21A05J6, 18D21A05M1** from **SRIDEVI WOMEN'S ENGINEERING COLLEGE, HYDERABAD,** successfully completed **MAJOR PROJECT** work at **TRU PROJECTS EDUCATIONAL SERVICES PRIVATE LIMITED from 16/11/2021 to 30/05/2022** during the project. They worked on the project entitled **"COMPUTE WITH NEARBY MOBILE DEVICES USING WORKSTEALING ALGORITHM – CLOUD TECHNOLOGY".**

They were found punctual, hardworking and interested to learn the technologies. During the project work they demonstrated good skills with self—motivate attitude towards learning.

Their association with the team was fruitful. We wish them all the best for future!

For**TRUPROJECTS**

J. Manoj Kumar

JMANOJKUMAR

MANAGINGDIRECTOR

Tru Projects Educational Services Pvt. Ltd. HYDERABAD

RS BROTHERS BUILDING
The Pavani Prestige 407, 4th floor,
Ameerpet, Hyderabad, Telangana

info@truprojects.in
www.truprojects.in

+91 96761 90678

DECLARATION

We hereby declare that the major project entitled **"COMPUTE WITH NEARBY MOBILE DEVICES USING WORKSTEALING ALGORITHM – CLOUD TECHNOLOGY"** is the work done during the period from 16.11.2021 to 30.05.2022 and submitted in partial fulfillment of the requirements for the award of degree of Bachelor of Technology in Computer Science and Engineering from Jawaharlal Nehru Technological University, Hyderabad.

M.Rachana 18D21A05L3
K.Supraja 18D21A05J6
P.Siri 18D21A05M1

ACKNOWLEDGMENT

There are many people who helped, directly or indirectly to complete our major project successfully. We would like to take this opportunity to thank one and all. First of all, we would like to express our deep gratitude towards our internal guide **Mr. L. Manikandan,** Assistant Professor, Department of CSE for his support in completion of our major project if any dissertation.

We express our sincere gratitude to Coordinator **Dr. M. Ramasubramanian**, professor, Department of CSE, for his support and motivation, and cooperation for the successful completion of this project.

We also wish to express our sincere thanks to **Dr. A. Gautami Latha**, Professor and HOD, Department of CSE for her valuable advices and suggestions.

We are also extremely thankful to **Dr. B. L. Malleswari**, Principal, Sridevi Women's Engineering College, for providing the facilities to complete the project.

Finally, we would like to thank all our family and friends for their help and constant cooperation during our project period. Finally, we are very much intended to our parents for their moral support and encouragement.

M.Rachana 18D21A05L3
K.Supraja 18D21A05J6
P.Siri 18D21A05M1

INDEX

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

As mobile devices evolve to be powerful and pervasive computing tools, their usage also continues to increase rapidly. However, mobile device users frequently experience problems when running intensive applications on the device itself, or offloading to remote clouds, due to resource shortage and connectivity issues. Ironically, most users' environments are saturated with devices with significant computational resources. This paper argues that nearby mobile devices can efficiently be utilized as a crowd-powered resource cloud to complement the remote clouds. Node heterogeneity, unknown worker capability, and dynamism are identified as essential challenges to be addressed when scheduling work among nearby mobile devices.

We present a work-sharing model, called Honeybee, using an adaptation of the well-known work stealing method to load balance independent jobs among heterogeneous mobile nodes, able to accommodate nodes randomly leaving and joining the system. The overall strategy of Honeybee is to focus on short-term goals, taking advantage of opportunities as they arise, based on the concepts of proactive workers and opportunistic delegator. We evaluate our model using a prototype framework built using Android and implement two applications. We report speedups of up to 4 with seven devices and energy savings up to 71% with eight devices.

CHAPTER 1
INTRODUCTION

1.1 PURPOSE

A different unique model is desired to build and can be differentiated in terms of using only local mobile resources opportunistically, satisfying the requirements of a mobile device cloud of being proactive, opportunistic and load-balanced while showing speedups and energy savings in an actual implementation. Our focus is on a model that can be used to implement a variety of tasks, not limited to query processing, sensing, or human validation.

1.2 SCOPE

Today's environments are becoming embedded with mobile devices with augmented capabilities, equipped with various sensors, wireless connectivity as well as limited computational resources. Whether we are on the move, on a train, or at an airport, in a shopping center or on a bus, a plethora of mobile devices surround us every day, thus creating a resource-saturated ecosystem of machine and human intelligence.

However, beyond some traditional web-based applications, current technology does not facilitate exploiting this resource rich space of machine and human resources. Collaboration among such smart mobile devices can pave the way for greater computing opportunities, not just by creating crowd-sourced computing opportunities needing a human element, but also by solving the resource limitation problem inherent to mobile devices. While there are research projects in areas such as mobile grid computing where mobile work sharing is centrally coordinated by a remote server (HTC power to give1) and crowd-powered systems using mobile devices (Kamino2 , Parko3) a gap exists for supporting collective resource sharing without relying on a remote entity for connectivity and coordination.

1.3 MODEL DIAGRAM/OVERVIEW

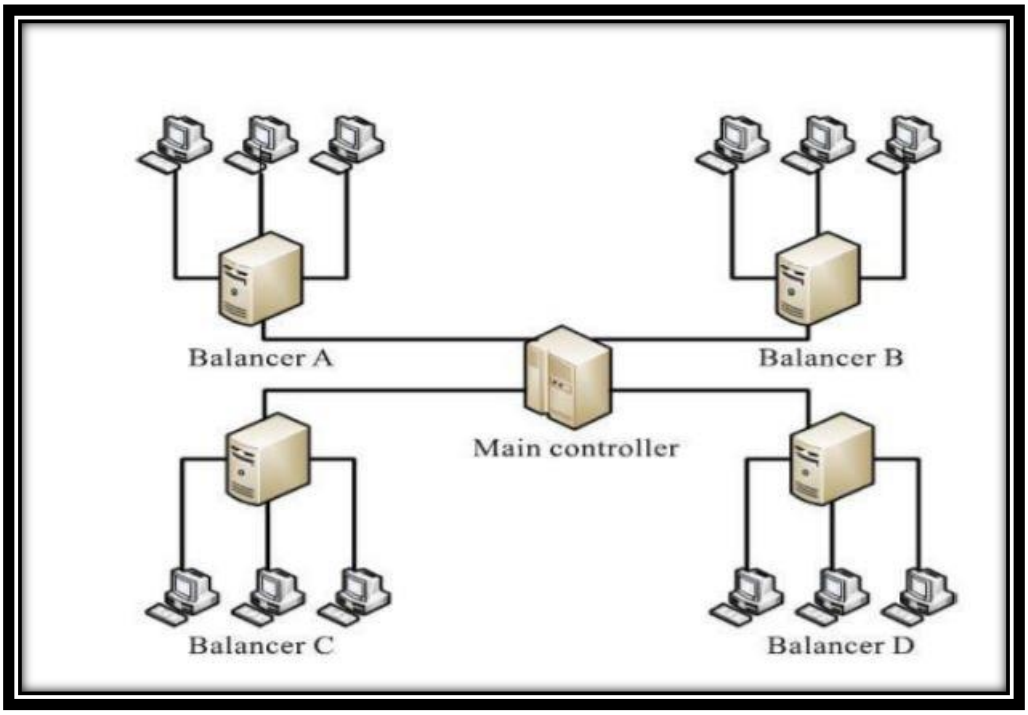

Fig. 1.1 Model Diagram

The above model diagram depicts the information how Main Device control different mobile device.

Here, Main controlling device is termed as Main controller and other devices that are controlled are termed as Balancers.

CHAPTER 2
LITERATURE SURVEY

Cuckoo, based on the Ibis communication middleware, to offload to a remote resource, and supports Bluetooth with Wi-Fi and cellular . Although Honeybee has used Bluetooth in previous versions, the current implementation uses Wi-Fi Direct due to better speeds and range.

FemtoCloud proposes an opportunistic mobile edge-cloud platform that offloads jobs to nearby mobiles, similarly to Honeybee. However, whereas Honeybee does not require prior information about the computational capabilities of the worker nodes to load- balance the task, FemtoCloud's scheduling strategy depends on periodic capability estimations of each worker node.

At the other end of the spectrum, crowd computing has been shown to have the potential to use mobile devices in a social context to perform large scale distributed computations, via a static farming method.However, our results show that the work stealing method can provide better results.

Social aware task farming has been proposed as an improvement on simple task farming, and social aware algorithms show better performance in their simulation based on real world human encounter traces. In the future we hope to build on this result (social aware task sharing) as an incentive for participation.

In, human expertise is used to answer queries that prove too complicated for search engines and database systems, and in Crowd- Search, image search on mobile devices is performed with human validation via Amazon Mechanical Turk. A generic spatial crowdsourcing platform using smartphones is discussed, where queries are based on location information. Mobile phones are used to collect sensor data on Medusa, according to sensing tasks specified by users.

In Rankr, an online mobile service is used to ask users to rank ideas and photos. These are primarily concerned with the crowdsourcing aspect, using mobile devices as tools to access an online crowdsourcing service that is hosted on a remote server.

In contrast, Honeybee defines the crowd as the surrounding mobile devices and their users, and focuses on sharing the tasks on a crowd of local mobile devices with performance gain and saving energy as the main goal. Indeed, results from the above research show us that user participation is at a considerable level, and using micro payments for such ‘micro tasks’ is viable.

CHAPTER 3
SYSTEM ANALYSIS

3.1 EXISTING SYSTEM

There are several unique features that differentiate mobile crowed environments from a typical grid/distributed computing cluster, such as less computation power and limited energy on nodes, node mobility resulting in frequent disconnections, and node heterogeneity. Hence, solution from grid/distributed computing cannot be used as they are, and need to be adapted to suit the requirements of mobile crowd environments.

This model presents the Honeybee model that supports P2P work sharing among dynamic mobile nodes. As proof of concept we present the Honeybee API, a programming frame Work for development mobile crowed computing applications.

We build on previous work where we initially investigated static job farming among a heterogeneous group of mobile devices in, which was followed by a more self adaptive approach in using the 'work stealing' method and in where three different mobile crowd sourcing applications were implemented and evaluated. The progress of our research on work share for mobile edge-clouds .

3.1.1 DISADVANTAGES OF EXISTING SYSTEM

- Load balancing schemes depending on whether the system dynamics are important can be either static or dynamic.
- Static schemes do not use the system information and are less complex.
- A dynamic scheme is used for its flexibility

3.2 PROBLEM STATEMENT

Node heterogeneity, unknown worker capability, and dynamism are identified as essential challenges to be addressed when scheduling work among nearby mobile devices. Present a work sharing model, called Honeybee, using an adaptation of the well-known work stealing method to load balance independent jobs among heterogeneous mobile nodes, able to accommodate nodes randomly leaving and joining the system

3.3 PROPOSED SYSTEM

- Our results show that the work stealing methods can provide better results. Social aware task farming has been proposed as an improvement on simple task farming and social aware algorithms show better performance in their simulation based on real world human encounter traces. In the future we hope to build on this result (social aware task sharing) as an incentive for participation.
- In, Human expertise is used to answer queries that prove too complicated for search engine and database systems, and in Crowed –search, Image Search on mobile devices is performed with human validation via amazon e-mechanical Truk.
- A generic spatial crowd sourcing platform using smart phones is discussed in, where queries are based on location information. Mobile Phones are used to collect sensor data on Medusa, according to sensing task specified by users. In Rankr, an online mobile service is used to ask users to rank ideas and photos.
- These are primarily concerned with the crowd sourcing aspect, using mobile devices as tools access an online crowd sourcing service that is hosted on a remote server. In contrast, Honeybee defines the crowd as the surrounding mobiles devices and their users and focuses on sharing the tasks on a crowd of local mobile devices with performance gain and saving energy as the main goal.

3.3.1 ADVANTAGES OF PROPOSED SYSTEM

Each of the three methods have advantages depending mainly on the existence of high connectivity, additional infrastructure or node encounters respectively.

In our work, we focus on the third methods, ie., opportunistically sharing work with the surrounding mobile devices, owing to issues with the other two approaches in cases of low network availability and lack of established infrastructure.

Furthermore, in Honeybee, we also recognize the potential of using mobile devices as agents of crowd sourcing, thereby exploiting the collective power of human expertise and machine resources

CHAPTER 4

SYSTEM REQUIREMENT SPECIFICATION

4.1 FUNCTIONAL REQUIREMENTS

1. Generating Mobile Devices
2. Preprocess Image Data
3. Local Run Task
4. Offload Task
5. Extension compress and offload Task
6. Response Time
7. Task Execution Time Graph

4.2 NON-FUNCTIONAL REQUIREMENTS

NON-FUNCTIONAL REQUIREMENT (NFR) specifies the quality attribute of a software system. They judge the software system based on Responsiveness, Usability, Security, Portability and other non-functional standards that are critical to the success of the software system. Example of nonfunctional requirement, *"how fast does the website load?"* Failing to meet non-functional requirements can result in systems that fail to satisfy user needs. Non- functional Requirements allows you to impose constraints or restrictions on the design of the system across the various agile backlogs. Example, the site should load in 3 seconds when the number of simultaneous users is > 10000. Description of non-functional requirements is just as critical as a functional requirement.

- Data Integrity
- Scalability
- Interoperability
- Reliability

4.3 HARDWARE REQUIREMENTS

Minimum hardware requirements are very dependent on the particular software being developed by a given Enthought Python / Canopy / VS Code user. Applications that need to store large arrays/objects in memory will require more RAM, whereas applications that need to perform numerous calculations or tasks more quickly will require a faster processor.

- **Operating system : windows, linux**
- **Processor : minimum intel i3**
- **Ram : minimum 4 GB**
- **Hard disk : minimum 250gb**

4.4 SOFTWARE REQUIREMENTS

The functional requirements or the overall description documents include the product perspective and features, operating system and operating environment, graphics requirements, design constraints and user documentation.

The appropriation of requirements and implementation constraints gives the general overview of the project in regards to what the areas of strength and deficit are and how to tackle them.

- **Python ide 3.7 version**
- **Anaconda**
- **Jupiter**
- **Google Colab**

CHAPTER 5
SYSTEM DESIGN

5.1 SYSTEM ARCHITECTURE

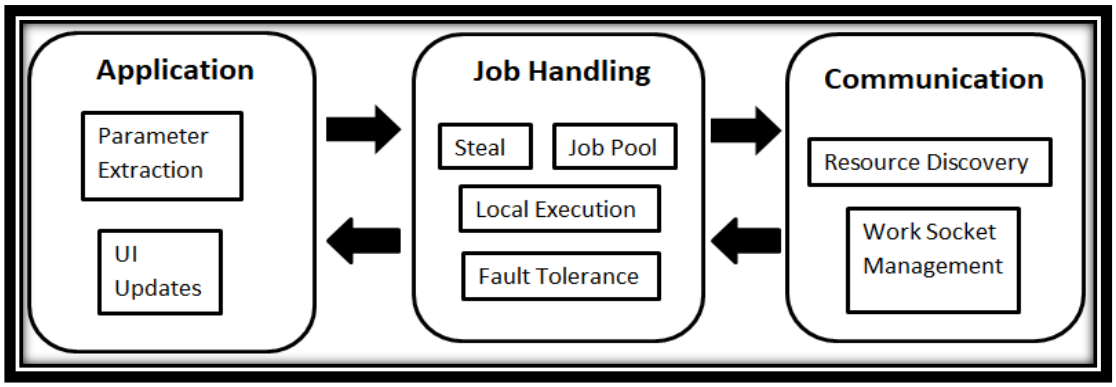

Fig 5.1 SYSTEM ARCHITECTURE

Honeybee is implemented on Android, using Wi-Fi Direct as the communication protocol. Application developers can use the methods and interfaces provided by the framework for writing work sharing mobile apps. As shown in Figure 6, the framework contains three main components responsible for the main areas of Application interfacing, Job Handling, and Communication

5.1.1 The Application Component

The Application layer method interfaces between application specific code and the core structure. At the starting point of execution, the framework extracts the application specific parameters via the AppRequest interface that provides abstractions to represent the task as a list of jobs.

5.1.2 The Job Handling Component

The initialization of the job pool begins with processing the AppRequest object. Once the AppRequest task is broken into sub tasks, or jobs, they are stored in two data structures: in an array all Jobs that is not modified throughout execution and in a ConcurrentLinkedQueue jobList, which is subject to change dynamically, as jobs

are removed and added from accessing threads. The array is maintained for validation purposes.

During the lifetime of the program, jobList is accessed by several processes as given below:

1) Delegator's local job execution thread: Jobs are consumed by polling the jobList from the head.

2) Steal request threads from workers: as steal requests arrive from the Communication component, they are processed by the Job Handling component which creates Callable instances to consume jobs in k chunks from jobList. More than one steal requests may arrive and be processed at the same time.

3) Threads carrying stolen jobs from workers: When the delegator steals jobs from workers, they are passed from the Communication component to the Job handling component, where the job parameters are decoded and assembled into jobs, and added to the jobList.

4) Fault tolerance threads: as fault tolerance mechanisms, jobs that were already assigned to workers may presumed 'lost' and be added back to jobList. This takes place when jobs expire, or worker heartbeats are missed.

In cases (3) and (4) above, when jobs are added, new local execution threads are spawned as Runnable tasks and added to a single thread pool, thereby ensuring that only one local execution thread is running at a given time. These are then executed locally as described in case (1).

5.1.3 The Communication Component

Potential workers are identified by running resource discovery every t seconds. Whenever a new resource is detected, the user has the choice to initiate a connection. For each successful worker connection, a reading thread is kept alive throughout the lifetime of the connection as the delegator needs to receive various messages from the workers at intermittent intervals. The messages expected to be received and written by the delegator are summarized in Table 5.1.

READ	WRITE
1. Steal requests by workers	1. Jobs stolen successfully by workers
2.Workers' acknowledgement of receiving job data	2. Reply to unsuccessful steal attempts by workers (when the delegator does not have any jobs for workers to steal)
3. Negative replies to steal attempts by the delegator (when a worker does not have any jobs)	3. Steal requests from delegator (when the delegator attempts to steal from others)
4. Stolen jobs in cases of successful steals	4. Termination signal sent to workers once delegator verifies all jobs have been completed.
5. Results sent by workers	
6. Worker heartbeats	

TABLE 5.1: **Types of I/O messages handled by the Model .**

5.2 SYSTEM COMPONENTS (MODULES)

In proposed paper is performing following steps

- **Module 1 – Local Run Task :** Task is done in the local system using this module and this returns required time to execute
- **Module 2 – Generating Mobile Devices:** This module deals with generating nearby mobile devices to the inputted mobile device id.
- **Module 3 – Offload Task :** Using this module mobile devices are connected using cloud and work is done using two or more mobile devices using work stealing and honeybee algorithm and this returns execution time.
- **Module 4 – Extension Offload Task :** Using this module, devices are connected in cloud, using pill module offload task is being performed and this returns response time
- **Module 5 – Task Execution Time Graph :** Compares the local run task time and offload task time and Extension Offload task time required to execute and plots a bar graph

5.3 DATA FLOW DIAGRAM AND UML DIAGRAMS

5.3.1 DATA FLOW DIAGRAM

Data Flow Diagram below depicts the flow of data in the model presented.

Data, initially is Mobile id and that is inputted by User and passed through further modules – Local Run Task, Generate Mobile Devices.

From which Local Run Task gets executed and Passes the data to the last module 'Task Execution Time Graph'. The Other module Generate Mobile Devices passes date to other two modules for further processing. They are Offload Task and Extension Offload Task which further passes data after processing to Time Execution Time Graph.

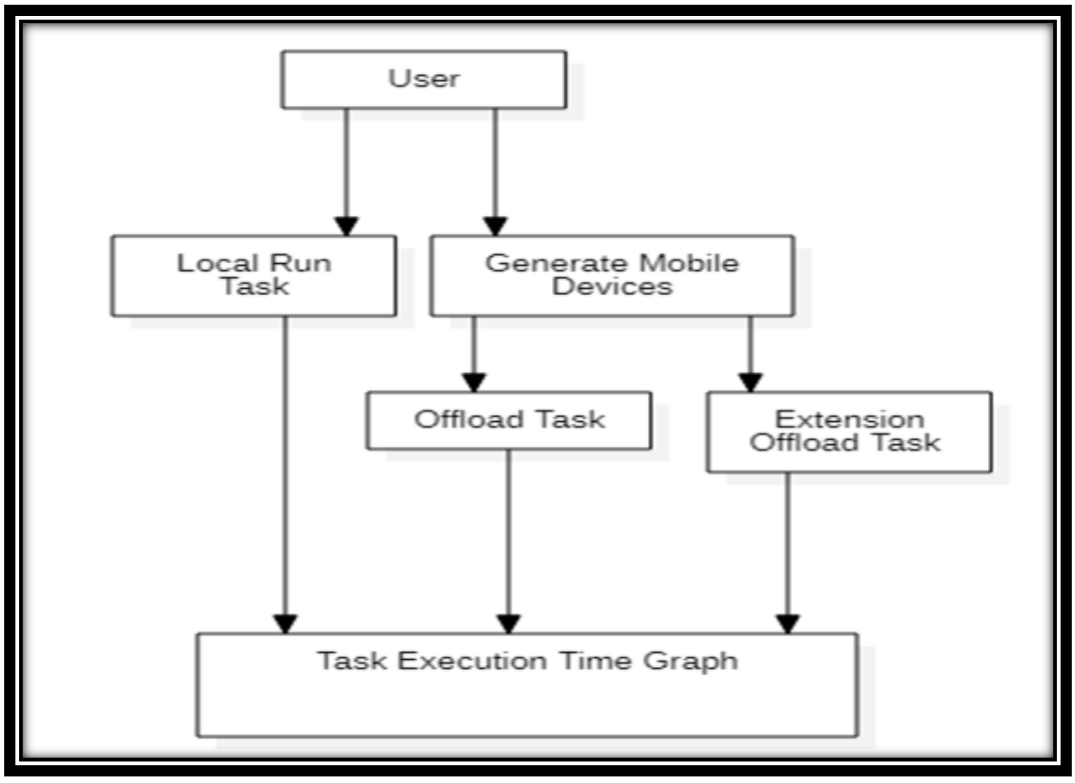

Fig 5.2 DATA FLOW DIAGRAM

5.3.2 UML DIAGRAMS

The underlying premise of UML is that no one diagram can capture the different elements of a system in its entirety.

5.3.2.1 Use case diagram:

The use case diagram is used to identify the primary elements and processes that form the system. The primary elements are termed as "actors" and the processes are called "use cases." The use case diagram shows which actors interact with each use case.

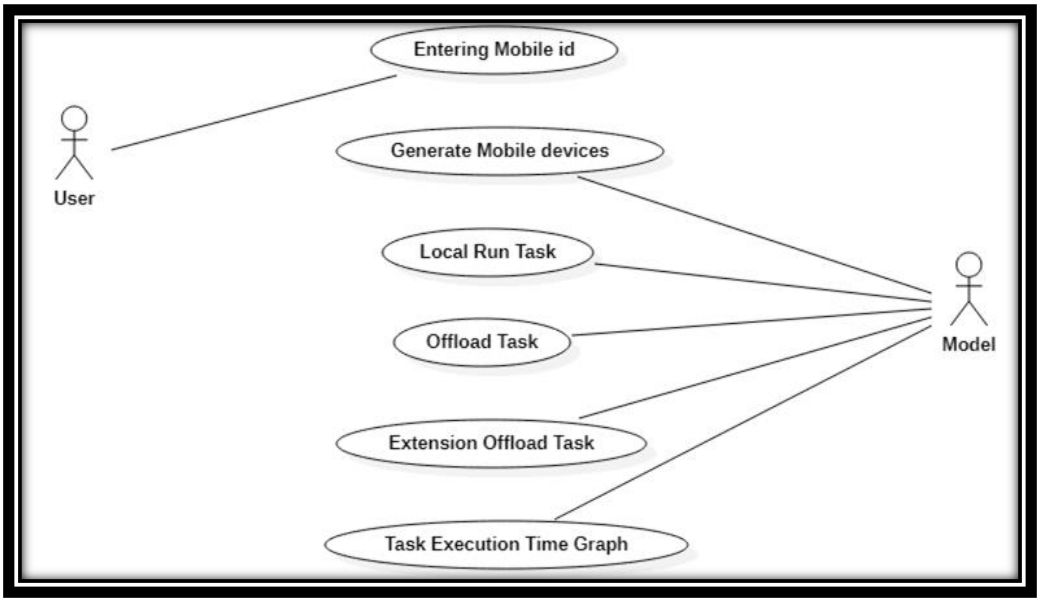

Fig 5.3 Use Case Diagram

To identify the primary elements and processes that form the system, the "actors" considered are User, Model and the "use cases" considered are Entering Mobile Id, Generate Mobile Devices, Local Run Task, Offload Task, Extension Offload Task, Task Execution Time Graph.

5.3.2.2 Class diagram:

The class diagram is used to refine the use case diagram and define a detailed design of the system. The class diagram classifies the actors defined in the use case diagram into a set of interrelated classes. The relationship or association between the classes can be either an "is-a" or "has-a" relationship. Each class in the class diagram may be capable of providing certain functionalities. These functionalities provided by the class are termed "methods" of the class.

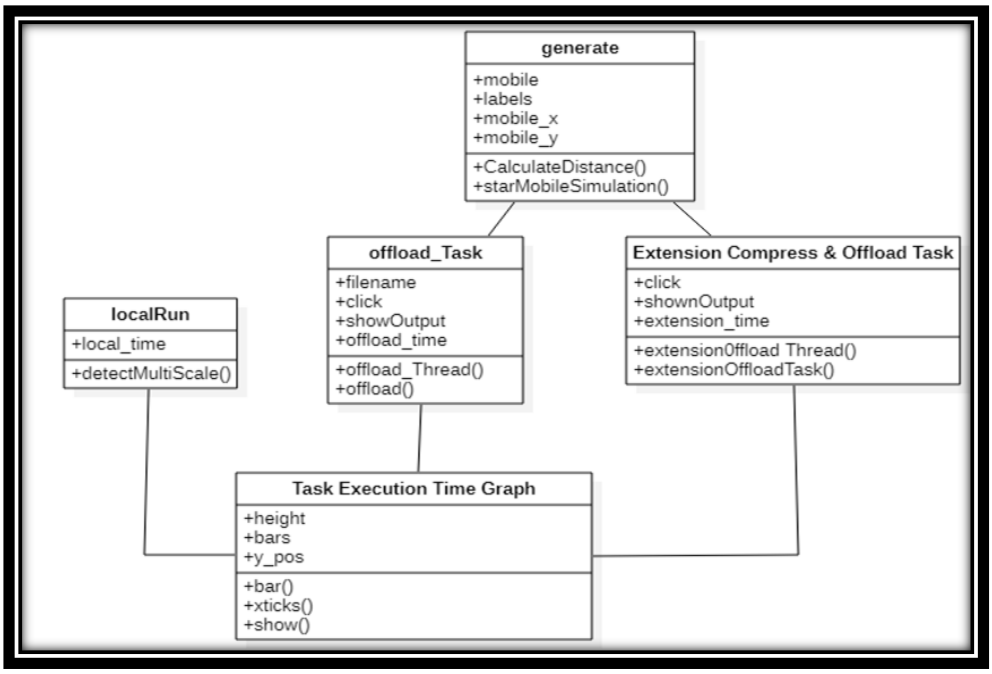

Fig 5.4 Class Diagram

To refine the use case diagram and define a detailed design of the system, the class diagram classifies the actors defined in the use case diagram into a set of interrelated classes. The classes considered in our model is generate, localRun, offload_Task, Extension Compress & Offload Task, Task Execution Time Graph

5.3.2.3 State Chart diagram:

A state diagram, as the name suggests, represents the different states that objects in the system undergo during their life cycle. Objects in the system change states in response to events. In addition to this, a state diagram also captures the transition of the object's state from an initial state to a final state in response to events affecting the system.

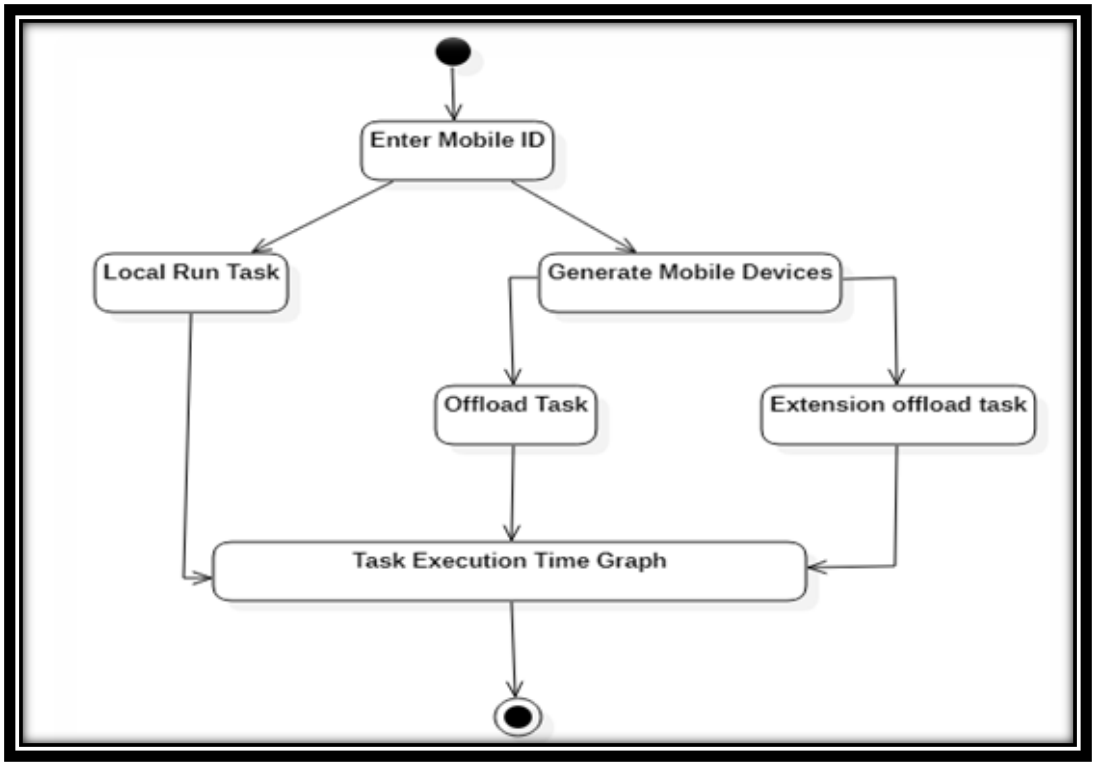

Fig 5.5 State Diagram

To represent the different states that objects in the system undergo during their life cycle, States considered here are Enter Mobile ID, Local Run Task, Generate Mobile Devices, Offload Task, Extension Offload Task, Task Execution Time Graph

5.3.2.4 Activity diagram:

The process flows in the system are captured in the activity diagram. Similar to a state diagram, an activity diagram also consists of activities, actions, transitions, initial and final states, and guard conditions.

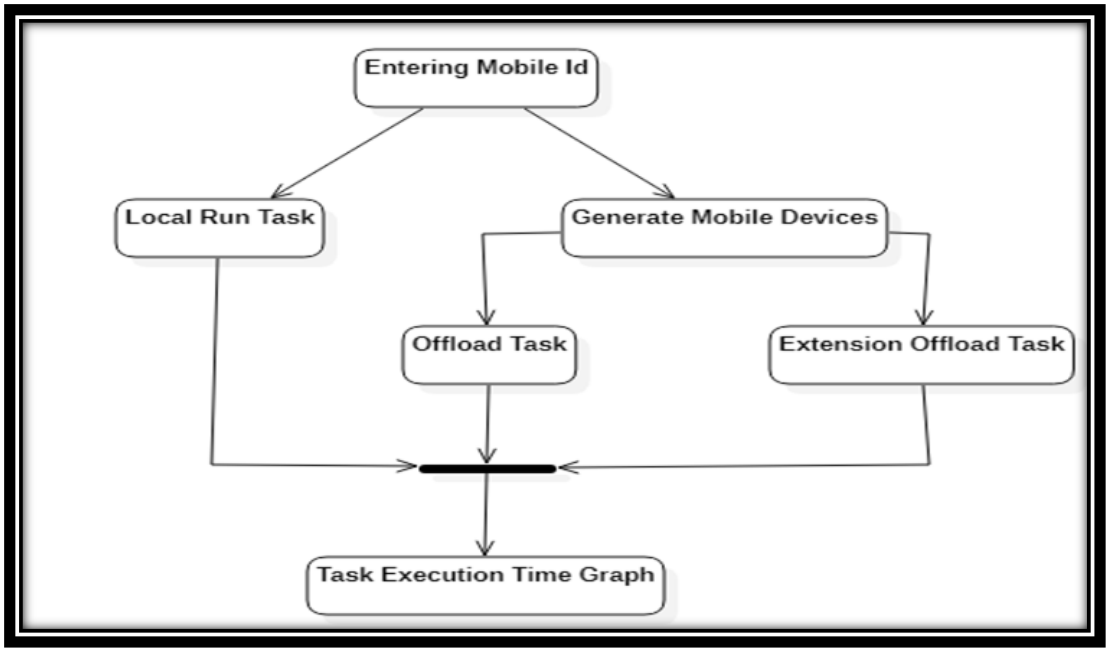

Fig 5.6 Activity Diagram

Capturing the process flows in the system, the activity diagram is constructed. The activities, actions, transitions, initial and final states, and guard conditions considered while building the model are presented below.

5.3.2.5 Sequence diagram:

A sequence diagram represents the interaction between different objects in the system. The important aspect of a sequence diagram is that it is time-ordered. This means that the exact sequence of the interactions between the objects is represented step by step. Different objects in the sequence diagram interact with each other by passing "messages".

The interaction between different objects in the presented system is illustrated below. The presented model depicts three different flows of execution considered in the model.

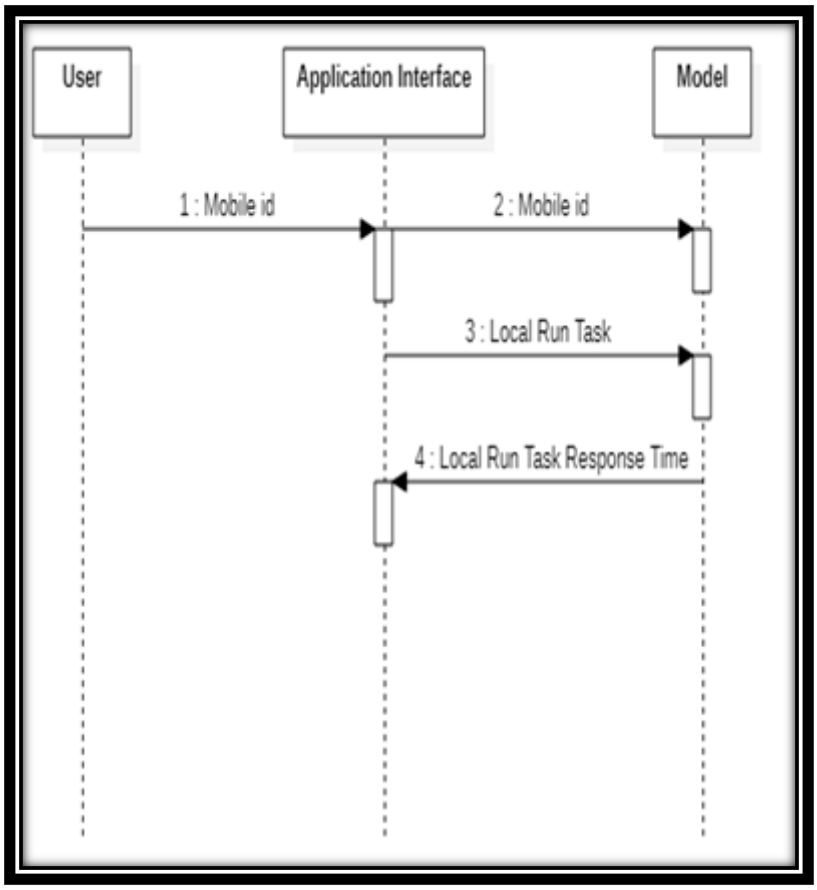

Fig 5.7 Sequence Diagram 1

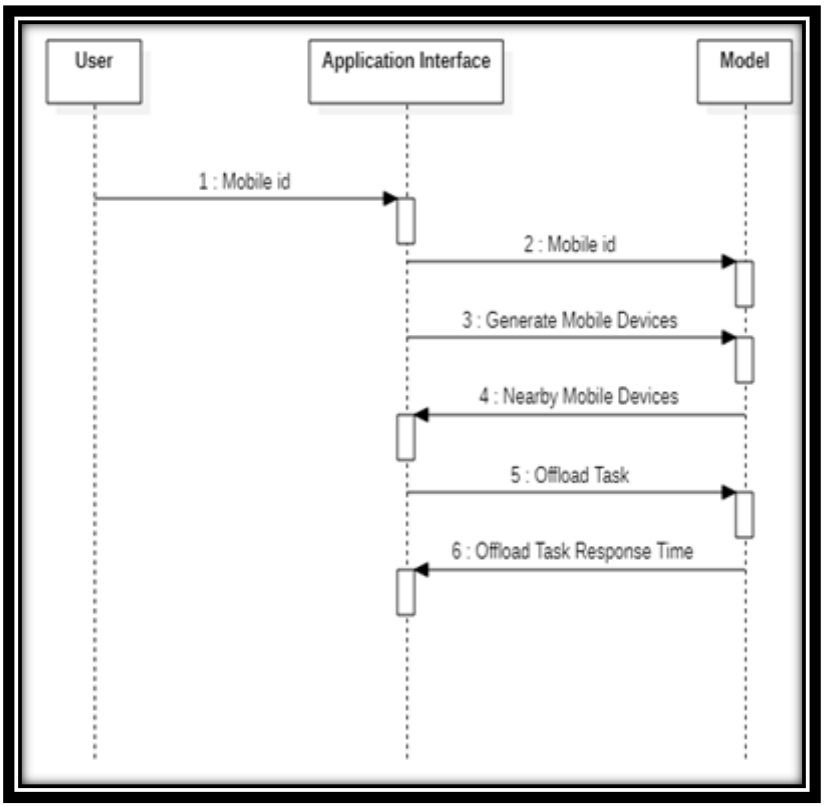

Fig 5.8 Sequence Diagram 2

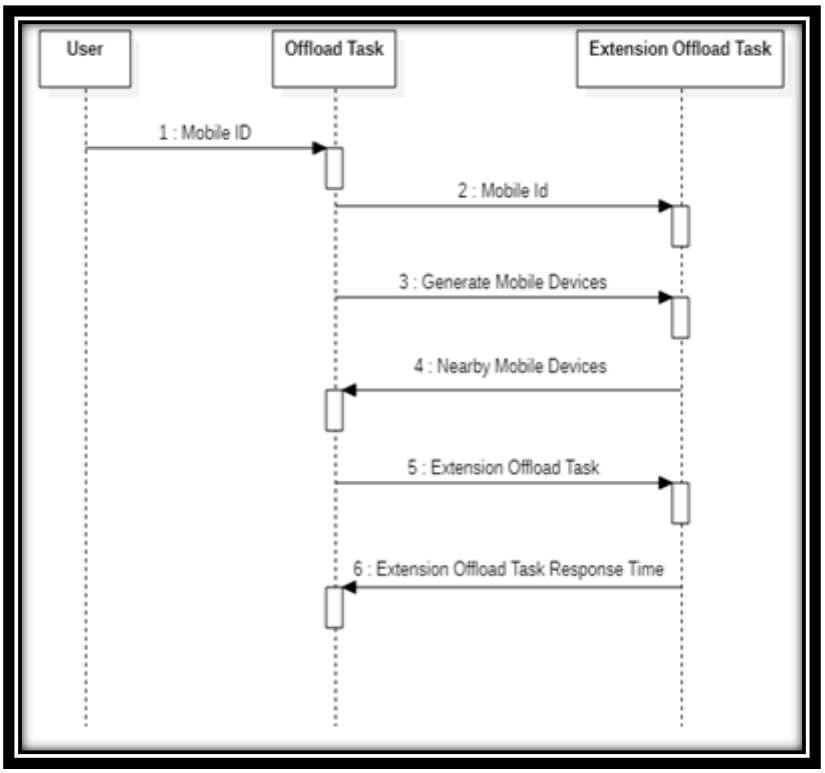

Fig 5.9 Sequence Diagram 3

5.3.2.6 Deployment diagram:

The deployment diagram captures the configuration of the runtime elements of the application. This diagram is by far most useful when a system is built and ready to be deployed.

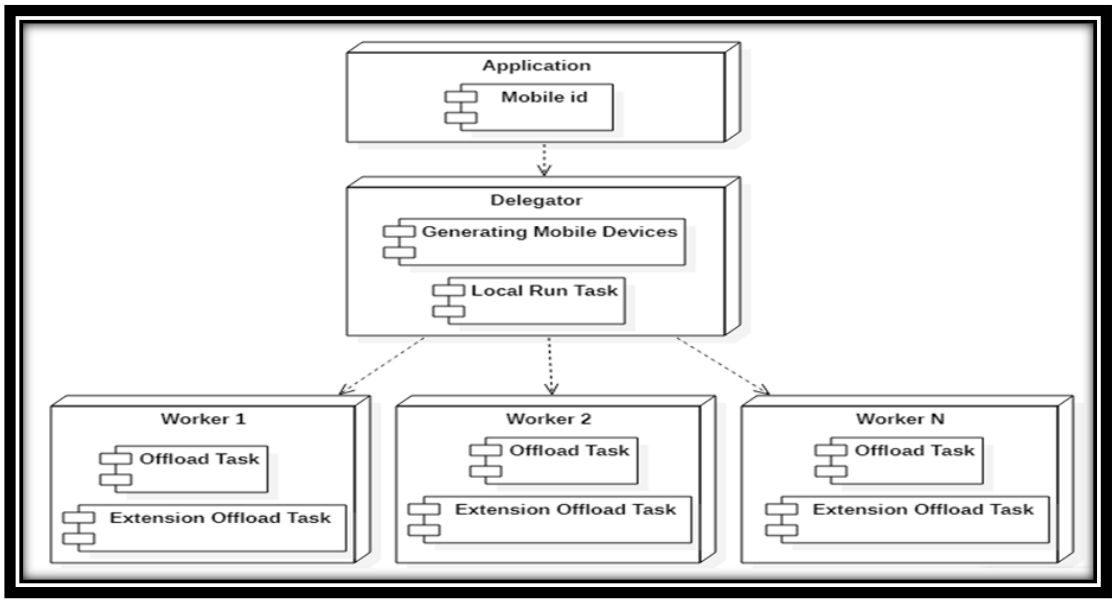

Fig 5.10 Deployment Diagram

To capture the configuration of the runtime elements of the model that is built, the following deployment model is illustrated.

CHAPTER 6
IMPLEMENTATION

6.1 SAMPLE CODE

6.1.1 Implementing Workers Model

6.1.1.1 Defining the global variables

```
import socket

import sys

import cv2

import pickle

import numpy as np

import struct ## new

import zlib


HOST='localhost'

PORT=8485


cascPath = "haarcascade_frontalface_default.xml"

faceCascade = cv2.CascadeClassifier(cascPath)

encode_param = [int(cv2.IMWRITE_JPEG_QUALITY), 90]

s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)

s.bind((HOST,PORT))

s.listen(10)

print('Workers started')
```

6.1.1.2 Implementing finding workers method

```
while True:
    conn,addr=s.accept()
    data = b""
    payload_size = struct.calcsize(">L")
    print("payload_size: {}".format(payload_size))
    while len(data) < payload_size:
        print("Recv: {}".format(len(data)))
        data += conn.recv(4096)

    print("Done Recv: {}".format(len(data)))
    packed_msg_size = data[:payload_size]
    data = data[payload_size:]
    msg_size = struct.unpack(">L", packed_msg_size)[0]
    print("msg_size: {}".format(msg_size))
    while len(data) < msg_size:
        data += conn.recv(4096)
    frame_data = data[:msg_size]
    data = data[msg_size:]
    frame=pickle.loads(frame_data, fix_imports=True, encoding="bytes")
    frame = cv2.imdecode(frame, cv2.IMREAD_COLOR)
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
```

```
    faces = faceCascade.detectMultiScale(gray,scaleFactor=1.1, minNeighbors=5, minSize=(30, 30), flags = cv2.CASCADE_SCALE_IMAGE)

    print("Found {0} faces!".format(len(faces)))

    for (x, y, w, h) in faces:

        cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2)

    cv2.imwrite("test.jpg",frame);

    img = cv2.imread("test.jpg")

    result, img = cv2.imencode('.jpg', img, encode_param)

    data = pickle.dumps(img, 0)

    size = len(data)

    conn.sendall(struct.pack(">L", size) + data)

    #cv2.imshow("Faces found", frame)

    #cv2.waitKey(0)
```

6.1.2 Implementing HoneyBee model

6.1.2.1 Import Libraries

```
import socket

import time

import sys

import tkinter

from tkinter import *

import math

import random

from threading import Thread

from collections import defaultdict

from tkinter import ttk

from tkinter import filedialog

from multiprocessing import Queue

import matplotlib.pyplot as plt

import cv2

import struct

import pickle

import zlib

import numpy as np

from PIL import Image
```

(where tkinter used for GUI(front end) , numpy - mathematical , matplotlib - data visualization ,socket - Sockets and the socket API are used to send messages across a

network, time – to represent time in code, sys – used to manipulate different parts of the Python runtime environment, math – used for mathematical calculations, random – used to generate the pseudo-random variables, Thread – used to implement different parts of your program run concurrently and can simplify your design, defaultdict – used to implement dictionary, Queue – use to implement multi-producer, multi-consumer queues, cv2 - used for image processing and performing computer vision tasks, struct - used to convert the native data types of Python into string of bytes and vice versa, pickle - Pickle is used for serializing and de-serializing Python object structures, zlib – used for compression and decompression of arbitrary data, whether it be a string, structured in-memory content, or files, Image – used for loading image, preprocessing image etc.)

6.1.2.2 Defining the global function

```
global mobile

global labels

global mobile_x

global mobile_y

global text, text1

global canvas

global mobile_list

global filename

encode_param = [int(cv2.IMWRITE_JPEG_QUALITY), 90]

queue = Queue()

click = False

shownOutput = False

files = []

global local_time

global offload_time

global extension_time
```

6.1.2.3 Defining the Calculate Distance function

```
def calculateDistance(iot_x,iot_y,x1,y1):
    flag = False
    for i in range(len(iot_x)):
        dist = math.sqrt((iot_x[i] - x1)**2 + (iot_y[i] - y1)**2)
        if dist < 60:
            flag = True
            break
    return flag
```

6.1.2.4 Defining the Start Mobile Stimulation function

```
def startMobileSimulation(mobile_x,mobile_y,canvas,text,mobile,labels):
    class SimulationThread(Thread):
        def __init__(self,mobile_x,mobile_y,canvas,text,mobile,labels):
            Thread.__init__(self)
            self.mobile_x = mobile_x
            self.mobile_y = mobile_y
            self.canvas = canvas
            self.text = text
            self.mobile = mobile
            self.labels = labels
        def run(self):
            while(True):
```

```
        for i in range(len(mobile_x)):
            x = random.randint(10, 450)
            y = random.randint(50, 600)
            flag = calculateDistance(mobile_x,mobile_y,x,y)
            if flag == False:
                mobile_x[i] = x
                mobile_y[i] = y
                canvas.delete(mobile[i])
                canvas.delete(labels[i])
                name = canvas.create_oval(x,y,x+40,y+40, fill="blue")
                lbl = canvas.create_text(x+20,y-10,fill="darkblue",font="Times 10
italic bold",text="Mobile "+str(i))
                labels[i] = lbl
                mobile[i] = name
        canvas.update()
        offload()
        if click == True and filename not in files:
            print(filename)
            #queue.put(filename)
            files.append(filename)
        time.sleep(4)
    newthread = SimulationThread(mobile_x,mobile_y,canvas,text,mobile,labels)
    newthread.start()
```

6.1.2.5 Defining generate function

```
def generate():
    global mobile
    global labels
    global mobile_x
    global mobile_y
    mobile = []
    mobile_x = []
    mobile_y = []
    labels = []


    for i in range(0,20):
        run = True
        while run == True:
            x = random.randint(10, 450)
            y = random.randint(50, 600)
            flag = calculateDistance(mobile_x,mobile_y,x,y)
            if flag == False:
                mobile_x.append(x)
                mobile_y.append(y)
                run = False
                name = canvas.create_oval(x,y,x+40,y+40, fill="blue")
```

```
            lbl = canvas.create_text(x+20,y-10,fill="darkblue",font="Times 10 italic
bold",text="Mobile "+str(i))
            labels.append(lbl)
            mobile.append(name)
    startMobileSimulation(mobile_x,mobile_y,canvas,text,mobile,labels)
```

6.1.2.6 Define Offload Thread Function

```
def offloadThread():
    global shownOutput
    global offload_time
    class OffloadThread(Thread):

def __init__(self):
        Thread.__init__(self)

def run(self):
        global shownOutput
        global offload_time
        while not queue.empty():
            if shownOutput:
                print(str(shownOutput))
                filename = queue.get()
```

```
img = cv2.imread(filename)
result, img = cv2.imencode('.jpg', img, encode_param)
mid = int(mobile_list.get())
worker = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
worker.connect(('localhost', 8485))
data = pickle.dumps(img, 0)
size = len(data)
worker.sendall(struct.pack(">L", size) + data)
data = b""
payload_size = struct.calcsize(">L")
while len(data) < payload_size:
    #print("Recv: {}".format(len(data)))
    data += worker.recv(4096)
packed_msg_size = data[:payload_size]
data = data[payload_size:]
msg_size = struct.unpack(">L", packed_msg_size)[0]
print("msg_size: {}".format(msg_size))
start = time.time()
while len(data) < msg_size:
    data += worker.recv(4096)
frame_data = data[:msg_size]
data = data[msg_size:]
frame=pickle.loads(frame_data, fix_imports=True, encoding="bytes")
```

```
                frame = cv2.imdecode(frame, cv2.IMREAD_COLOR)
                worker.close()
                end = time.time()
                click = False
                files.clear()
                offload_time = end - start
                text1.insert(END,"Mobile Task Offloading execution time : "+str(offload_time)+"\n")
                print("Mobile Task Offloading execution time : "+str(offload_time))
                shownOutput = False
                cv2.imshow("Detected Faces ", frame)
                cv2.waitKey(0)


 if shownOutput:
        newthread = OffloadThread()
        newthread.start()
```

6.1.2.7 Define Offload Task

```
def offloadTask():
    global filename
    global click
    global shownOutput
    files.clear()
    filename = filedialog.askopenfilename(initialdir="images")
    queue.put(filename)
    click = True
    shownOutput = True
    offload()
```

6.1.2.8 Define Offload

```
def offload():
    global shownOutput
    global offload_time
    text.delete('1.0', END)
    offload_1 = 0
    offload_2 = 0
    selected = int(mobile_list.get())
    x = mobile_x[selected]
    y = mobile_y[selected]
```

```
    canvas.delete(labels[selected])
    lbl = canvas.create_text(x+20,y-10,fill="red",font="Times 10 italic bold",text="Mobile "+str(selected))
    labels[selected] = lbl
    distance = 300
    for i in range(len(mobile_x)):
        if i != selected:
            x1 = mobile_x[i]
            y1 = mobile_y[i]
            dist = math.sqrt((x - x1)**2 + (y - y1)**2)
            if dist < distance:
                offload_2 = offload_1
                offload_1 = i
                distance = dist
                text.insert(END,"Mobile "+str(i)+" is in proximity of source "+str(selected)+"\n")
    text.insert(END,"\n\nCurrent Selected Mobile Worker to offload = "+str(offload_1))
    if click:
        offloadThread()
```

6.1.2.9 Define Local Run

```
def localRun():
    global local_time
    text1.delete('1.0', END)
    filename = filedialog.askopenfilename(initialdir="images")
    local_time = time.time()
    cascPath = "haarcascade_frontalface_default.xml"
    faceCascade = cv2.CascadeClassifier(cascPath)
    frame = cv2.imread(filename)
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    faces = faceCascade.detectMultiScale(gray,scaleFactor=1.1, minNeighbors=5, minSize=(30, 30), flags = cv2.CASCADE_SCALE_IMAGE)
    print("Found {0} faces!".format(len(faces)))
    for (x, y, w, h) in faces:
        cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2)
    local_time = time.time() - local_time
    text1.insert(END,"Local Task running time : "+str(local_time)+"\n")
    print("Local Task running time : "+str(local_time))
    cv2.imshow("Num Faces found ", frame)
    cv2.waitKey(0)
```

6.1.2.10 Defining Extension Offload Thread

```
def extensionoffloadThread():
    global shownOutput
    global extension_time
    class OffloadThread(Thread):


 def __init__(self):
          Thread.__init__(self)


     def run(self):
          global shownOutput
          global extension_time


         while not queue.empty():
              if shownOutput:
                  print(str(shownOutput))
                  filename = queue.get()
                  img = Image.open(filename)
                  img.save("test.jpg",optimize=True,quality=10)
                  img = cv2.imread("test.jpg")
                  result, img = cv2.imencode('.jpg', img, encode_param)
                  mid = int(mobile_list.get())
```

```
worker = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
worker.connect(('localhost', 8485))
data = pickle.dumps(img, 0)
size = len(data)
worker.sendall(struct.pack(">L", size) + data)
data = b""
payload_size = struct.calcsize(">L")
while len(data) < payload_size:
    #print("Recv: {}".format(len(data)))
    data += worker.recv(4096)
packed_msg_size = data[:payload_size]
data = data[payload_size:]
msg_size = struct.unpack(">L", packed_msg_size)[0]
print("msg_size: {}".format(msg_size))
start = time.time()
while len(data) < msg_size:
    data += worker.recv(4096)
frame_data = data[:msg_size]
data = data[msg_size:]
frame=pickle.loads(frame_data, fix_imports=True, encoding="bytes")
frame = cv2.imdecode(frame, cv2.IMREAD_COLOR)
worker.close()
end = time.time()
```

```
                click = False
                files.clear()
                extension_time = end - start
                text1.insert(END,"Mobile Extension Compress Task Offloading execution time : "+str(extension_time)+"\n")
                print("Mobile Extension Compress Task Offloading execution time : "+str(extension_time))
                shownOutput = False
                cv2.imshow("Detected Faces ", frame)
                cv2.waitKey(0)


 if shownOutput:
       newthread = OffloadThread()
       newthread.start()
```

6.1.2.11 Define Extension Offload Task

```
def extensionOffloadTask():
    global click
    global shownOutput
    global extension_time
    queue.put(filename)
    click = True
    shownOutput = True
    extensionoffloadThread()
```

6.1.2.12 Defining Graph Function

```
def graph():
    print(str(local_time)+" "+str(offload_time)+" "+str( extension_time))
    height = [local_time,offload_time, extension_time]
    bars = ('Local Task Execution Time', 'Offload Task Execution Time','Extension Compress Task Time')
    y_pos = np.arange(len(bars))
    plt.bar(y_pos, height)
    plt.xticks(y_pos, bars)
    plt.show()
```

6.1.2.13 Defining the button size and other configuration

```
global text
    global text1
    global canvas
    global mobile_list
    root = tkinter.Tk()
    root.geometry("1300x1200")
    root.title("Computing with Nearby Mobile Devices: a Work Sharing Algorithm for Mobile Edge-Clouds")
    root.resizable(True,True)
    font1 = ('times', 12, 'bold')


    canvas = Canvas(root, width = 800, height = 700)
    canvas.pack()
    l1 = Label(root, text='Mobile ID:')
    l1.config(font=font1)
    l1.place(x=850,y=10)


    mid = []
    for i in range(0,20):
        mid.append(str(i))
    mobile_list = ttk.Combobox(root,values=mid,postcommand=lambda: mobile_list.configure(values=mid))
```

```
    mobile_list.place(x=1000,y=10)
    mobile_list.current(0)
    mobile_list.config(font=font1)

    createButton = Button(root, text="Generate Mobile Devices", command=generate)
    createButton.place(x=850,y=60)
    createButton.config(font=font1)

    localButton = Button(root, text="Local Run Task", command=localRun)
    localButton.place(x=850,y=110)
    localButton.config(font=font1)
    offloadButton = Button(root, text="Offload Task", command=offloadTask)
    offloadButton.place(x=1000,y=110)
    offloadButton.config(font=font1)

    extensionButton = Button(root, text="Extension Compress & Offload Task",
command=extensionOffloadTask)
    extensionButton.place(x=850,y=160)
    extensionButton.config(font=font1)

    graphButton = Button(root, text="Task Execution Time Graph", command=graph)
    graphButton.place(x=1130,y=160)
    graphButton.config(font=font1)
```

```
text=Text(root,height=15,width=60)

scroll=Scrollbar(text)

text.configure(yscrollcommand=scroll.set)

text.place(x=750,y=200)


text1=Text(root,height=10,width=70)

scroll1=Scrollbar(text1)

text1.configure(yscrollcommand=scroll1.set)

text1.place(x=750,y=490)


root.mainloop()
```

CHAPTER 7
SYSTEM TESTING

7.1. TESTING STRATERGIES

7.1.1 Unit Testing

Unit testing, a testing technique using which individual modules are tested to determine if there are issues by the developer himself, it is concerned with functional correctness of the standalone modules. The main aim is to isolate each unit of the system to identify, analyze and fix the defects.

Unit Testing Techniques:

Black Box Testing - Using which the user interface, input and output are tested.

White Box Testing –Used to test each one of those functions behavior is tested.

7.1.2 Functional Testing

Functional testing systematic demonstrations that functions tested are available are specified by the business and technical requirements, system documentation, and user manuals.

7.1.3 Integration Testing

Integration Testing done upon completion of unit testing, the units or modules are to be integrated which gives raise too integration testing. The purpose of integration testing is to verify the functional, performance, and reliability between the modules that are integrated.

7.1.4 Data Flow Testing

Data flow testing is a family of testing strategies based on selecting paths through the program's control flow in order to explore sequence of events related to the status of Variables or data object. Dataflow Testing focuses on the points at which variables receive and the points at which these values are used.

7.1.5 User Interface Testing

User interface testing, a testing technique used to identify the presence of defects is a product/software under test by Graphical User interface [GUI].

7.1.5 Black Box Testing

A testing technique that examines the functionality of an application without peering into its internal Structures or workings .

7.1.5 White Box Testing

A testing technique used to test internal structures or workings of an application.

7.2 TEST CASES:

TEST CASE No.	INPUT	OUTPUT	DESCRIPTION	RESULT
Test Case 1 (Unit Testing) – Local Run	Image	No. of faces detected and response time	Using xml file, faces are detected from inputted image and response time is displayed on application interface	SUCCESS
Test Case 2 (Functional Testing) – Offload Task	Image	No. of faces detected and response time	Using xml file, faces are detected from inputted image by connecting to nearby mobile devices and response time is displayed on application interface	SUCCESS
Test Case 3 (Functional Testing)- Extension Offload Task	Image	No. of faces detected and response time	Using xml file, faces are detected from inputted image by connecting to nearby mobile devices and PIL python library response time is displayed on application interface	SUCCESS
Test Case 4 (White Box Testing) – Task Execution Time Graph	Response Time	Comparison graph	Compares response time that is updated by LocalRun, OffloadTask, ExtensionOffloadTask functions and plots a bar graph using matplotlib python library.	SUCCESS
Test Case 5 (Black Box Testing)	Nearby Mobile Devices	Assigning Nearby Mobile Id	Uses “queue” library and insert the nearby mobile devices in queue in ascending order of their distance from Mobile Id.	SUCCESS
Test Case 6 (Integration Testing - 1)	Mobile Id	Nearby Mobile Devices	Uses the selected Mobile Id and finds the nearby mobile devices using “math” library.	SUCCESS
Test Case 7 (Integration Testing - 2)	Image	Accurate result of Response Time	Calculates the response time by following different approaches using “time” library.	SUCCESS
Test Case 8 (User Interface Testing - 1)	Mobile Id	Nearby Mobile Devices	Uses “tkinter” library and takes the mobile id as input and uses “Thread” library and generates nearby mobile devices	SUCCESS

Test Case 9 (User Interface Testing -2)	Image	No. of faces detected	Uses “cv2” and “image” libraries and performs image processing and preprocessing.	SUCCESS
Test Case 10 (Data Flow Testing - 1)	Nearby Mobile Devices	Message Sharing	Using “socket” library, nearby mobile devices share the message and perform the task	SUCCESS

Table 7.1 Test Cases

7.3 RESULTS AND DISCUSSIONS (SCREEN SHOTS)

To run project initially double click on workers.bat and then double click on 'run.bat' file to get below screen

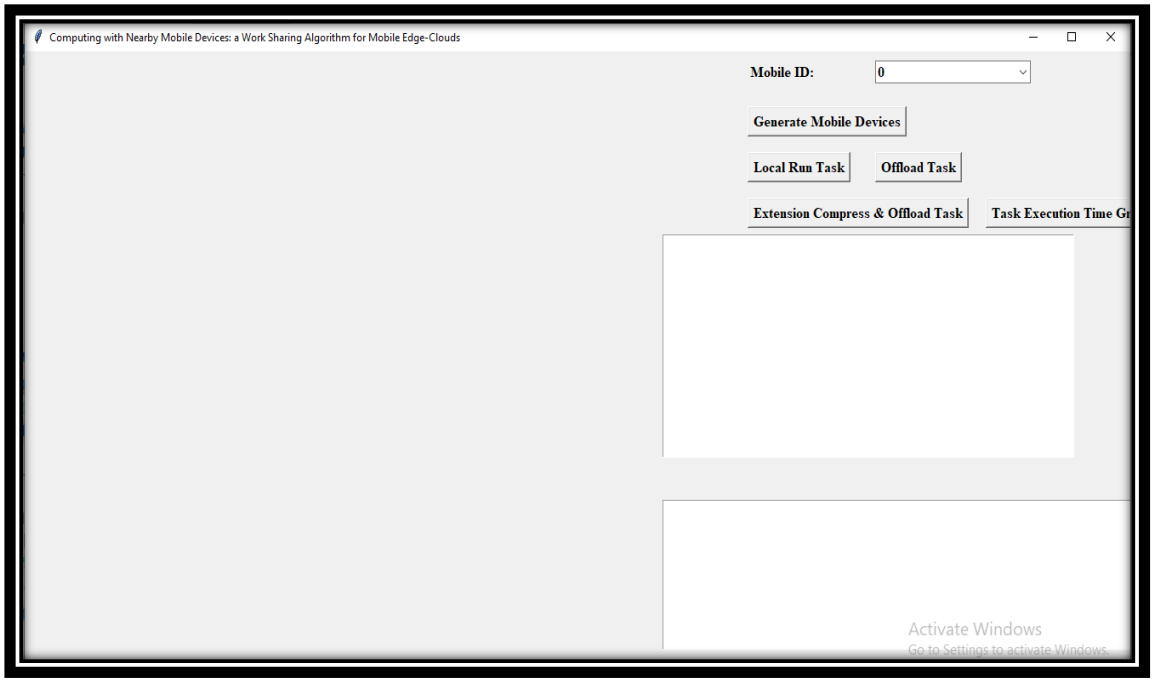

Fig 7.1 MAIN OUTPUT WINDOW

In above screen, select Mobile ID and click on Generate Mobile Devices

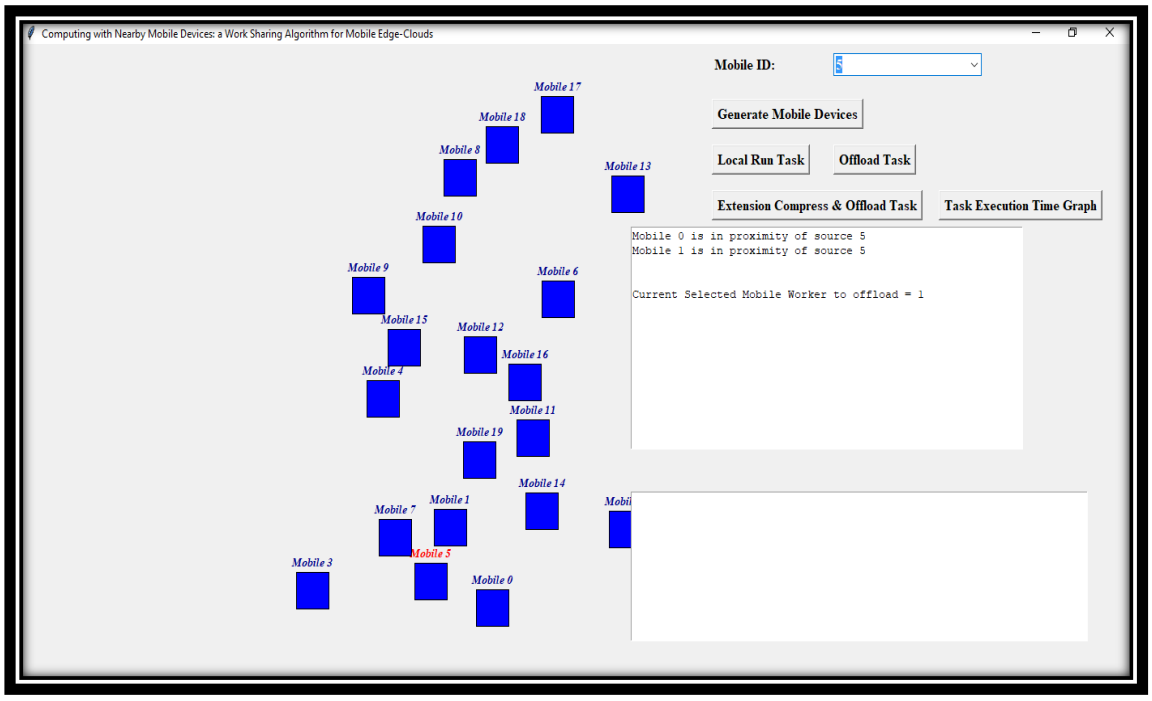

Fig 7.2 GENARATING NEARBY MOBILE DEVICES

Above screen represents the selected Mobile Id in "RED" and other mobile devise in "BLUE", and also nearby mobile devices are shown.

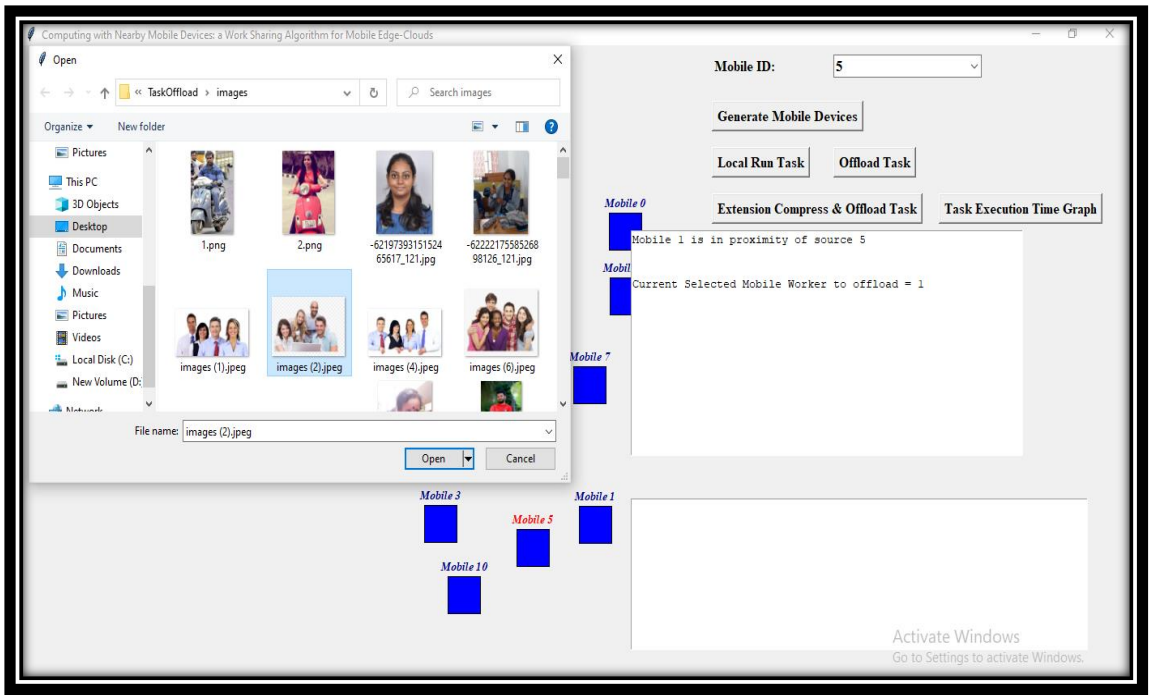

Fig 7.3 LOCAL RUN TASK - 1

In above screen, Click on Local Run Task and select an image and click on “open” to perform Local Run Task

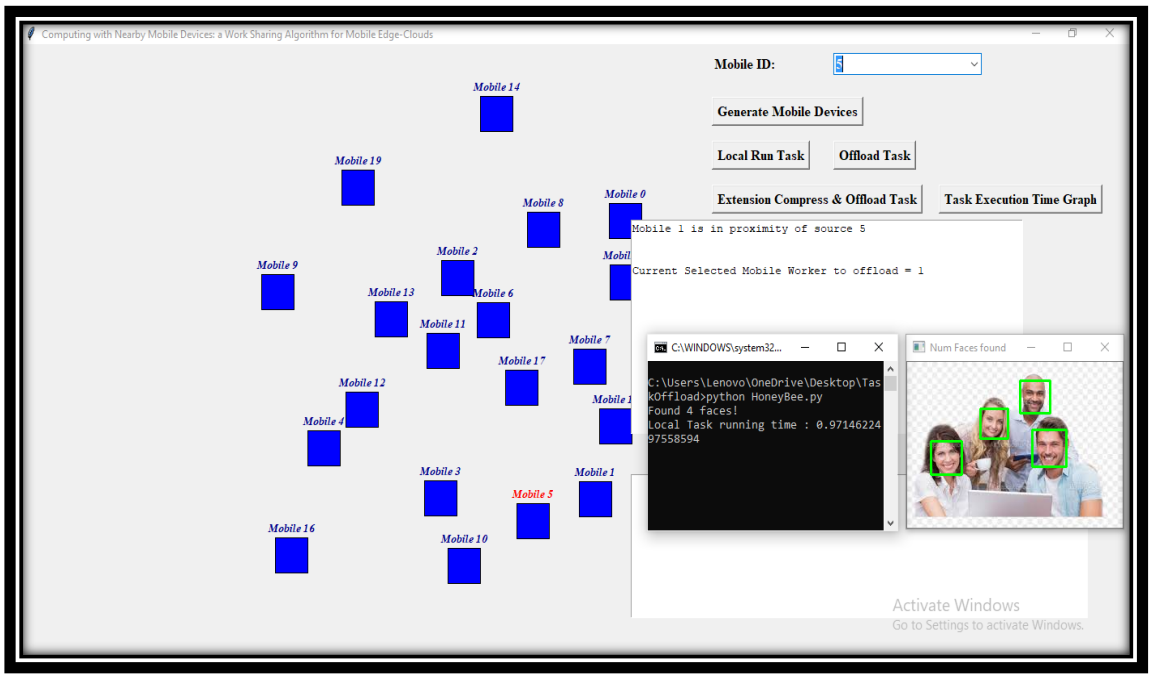

Fig 7.4 LOCAL RUN TASK - 2

Above screen represents the output of Local Run Task – No. of faces detected in image, Time consumed to perform the task.

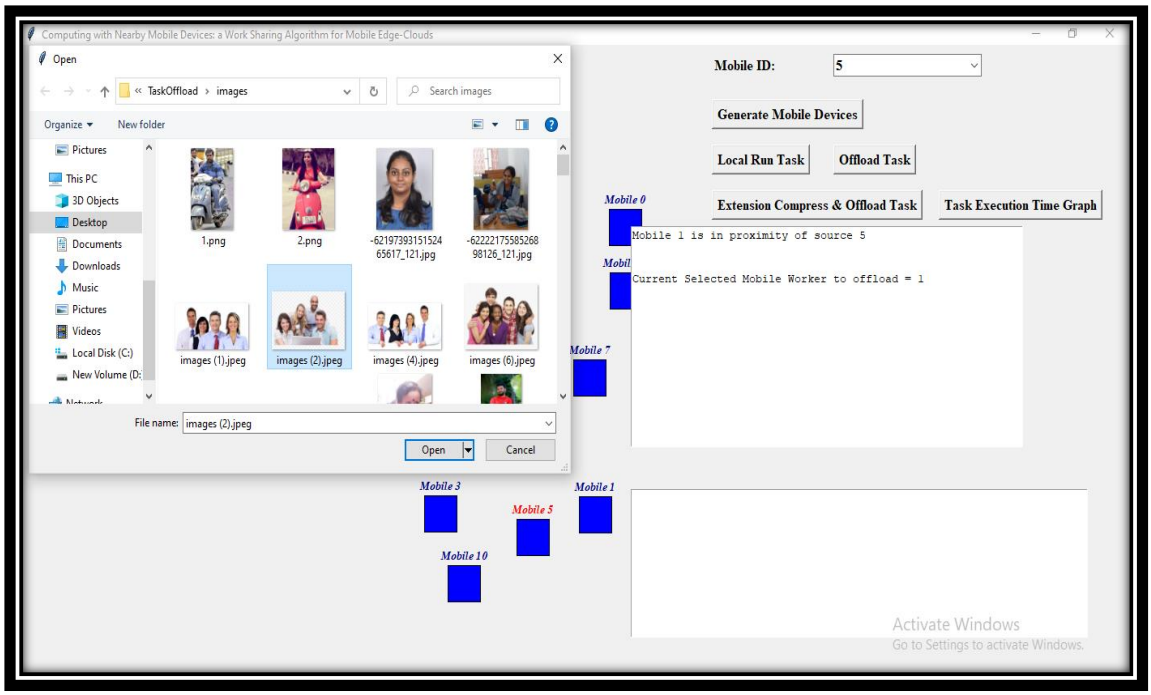

Fig 7.5 OFFLOAD TASK – 1

In above screen, Click on Offload Task and select an image and click on "open" to perform Offload Task

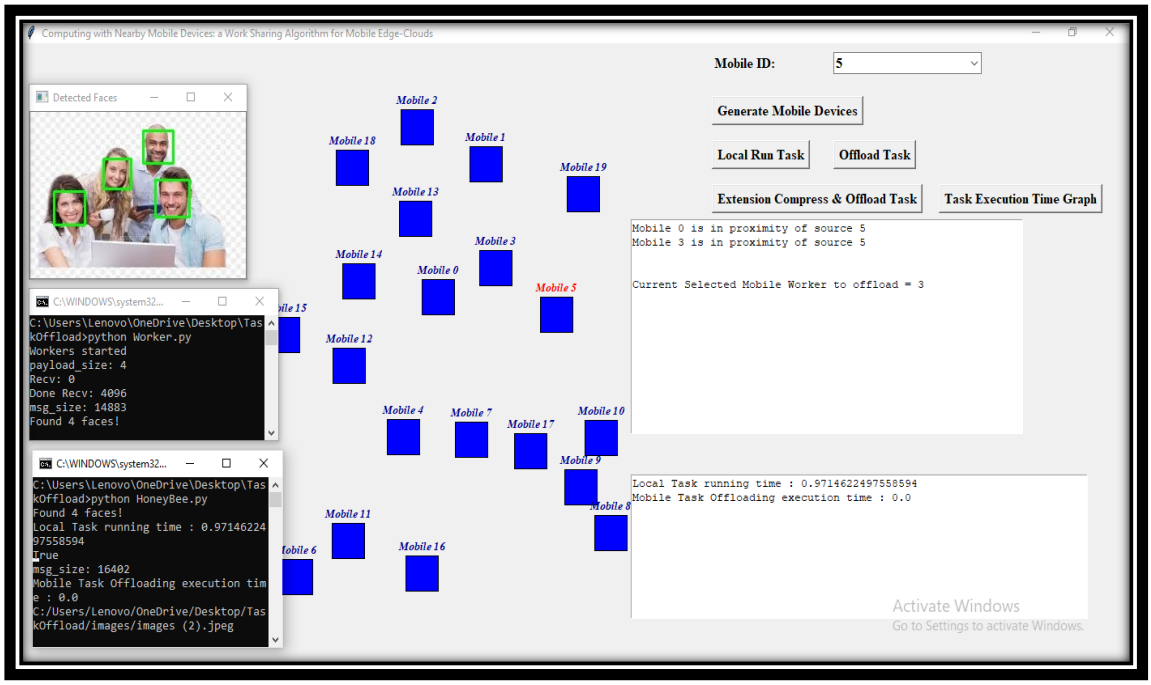

Fig 7.6 OFFLOAD TASK – 2

Above screen represents the output of Local Run Task – No. of faces detected in image, Time consumed to perform the task.

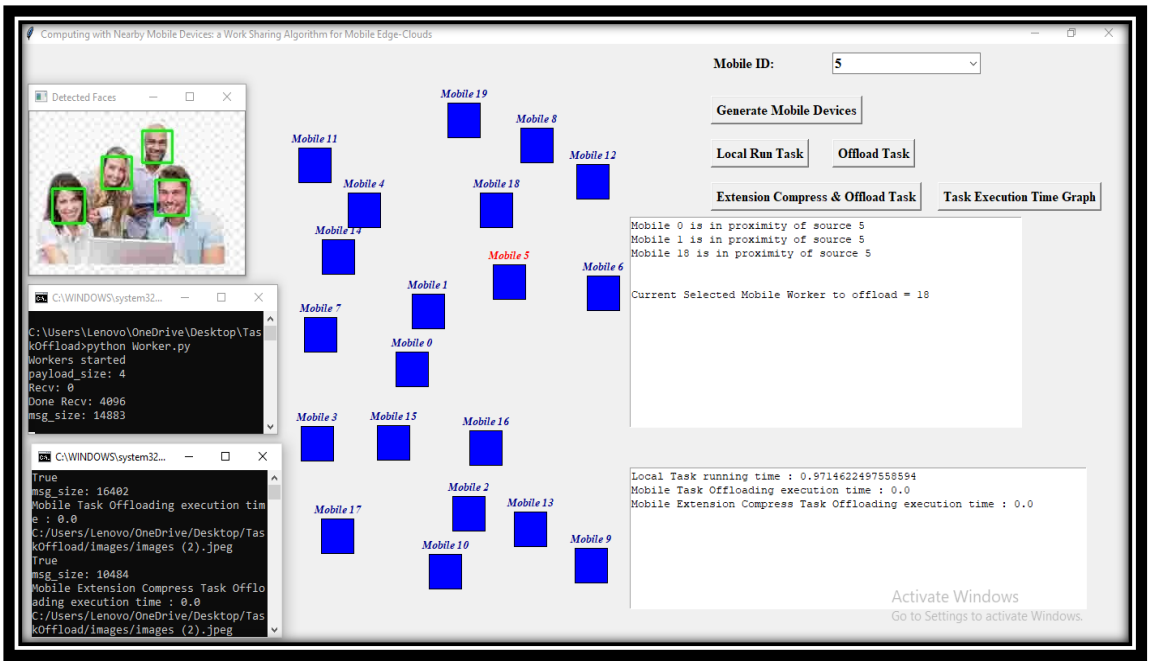

Fig 7.7 EXTENSION OFFLOAD TASK

In above screen click on Extension Offload Task and the output of Extension Offload Task , i.e Time consumed to perform the task, No. of faces detected, is prompted as above

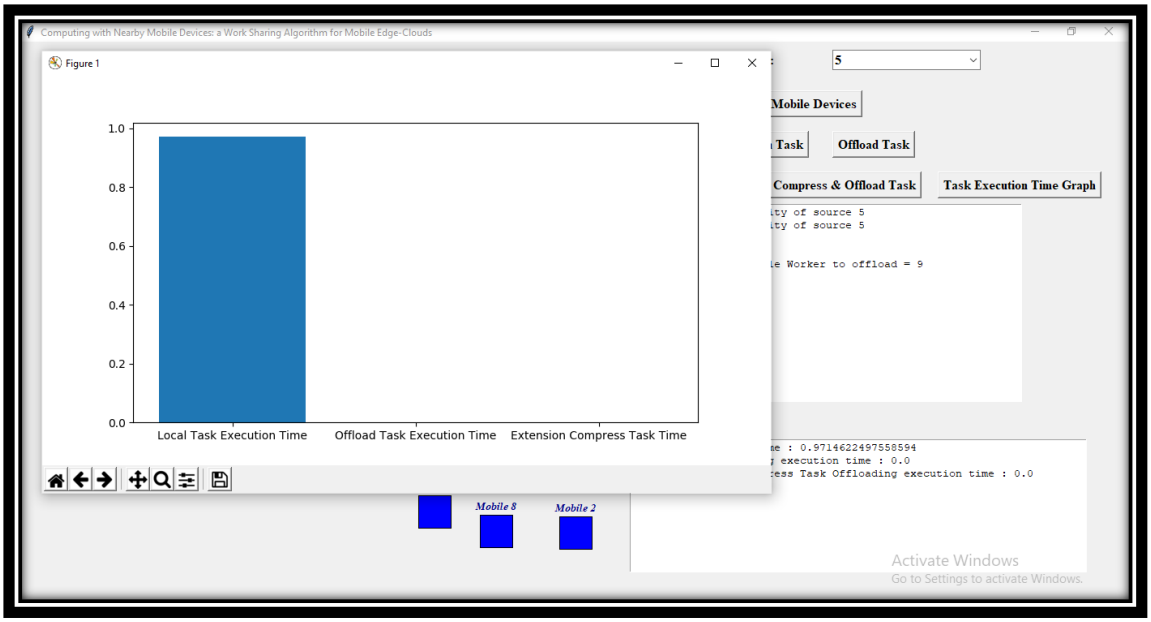

Fig 7.8 TASK EXECUTION TIME GRAPH

In above screen click on task execution time graph, Graph is viewed representing X-axis as Different approaches to perform the task and Y-axis as Time consumed by following different approaches to perform the task.

CHAPTER 8
CONCLUSION AND FUTURE ENHANCEMENTS

8.1 CONCLUSION:

We present the following conclusions.

- Firstly, work sharing among an autonomous local mobile device crowd is a viable method to achieve speedups and save energy. The addition of new resources up to an optimal amount, can yield increased speedups and power savings.
- Secondly, a generalized framework can be used for abstracting methods and enabling parameterization for different types of tasks made of independent jobs.
- Thirdly, inherent challenges of mobile computing such as random disconnections, having no prior information on participating nodes, and frequent fluctuations in resource availability can be successfully accommodated via fault tolerance methods and work stealing mechanisms.

The Honeybee model caters to tasks that can be decomposed into independent jobs. Many crowd computing tasks for mobile devices are suited to this model, for e.g., video transcribing ,language translation, medical data analysis face detection and mathematical demonstrations.

However, there are other tasks that cannot be easily decomposed into independent jobs. Work done by Agrawal et al. shows that work stealing can be further enhanced for dependent jobs and we aim to work in this area in the future. Incentive management and security are important for the deployment of successful mobile crowd applications. However, designing a comprehensive and realistic incentive scheme for mobile crowd computing applications requires further research in collaboration with policy, legal and economics scholars, as does providing security and trust mechanisms.

As the main focus of this paper was performance gain and energy conservation, these two areas were out of scope. For this work, we have built Honeybee with the assumption that an incentive system and a secure environment are already in place. Future work could be possible in designing a secure platform for mobile crowd computing applications, sup- porting incentive management. Moreover, this work focused on the evaluation of machine-centric computation. How-ever, as discussed, applications that employ human intelligence are also feasible using the Honeybee model. For example, the face detection app can be modified so that human intelligence is used to identify the faces detected by the machine.

8.2 Future Enhancement:

We aim to extend our evaluations to focus on this aspect, using additional criteria such as accuracy and usability in our future work. Further- more, as observed in our experiments, the performance gain plateaus as the number of worker nodes increase due to the additional costs that occur when a single device (delegator as P2P group owner) maintains multiple connections. To overcome this and scale up, we plan to extend Honeybee to support other topologies and initial experiments in [39], where an early version of the Honeybee model was ex- tended to support hierarchical Bluetooth connections, show consistent speedups using a linear topology, with an inter- mediate node functioning both as a worker and a delegator. For this approach, a combination of Bluetooth and Wi-Fi Direct in alternate hierarchical layers can be explored as Wi-Fi direct does not support multiple Wi-Fi direct groups. We also plan to experiment with latest D2D technologies such as LTE-Direct to improve performance. In addition, the experiments in this paper were performed in a controlled setting. We plan to extend these tests to more realistic scenarios by using mobility patterns to simulate churn.

REFERENCES

JOURNALS:

[1]. Cisco visual networking index: Global mobile data traffic forecast update. http://www.cisco.com/c/en/us/solutions/collateral/service-provider/ip-ngn-ip-next-generation-network/white paper c11-520862.html

[2].DARPA Creates Cloud Using Smartphones. http://www.informationweek.com/mobile/darpa-creates-cloud-using-smartphones/d/d-id/1111323.

[3]. The hyrax project. http://hyrax.dcc.fc.up.pt/.

[4]. K. Agrawal, C.E. Leiserson, and J. Sukha. Executing task graphs using work-stealing. In Parallel Distributed Processing (IPDPS), 2010 IEEE International Symposium on, pages 1–12, April 2010. [5]. M. S. Bernstein. Crowd-powered systems. KI - Ku¨nstliche Intelli- genz, 27(1):69–73, 2013.

[6]. K. Bhardwaj, S. Sreepathy, A. Gavrilovska, and K. Schwan. Ecc: Edge cloud composites. In Proceedings of 2nd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering, pages 38–47, 2014.

[7].Bluetooth. Specification of the bluetooth system version 4.1. https://www.bluetooth.org/DocMan/handlers/DownloadDoc. ashx?doc id=282159, December 2013. Accessed: 25/06/2014.

[8]. R. Blumofe and C. Leiserson. Scheduling multithreaded computa- tions by work stealing. J. ACM, 46(5):720–748, 1999.

[9]. R. D. Blumofe, C. F. Joerg, B. C. Kuszmaul, C. E. Leiserson, K. H. Randall, and Y. Zhou. Cilk: an efficient multithreaded runtime system. SIGPLAN Not., 30:207–216, August 1995.

[10]. D. Camps-Mur, A. Garcia-Saavedra, and P. Serrano. Device-to- device communications with wi-fi direct: overview and experi- mentation. Wireless Communications, IEEE, 20(3):96–104, June 2013.

[11]. Z. Chen, R. Fu, Z. Zhao, Z. Liu, L. Xia, L. Chen, P. Cheng, C. C. Cao, Y. Tong, and C. J. Zhang. gMission: a general spatial crowdsourcing platform. Proceedings of the VLDB Endowment, 7(13):1629–1632, 2014.

[12]. B. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti. Clonecloud: elastic execution between mobile device and cloud. In Proc. of the 6th conference on Computer systems, EuroSys, pages 301–314, 2011.

[13]. B. Chun and P. Maniatis. Dynamically partitioning applications between weak devices and clouds. In Proc. of the 1st ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond, MCS, pages 71–75, New York, USA, 2010. ACM.

[14]. E. Cuervo, A. Balasubramanian, Dae-ki Cho, A. Wolman, S. Saroiu,R. Chandra, and P. Bahl. Maui: making smartphones last longer with code offload. In Proc. of the 8th Intl conference on Mobile systems, applications, and services, MobiSys, pages 49–62, New York, USA, 2010. ACM.

[15]. L. Deboosere, P. Simoens, J. De Wachter, B. Vankeirsbilck, F. De Turck, B. Dhoedt, and P. Demeester. Grid design for mobile thin client computing. Future Generation Computer Systems, 27(6):681 – 693, 2011.

[16]. J. Dinan, D. B. Larkins, P. Sadayappan, S. Krishnamoorthy, and J. Nieplocha. Scalable work stealing. In Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, SC ’09, pages 53:1–53:11, NY, USA, 2009. ACM.

[17]. H. T. Dinh, C. Lee, D. Niyato, and P. Wang. A survey of mo- bile cloud computing: architecture, applications, and approaches. Wireless Communications and Mobile Computing, 2011.

[18]. D. C. Doolan, S. Tabirca, and L. T. Yang. Mobile parallel comput- ing. In Proc. of the 5th Int’l Symposium on Parallel and Distributed Computing, pages 161–167, 2006.

[19]. D. C. Doolan, S. Tabirca, and L. T. Yang. MMPI a message passing interface for the mobile environment. In Proceedings of the 6th International Conference on Advances in Mobile Computing and Multimedia, MoMM ’08, pages 317–321, NY, USA, 2008. ACM.

TEXTBOOKS:

- Programming Python, Mark Lutz
- Head First Python, Paul Barry
- Core Python Programming,R. Nageswara Rao
- Learning with Python, Allen B. Downey

WEBSITES:

1. https://www.w3schools.com/python/
2. https://www.tutorialspoint.com/python/index.htm
3. https://www.javatpoint.com/python-tutorial
4. https://www.learnpython.org/
5. https://www.pythontutorial.net/