THE EFFECT OF PARALLEL EXECUTION ON MULTI-SITE COMPUTATION OFFLOADING IN MOBILE CLOUD COMPUTING

by

Muhammad Ismail Sheikh
B.A.Sc., Ryerson University, 2015

A thesis presented to Ryerson University in partial fulfillment of the
requirements for the degree of Master of Applied Science
in the program of Electrical and Computer Engineering
Toronto, Ontario, Canada, 2018

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I authorize Ryerson University to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize Ryerson University to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

I understand that my thesis may be made electronically available to the public.

THE EFFECT OF PARALLEL EXECUTION ON MULTI-SITE COMPUTATION OFFLOADING IN MOBILE CLOUD COMPUTING

Muhammad Ismail Sheikh
Master of Applied Science
Department of Electrical and Computer Engineering
Ryerson University, 2018

Abstract

The demand for running complex applications on smart mobile devices is rapidly increasing. However, the limitations of resources are restricting the development of intensive applications on these devices. The restrictions can be overcome by offloading the computation of an application in the powerful cloud servers. The objective of the computation offloading is to offload the parts of an application to the cloud server to minimize the response time, energy consumption and monetary cost of the application. Unlike prior work in computation offloading, this work considers the effect of parallel execution—on different devices (external parallelism) and on the different cores of a single device (internal parallelism). This work models each device as a multi-server queueing station. It uses genetic algorithm to determine the near-optimal offloading allocation. The results show that considering the effect of parallel execution yields better pareto-optimal solution for the allocation problem compared to excluding parallelism.

Acknowledgments

I would like to thank my supervisor, Dr. Olivia Das for her guidance, encouragement, and her financial support. I greatly appreciate the opportunity she provided me to carry out the research under her supervision. I would like to express my sincerest gratitude for her continuous constructive feedback on the challenges and problems I faced during this Research. It was my pleasure to work under her supervision to contribute my part in the area of mobile cloud computing.

I would also like to thank the thesis defense committee members, Dr. Anpalagan, Dr. Yang, Dr. Jaseemuddin, and Dr. Das for their time and effort to review my thesis and to provide constructive feedback.

I also want to thank my parents, my wife and other family members for their guidance, financial and emotional support throughout my academic and professional career. Their continuous prayers, and blessings cannot be explained in words which shaped my personality and helped me face all the challenges in my life.

Last but not least, I would like to thank my Shaykh, Hazrat Khawaja Abdullah Jan Sahib, who motivated me to continue my studies after I moved to Canada in 2007. He guided me with his blessings and prayers on all the challenges I faced in my life.

Table of Contents

Author's Declaration ii

Abstract iii

Acknowledgements iv

List of Tables ix

List of Figures xi

1. Introduction 1

1.1. Motivation 1

1.2. Research Problem 3

1.3. Contribution 4

1.4. Research Overview 5

1.5. Thesis Outline 6

2. Background 7

2.1. Mobile Cloud Computing 7

2.2. Single-Site Offloading Framework 7

2.3. Multi-Site Offloading Framework 9

2.4. Contribution to the Literature 10

2.5. Genetic Algorithm 11

2.5.1. Initialization 11

2.5.2. Selection 12

2.5.3. Crossover 12

2.5.4. Mutation 12

2.5.5. GA Objectives 13

2.5.6. Fitness Function 13

3. The Pareto-Optimal Solution 14

3.1. Definition and Computing Parameters 15

3.2. Model without Considering Parallel Execution of Tasks 19

3.2.1. Response Time 19

3.2.2. Energy Consumption 19

3.2.3. Execution Cost 21

3.3. Introductory Example 1 22

3.4. Evaluating a given allocation without Considering Parallel Execution of Tasks (for the introductory example) 26

3.4.1. Response Time 26

3.4.2. Energy Consumption 27

3.4.3. Monetary Cost 29

3.5. Near-Optimal Allocation(s) using Genetic Algorithm without Considering Parallel Execution (for the Introductory Example) 30

3.5.1. GA Parameters 30

3.5.2. No Offloading 31

3.5.3. Single-Site Offloading 32

3.5.4. Multi-Site Offloading 32

3.6. Our Model considering Parallel Execution of Tasks 33

3.6.1. Definition 34

3.6.2. Job Generation 36

3.6.3. External Parallel Execution 38

3.7. Evaluating a given allocation Considering Only External Parallelism (for the introductory example) 39

3.8. Near-Optimal Allocation(s) using Genetic Algorithm Considering Only External

Parallelism (for the Introductory Example) 43
3.8.1. No Offloading 43
3.8.2. Single Site Offloading 44
3.8.3. Multi-Site Offloading 45
3.9. Evaluating a given allocation Considering both Internal and External Parallelism (for the introductory example) 46
3.10. Near-Optimal Allocation(s) using Genetic Algorithm Considering both Internal and External Parallelism (for the Introductory Example) 50
3.10.1. No Offloading 51
3.10.2. Single Site Offloading 51
3.10.3. Multi-Site Offloading 53
3.11. Summary 54

4. Case Study 56

4.1. Mobile Application Specification 56
4.2. Model Specification 59
4.2.1. Mobile Device 59
4.2.2. Mobile User Profile 60
4.2.3. Cloud Server d_1 60
4.2.4. Cloud Server d_2 60
4.2.5. Device to Device Bandwidth 61
4.2.6. Genetic Algorithm Configuration 61
4.2.7. System Configuration 62
4.3. Results and Discussions 62
4.3.1. Including or excluding parallel execution to find the near-optimal offloading Allocation 64
4.3.1.1. Response Time 64

4.3.1.2. Energy Consumption 66
4.3.1.3. Monetary Cost 67
4.3.2. Evaluating the effect of multi-core devices on near-optimal offloading allocation 69
4.3.2.1. Case1: No Offloading 70
4.3.2.2. Case 2: Single Site offloading 72
4.3.2.2.1. 1-Core each Resource 73
4.3.2.2.2. 2-Core in each Resource 74
4.3.2.2.3. 4-Core in each Resource 75
4.3.2.3. Case 3: Multi-Site Offloading 75
4.3.2.3.1. 1-Core each Resource 77
4.3.2.3.2. 2-Core each Resource 77
4.3.2.3.3. 4-Core each Resource 78
4.3.3. Summary 78

5. Conclusion 80

5.1. Conclusion 80
5.2. Future Work 81

References 83

List of Tables

Table 1: GA with no offloading 31

Table 2: GA Offloading with one VM 32

Table 3:GA Offloading with two VM 33

Table 4: Jobs scheduled in the Mobile Device d_0 39

Table 5: Jobs scheduled in the Cloud Server d_1 40

Table 6: Jobs scheduled in the Cloud Server d_2 41

Table 7: External Parallel Execution Gain 42

Table 8: External Parallel Execution with no offloading 43

Table 9: External Parallel Execution with single-site offloading (one VM) 44

Table 10: External Parallel Execution with multi-site offloading (two VMs) 45

Table 11: Jobs scheduled in the Mobile Device d_0 46

Table 12: Jobs scheduled in the Cloud Server d_1 47

Table 13: Jobs scheduled in the Cloud Server d_2 48

Table 14: External and Internal Parallel Execution Gain 49

Table 15: Internal and External Parallel Execution with no offloading 51

Table 16: Internal and External Parallel Execution for single-site offloading (with one VM) 52

Table 17: Internal and External Parallel Execution for multi-site offloading (with two VMs) 53

Table 18: Summary of Results for the Introductory Example 55

Table 19: Near-optimal Solution and Minimum Corresponding Response Time 64

Table 20: Near-optimal Solution for Energy Consumption 66

Table 21: Cost Objective with respect to Response Time 68

Table 22: Near-optimal Energy Consumption with Cost 69

Table 23:Offloading Allocation for No-Offloading 71

Table 24:Offloading Allocation for Single-Site Offloading 73

Table 25: Offloading Allocation for Multi-Site Offloading 76

Table 26: Effect Of The Number Of Cores In Each Device On Near-Optimal Offloading Allocation 79

List of Figures

Figure 1: A Simple Workflow Graph 22

Figure 2: The time-weighted workflow graph corresponding to the offloading allocation a. 25

Figure 3: The device du modeled as a multi-server queueing station with ru number of servers 34

Figure 4: Call graph of the face recognition application 56

Figure 5: Workflow - Graph of a Face Recognition Application 57

Figure 6: Simplified Work-Flow graph of Face Recognition Application 58

Figure 7: Internal Parallel Execution for No-Offloading 71

Figure 8: Internal Parallel Execution for Single-Site Offloading 73

Figure 9: Internal Parallel Execution for Multi-Site Offloading 76

Chapter 1: Introduction

1.1 Motivation

The demand of mobile devices is continuously increasing in our daily lives through their new impressive features such as face recognition, augmented reality and interactive gaming. However, these functionalities are offered through specific applications which are resource-hungry and demand intensive computation as well as high energy consumption. Further, the mobile devices are very resource constrained due to their physical size, limited processing speed, and battery life. These limitations cause excessive resistance in the development of these impressive applications. One promising approach to deal with the resource limitations of mobile devices is to use computation offloading. Computation offloading is a solution to improve the capability of mobile applications by migrating heavy computation tasks of an application to powerful servers in the cloud [18]. Computation offloading can save energy and prolong the battery life of mobile devices by running computation-intensive tasks in the cloud servers, which will drain a device's battery if executed locally. Computation offloading can improve the response time of the mobile application by running some tasks on the cloud servers (assuming that the processing speed of the cloud servers is higher than the mobile device). However, there are some factors that adversely affect the efficiency of offloading such as, the amount of data that must be transferred among the mobile device and the cloud servers, and the communication bandwidth between them. Computation offloading also incurs the following monetary cost for the mobile user:

(i) The user has to pay for the renting cost of the cloud servers for the duration of the application execution

(ii) In case of excessive data exchange between the mobile device and cloud servers, the user may have to pay for the additional data usage if it exceeds the monthly subscription.

Thus, a mobile device should judiciously determine whether to offload computation, what tasks (i.e. parts) of an application should be offloaded, and to which servers in the cloud. Further, the code offloading can be deployed either by offloading any method, any thread or any class of an application to the cloud server. The greatest benefit from computation offloading can be obtained by finding the optimal allocation for the tasks of an application to different devices (i.e. the mobile device and the cloud servers) that minimizes the application objective. The objectives can be the total response time, the mobile battery energy required for the computation, or the monetary cost incurred by the user for the execution on the cloud servers. The workflow (the execution sequence of tasks) of a mobile application may not be linear, i.e. it may contain tasks that can execute in parallel in multiple different resources. The computation offloading can be further enhanced due to this non-linear property of the workflow, by adopting parallelism in the task execution. It can be implemented among the different computation resources (i.e. mobile device and cloud servers), referred to as *external parallelism* or to different processing cores of a single device, referred to as *internal parallelism* in this research. Both external and internal parallel execution can significantly improve the mobile application response time based on the offloading allocation. As a result, the energy consumption and the monetary cost needed to run the application will also be affected depending on the offloading allocation.

1.2 Research Problem

Mobile Cloud Computing (MCC) has dramatically improved from its initial term of cyber-forging and continuously getting substantial attention of researchers, investors, and analysts due to its ability to leverage execution of an applications from mobile device to powerful cloud server(s). There has been continuous research conducted in the area of MCC to make it more convenient and user-friendly to the end user. The current literature emphasizes on both, the single-site as well as multi-site code offloading, between the mobile device and cloud servers. However, to the best of our knowledge, the current research in the models of code offloading is still performing the computation of an application among different resources sequentially by adding the execution time of all parallel tasks. This assumption of sequential execution of the parallel tasks dramatically affects the prediction of overall response time and energy consumption of the mobile application.

Further, there are diverse types of mobile device users whose objectives, needs, and perceptions of the mobile applications are different. For example, some users are aggressive and their main concern is the performance, some are conservatives and their concern is the battery life of their mobile device and others are reluctant to spend any additional cost on the application execution. Thus, to make the code offloading more reliable and available to these diversified users, it must be multi-objective. To the best of our knowledge, the current literature focuses on the application response time and energy consumption for computation offloading. However, the monetary cost which arises by renting the cloud servers as well as the network service charges of the mobile device are ignored. Thus, to achieve a more accurate estimate of code offloading, the network charges and cloud service renting cost should also be considered. This research proposes a unique

multi-site code offloading model by considering external and internal parallel execution of the application tasks with the consideration of multiple objectives, i.e. the response time, the energy consumption and the monetary cost to provide a user with more realistic and near-optimal code offloading allocation.

1.3 Contributions

This thesis solved the problem of multisite offloading of mobile applications. Our work goes beyond existing approaches by considering parallel execution of tasks during offloading decision in contrast to others who primarily focused on sequential executions.

The contributions of this thesis are as follows:

1. It proposes a theoretical framework for the near-optimal offloading allocation problem in multi-site offloading scenario.
2. It uses genetic algorithm to find the near-optimal allocation of tasks to different devices. The genetic algorithm iteratively evaluates multiple allocations to find the near-optimal solution.
3. To evaluate an offloading allocation, we propose a new algorithm that computes the application's response time, the energy consumption on the mobile device, and the monetary cost. Our algorithm accounts for the execution dependencies of the tasks and the parallel execution of tasks across the cores of a device as well as across different devices.
4. We implement our novel algorithm that considers parallel execution of tasks, and an existing algorithm that ignores the parallel execution of tasks. We accomplish this

implementation using an existing library of genetic algorithms in Java (the MOEA Framework [15]).

5. We compare and analyze our novel algorithm against the existing algorithm for a real-world face recognition application. The results show that accounting for the effect of parallel execution yields better near-optimal solution for the allocation problem compared to not accounting for parallelism at all.

The results of this analysis are incorporated into a research manuscript and submitted to the 26th IEEE International Symposium on the Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS 2018) in Milwaukee, Wisconsin, USA [29].

1.4 Research Overview

The response time and energy consumption are the key elements in the performance and reliability of an application. The enhancement of these two factors can open a straight path of the development of the intensive applications such as face recognition and GPS services on mobile devices. The code offloading framework has the capability of dramatically improving these two factors at a tiny processing cost by leveraging the intensive execution from resource hungry mobile processor to the powerful cloud server. This research compares the enhancement of the response time and energy consumption of an application with the required additional processing cost as a multi-objective code offloading framework. The trade-off between the Response Time, the Energy Consumption and the monetary cost is further examined by introducing parallel execution amongst different available code offloading sites (VMs) as well as partitioning between different cores of the processors using Genetic Algorithm (GA). The GA finds the near-optimal values of the Response Time, the Energy Consumption and the monetary cost by examining the solution

population in all possible scenarios such as complete offloading to the cloud server(s), or performing total local execution (in the mobile device), or performing hybrid execution between the cloud server(s) and the mobile device.

1.5 Thesis Outline

This thesis consists of total six chapters. A brief description of each chapter is as follows:

Chapter 1:

The first chapter provides the introduction. It summarizes the motivation behind this research and provides a brief overview of the research.

Chapter 2:

This chapter provides a background of available code offloading frameworks, their solution to the research problem, along with limitations and areas of improvement on each framework. It also summarizes our contribution to the literature.

Chapter 3:

This chapter illustrates our multi-objective framework that accounts for the effects of internal and external parallelism on offloading allocation problem.

Chapter 4:

This chapter compares the effect of external and internal parallel execution with the sequential execution proposed in the current literature though a real-world face recognition application.

Chapter 5:

This chapter provides the conclusions and the future work.

Chapter 2: Background

Mobile devices are an inseparable part of our daily life and continuous research has been conducted to make them more user-friendly and intuitive by enabling new and updating existing features on them. Cloud Computing offers several different on-demand services such as Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS) to ease the resistance of the development of intensive application on mobile devices. [2] The utilization of these cloud services on a mobile device refers to Mobile Cloud Computing (MCC). [4], [22].

2.1 Mobile Cloud Computing

Mobile Cloud Computing (MCC) enables the execution of intensive applications such as face recognition and augmented reality on resource constrained mobile devices. The primary role of the MCC is to serve as a terminal between the resources-rich cloud server and resource constrained mobile device to improve the application execution time as well as to reduce the application energy consumption [19]. One of the method of creating a server-client bridge is called code offloading. There are several existing code offloading frameworks which one can use based on their objective such as improving response time, and reducing the energy consumption of a mobile application.

2.2 Single-Site Offloading Frameworks:

The MAUI [7] proposes an offloading framework based on the reduced energy consumption by using the integer linear programing to find the near-optimal offloading solution [14]. The MAUI framework provides method level code offloading and requires the developer to manually annotate the methods which can be offloaded to the cloud server. This framework maps the application as

a call graph where methods are represented by vertices and their invocation is represented by the edges [7].

The Clone-Cloud [6] provides a transparent code migration code offloading framework based on the energy consumption and execution time [12]. This framework uses a combination of static analysis and dynamic profiling to automatically partition the application and migrate the thread of the application to the cloud server. This framework converts the problem as a tree-diagram.

The ThinkAir [17] framework focuses on the scalability of cloud VM and dynamically scales the cloud server instances to allow parallel execution of offloading code on multiple instances [3]. As in MAUI, this framework also requires the application developer to manually annotate part of the code which can be offloaded to the cloud server. This framework contains an execution controller which determines the execution time, energy consumption and cost of offloading before generating offloading policy.

The COMET (Code Offloading by migrating execution transparently) provides a transparent code migration through distributed shared memory (DSM) between the mobile device and cloud server [13]. Similar to CloneCloud, this framework does not require manual annotation from the developer on the application code. It contains an automated code profiler to analyze the application for the offloading policy.

The framework in [11] dynamically partition the application by classify each task as offloadable or unoffloadable to minimize the response time and energy consumption. Their model constructs

the application as weighted consumption graph (WCG) to estimate the computational and communication cost and optimize it using min-cut offloading partitioning algorithm (MCOP).

2.3 Multi-Site Offloading Framework:

Multi-Site code offloading is a well-regarded approach for minimizing energy consumption of the mobile application. [26]. To the best of our knowledge, multi-site code offloading is considered by [24, 21, 14, 26].

The [24] multi-site code offloading framework assumes each cloud server has different computational capacities and network bandwidth. The application in this model is represented as a graph partitioning problem where nodes refer to computation module and edges refer to the interaction between modules. This model assigns weight to all nodes and edges to minimize the computation and communication cost using 0-1 Integer Linear programming (ILP) problem. The model is motivated by the data-centric offloading to provide solution to applications that requires multiple sources of data.

The [21] research developed an Energy Efficient Multisite Offloading (EMSO) algorithm by formulating partition problem as 0-1 Integer Linear programming (ILP). They perform object level offloading based on the constructed Weight Object Relation Graph (WORG) dynamic profiling. Using static analysis, weight is assigned to the nodes and edges of the graph to find the near-optimal offloading solution.

The energy-efficient multisite offloading policy [EMOP] in research [26] optimizes the application energy consumption using discrete time Markov chain (DTMC) model. It uses value iteration algorithm (VIA) to determine the offloading policy for the Markov chain model. Their model considers heterogeneity of offloading sites and perform data and process-centric offloading.

2.4 Contribution to the literature:

To the best of our knowledge, all single-site or multi-site frameworks mentioned in section 2.4 and section 2.5 uses a binary decision variable to decide whether a task of an application should be offloaded to the cloud server or be processed locally in the mobile device. However, our model introduces a multi-state decision variable to decide if the task should be offloaded to the cloud server or process locally in the mobile device. The states of the decision variable are equal to the number of computing resources available for the execution. Similar to the offloading framework of Sinha et al. [24], our model also allows each cloud server a different computational capacity and network bandwidth. The multi-state decision variable finds and offloads the application tasks to the cloud server using genetic algorithm.

In addition to that, all single-site and multi-site offloading frameworks focuses on the minimization of the response time and energy consumption. However, the monetary offloading cost which arises from the mobile data network and renting cloud servers are being ignored in the process. Our partitioning model optimizes the application based on three objectives: minimize the response time, minimize the energy consumption, and minimize the operating cost; and produces pareto-optimal solutions using genetic algorithm. The user can choose any pareto-optimal solution for code offloading based on the current state of the mobile battery and network bandwidth.

In the current literature, the partitioning of an application with multiple parallel nodes, is the addition of the computational time of the nodes. However, if the parallel nodes are assigned to the multiple different cloud servers, the computational time is the maximum time of all nodes since the execution is parallel among all resources. Our model addresses this issue and introduces a queue to each cloud server to perform parallel execution of the parallel nodes to further improve the application response time prediction. Our model takes into consideration, multi-processor code offloading since the cloud servers are equipped with the multiple processors and available for code offloading.

2.5 Genetic Algorithm

The genetic algorithm (GA) has been the most popular technique in computation research [25] widely used in the area of mobile computing. The research [5], [9] and [28] uses genetic algorithm to find the near-optimal offloading solution for the code offloading problem. The genetic algorithm starts with a set initial population and produces new solutions based on the probability of crossover and mutation. It uses the fitness function to examine the solution and optimizes the objectives of the application.

2.5.1 Initialization:

In initialization, the user defines the initial population size, the probability of crossover and mutation. The GA initializes the user pre-defined population size of chromosomes. In computation offloading problem, each chromosome refers to a unique offloading solution.

2.5.2 Selection:

Selection is the process of choosing two chromosomes from the population to recombine for generating new population via crossover or mutation. The purpose of selection is to filter individuals in the hope that their offspring (chromosome) has higher fitness.

2.5.3 Crossover

The crossover operator is to combine two sets of chromosomes to generate new offspring(s) (chromosomes). It is applied to selected individuals with the hope that they produce child(ren) with better fitness. The process of recombination is as follow:

1. The selection operator selects at random a pair of two chromosomes to mate
2. A random cross-site is selected in the gene
3. The position values are swapped between both chromosomes following the cross-site to produce new offspring(s).

2.5.4 Mutation

The mutation operator slightly modifies chromosomes to improve the fitness and avoid early convergence. It prevents the algorithm to get trapped in the local minimum. The crossover exploits the chromosome to find the better solution and mutation helps in the exploration of the whole search space. There are different forms of mutation, for different kinds of representation. A simple mutation is about inverting the value of each gene with the user pre-defined probability.

2.5.5 GA Objectives:

The GA has the ability to optimize multiple objective of the application simultaneously. In a multi-objective problem there is no best solution with respect to all objectives. Thus, it produces the pareto-optimal solution for the objectives which cannot be simply compared with each other.

2.5.6 Fitness Function:

The fitness function consists of mathematical model of GA objective. In this research, the fitness function consists of three objectives: minimize response time, minimize energy consumption, and minimize monetary cost of the offloading problem.

Chapter 3: The Pareto-Optimal Solution

In this chapter, the theoretical research framework is discussed to find a near-optimal offloading allocation for the multi-objective code offloading problem. The gain due to the external and internal parallel execution, the relevant definitions, and computing parameters to achieve the near-optimal offloading solution will also be discussed in this chapter. The conversion of a mobile application to workflow graph and the effect of genetic algorithm will be discussed in this chapter. This chapter is organized as follow:

3.1: Definitions and Computing Parameters

3.2: Model without Considering Parallel Execution of Tasks

3.3: Introductory Example

3.4: Evaluating a given allocation without Considering Parallel Execution of Tasks (for the introductory example)

3.5: Near-Optimal Allocation(s) using Genetic Algorithm without Considering Parallel Execution (for the Introductory Example)

3.6: Our Model considering Parallel Execution of Tasks

3.7: Evaluating a given allocation Considering Only External Parallelism (for the introductory example)

3.8: Near-Optimal Allocation(s) using Genetic Algorithm Considering Only External Parallelism (for the Introductory Example)

3.9: Evaluating a given allocation Considering both Internal and External Parallelism (for the introductory example)

3.10: Near-Optimal Allocation(s) using Genetic Algorithm Considering both Internal and External Parallelism (for the Introductory Example)

3.1 Definitions and Computing Parameters:

Definition 1 (Mobile application): A mobile application is invoked by a mobile user through his/her mobile device for a particular purpose. A mobile application typically consists of several tasks.

Definition 2 (resource). A resource can be either the mobile device (from which the computation can be offloaded) or a remote cloud server. Let D be the set of all devices. The set D contains the mobile device and the K cloud servers. The set D thus has K+1 elements.

$$D = \{d_0, d_1, d_2, \dots d_k\}$$

Where $d_0$is the mobile device and $d_1, d_2, \dots\dots d_k$ represent the cloud servers.

Definition 3 (Mobile device). A mobile device is a cell-phone or any portable device that can connect to the internet and request execution of application tasks from computing clouds. The mobile device d_0 is a homogenous multi-core device which is modeled as a five tuple $< b_0, n_0, s_0, pc_0, pd_0, pi_0 >$ where b_0 is the current battery percentage of the mobile device, n_0 is the number of processors in the mobile device, and for each processor s_0 is the processing speed of that processor (in million instructions per second), pc_0 is the computation power consumption, $pd_0$is the power consumption for communication (send and receive data), and pi_0 is the power consumption while the device is idle.

Definition 4 (Remote Cloud Servers). In this work, a mobile device can offload its computation to more than one cloud servers. A cloud server is a homogenous multi-core computational resource

(e.g. a virtual machine) that can execute tasks of a mobile application. A cloud server $d_c \; where \; c = 1, 2, \ldots K$ is modelled as a three tuple $< n_c, s_c, r_c >$ where n_c is the number of cores in the cloud server, s_c is the processing speed of each core (in million instructions per second), and r_c is the monetary rate of renting the cloud server from the cloud provider (in dollars per minute). It is assumed that if a cloud server is used for executing certain tasks of a mobile application, then the mobile user must rent the server from the cloud provider for the whole duration of execution of the application.

Definition 5 (Device-to-device bandwidth): The current data bandwidth between any two devices must be known. This is necessary to estimate the communication time between the two devices for data transfer. Let bandwidth (d_u, d_v) be the bandwidth between device d_u and device d_v, where u, v = 0, 1, 2, …K, and u is not equal to v.

Definition 6 (Mobile User Profile): Usually a mobile user pays a fixed monthly monetary cost to the internet service provider for uploading and downloading data from the internet. This fixed cost is charged for a limited fixed amount of data regardless of whether the user uses this data or not. If the user goes over the limited amount, a different monetary rate for consumption is applied to the additional amount consumed. For example, let a mobile user pays 25 dollars per month for 1GB data and if the user goes over 1GB, then a rate of 30 cents per MB will be charged for the additional amount. A mobile user profile is modelled as a two tuple $\langle \rho, \alpha \rangle$where ρ is the current remaining amount of data left from the fixed portion and α is the monetary rate for the additional amount of data (in dollars per MB).

Definition 7 (Mobile Application Workflow): The workflow of a mobile application defines the execution sequence of the tasks. It is modelled as a *workflow graph* $G = (T, E)$ where the set of vertices $T = \{t_1, t_2, \ldots t_N\}$ represents the N tasks of the mobile application and the set of edges $E = \{e\ (t_i, t_j)\ such\ that\ t_i, t_j \in T\}$ defines the inter-dependency between the tasks.

A task of the mobile application receives some input data and produces some output data. All the tasks of a mobile application may not be suitable for offloading to remote cloud servers. A task may not be offloadable if it needs access to local components (such as a camera or other sensors) or its execution on a remote cloud server might cause security problems.

In the workflow graph G, each task $t_i \in T$ is modelled as a two tuple $< o_i, \omega_i >$ where o_i is the type (true for offloadable or false for non-offloadable) of the task t_i and ω_i is the amount of CPU cycles (in million instructions) required for execution of task t_i.

Each directed edge $E = \{e\ (t_i, t_j)\ such\ that\ t_i, t_j \in T\}$ represents the dependency of t_j on t_i for execution. Each edge $e\ (t_i, t_j)$ is associated with a value $< \omega_{ij} >$ where ω_{ij} represents the amount of data that needs to be communicated between the devices executing the tasks t_i and t_j. This data transfer does not happen if the tasks t_i and t_j are executed on the same device. Let $source(t_i)$ be the set of tasks on which the task t_i depends on for execution. Let $sink(t_i)$ be the set of tasks which depends on task t_i for execution. We define the level of task t_i, $level(t_i)$ be the maximum of the levels of the tasks on which t_i depends on for execution plus 1, i.e.

$$level(t_i) = max\ \{level(source(t_i))\} + 1$$

For a given offloading allocation$[f_1, f_2, \dots, f_N]$, we can construct a *time-weighted workflow graph* $TWG = (T, E)$ from the workflow graph G as follows: Each vertex $t_i \in T$ is associated with a weight $w_i$that represents the time to execute the task t_i on computation resource f_i. w_i can be computed by dividing the amount of CPU cycles required for execution of task t_i by the processing speed of single core for device d_u i.e. $w_i = {}^{\omega_i}/_{S_u}$. Each $e(t_i, t_j)$ such that $t_i, t_j \in T$ is associated with a weight w_{ij} that represents the communication time needed for data transfer when task t_i will be executed on device du (i.e. $f_i = d_u$) and task t_j will be executed on device d_v (i.e. $f_j = d_v$). This communication time depends on the amount of the data that needs to be transferred and the bandwidth between the devices d_u and d_v. Thus, w_{ij} can be computed as:

$$w_{ij} = {}^{\omega_{ij}}\Big/_{bandwidth_{(d_u, d_v)}} \; where \; d_u \neq d_v \; otherwise \; w_{ij} = 0$$

Definition 8 (Offloading Allocation): In a multi-server offloading scenario, each offloadable task of a mobile application can be allocated to run on either the mobile device or on one of the remote cloud servers. Each non-offloadable task must be allocated to run on the mobile device. An offloading allocation is defined as one such allocation of tasks of the workflow graph to devices. An offloading allocation *a*, of the tasks in set T to the devices in set D is represented as $[f_1, f_2, \dots, f_N]$ where each $f_i = d_u$ where $u = 0, 1, 2, \dots K$.

3.2 Model without Considering Parallel Execution of Tasks

In this section, a mathematical model is represented, which can be used to calculate the theoretical values of any offloading allocation a. It consists of response time RT_a of the application, battery energy consumption E_a of the mobile device, and monetary cost C_a that will be incurred for the

mobile user for executing the application according to the allocation a. This model does not consider the parallel execution of tasks. Here we follow the philosophy of the works in [24] and [27].

3.2.1 Response Time:

The response time is the sum of the execution time of each task $T = \{t_1, t_2, \dots t_N\}$ and communication time of all edges $E = \{e\ (t_i, t_j)\ such\ that\ t_i, t_j \in T\}$

$$RT_a = \sum_{t_i \in T} w_i + \sum_{e(t_i,t_j) \in E} w_{ij} \qquad (1)$$

Where ω_i is the amount of CPU cycles (in million instructions) required for execution of task t_i. Each edge $e\ (t_i, t_j)$ is associated with a value $< \omega_{ij} >$ where ω_{ij} represents the amount of data that needs to be transferred between the devices executing the tasks t_i and t_j for communication.

3.2.2 Energy Consumption:

The energy consumption of an offloading allocation is the sum of execution energy, communication energy and idle energy. The energy consumption for an offloading allocation a is as follow:

$$Execution\ Energy = \sum_{\substack{t_i \in T, \\ f_i = d_0}} w_i * pc_0 \qquad (2)$$

The execution energy is a product of the execution time w_i of the mobile processor with the user pre-defined value of the execution power pc_0

$$Communication\ Energy = \sum_{\substack{e(t_i,t_j)\in \mathrm{E},\\ f_i \neq f_j\ and\\ either\ f_i=d_0\\ or\ f_j=d_0}} w_{ij} * pd_0 \quad (3)$$

The communication energy is the product of the communication time w_{ij} of each edge $e(t_i, t_j) \in$ E with the user pre-defined pd_0.

$$Idle\ Energy = \sum_{\substack{t_i \in T,\\ f_i \neq d_0}} w_i * pi_0 + \sum_{\substack{e(t_i,t_j)\in \mathrm{E},\\ f_i \neq f_j \neq d_0}} w_{ij} * pi_0 \quad (4)$$

The idle Energy is the sum of the amount of time the local processor is idle when the execution is taking place in other computing resources (cloud servers) and the amount of communication time between two cloud servers where local processor is staying idle.

Thus, the mathematical model of energy consumption is given below.

$$\mathrm{E_a} = \sum_{\substack{t_i \in T,\\ f_i = d_0}} w_i * pc_0 + \sum_{\substack{e(t_i,t_j)\in \mathrm{E},\\ f_i \neq f_j\ and\\ either\ f_i=d_0\\ or\ f_j=d_0}} w_{ij} * pd_0 + \sum_{\substack{t_i \in T,\\ f_i \neq d_0}} w_i * pi_0 + \sum_{\substack{e(t_i,t_j)\in \mathrm{E},\\ f_i \neq f_j \neq d_0}} w_{ij} * pi_0 \quad (5)$$

3.2.3 Execution Cost:

The execution cost of an application is dependent on the user mobile network package and cloud VM renting cost based on the offloading allocation a. It is assumed that the monthly subscription

is limited and ρ is the current remaining amount of data available for offloading and α is the monetary rate for the additional amount of data (in dollars per MB).

The cost is calculated by calculating the additional data required D_a for any offloading allocation a

$$D_a = \left(\sum_{\substack{e(t_i,t_j)\in E, \\ f_i \neq f_j \; and \\ either \; f_i = d_0 \\ or \; f_j = d_0}} w_{ij} * bandwidth(f_i, f_j) \right) - \rho \qquad (6)$$

Where D_a is the difference between the remaining network data bandwidth and required network bandwidth.

$$C_a = \begin{cases} \sum_{\substack{c=1, \\ \text{if any } f_i = d_c, t_i \in T}}^{K} RT_a * r_c & if \; D_a < 0 \\ \sum_{\substack{c=1, \\ \text{if any } f_i = d_c, t_i \in T}}^{K} RT_a * r_c + (D_a * \alpha), & if \; D_a > 0 \end{cases} \qquad (7)$$

Thus, the execution cost C_a is the sum of the products of total response time RT_a of an allocation a with cloud server operating charges if there is no additional network data bandwidth D_a is required. In case, if there is additional network data bandwidth required, the network charges $(D_a * \alpha)$ are added in the overall cost.

3.3 Introductory Example

Workflow Graph:

Figure 1 shows a workflow graph example consisting of seven different tasks for a mobile application. Similarly, there are three different resources available for execution, local mobile device, and cloud servers $VM_1 and\ VM_2$. Such that each of these seven tasks can be either offloaded to one of the cloud server (VM_1, VM_2) or executed in local processor. The goal is to process all seven tasks in the shortest possible time with minimum consumption of mobile energy and with spending lowest processing cost possible.

In *Figure 1*, the task $t_1$must be executed first. When it is finished, the tasks t_2, t_3, $t_4$and t_5 can execute in parallel. When task t_2 finishes, task t_6 can start its execution. Similarly, when tasks t_3, t_4, t_5 and t_6 are finished, task t_7 can begin execution. Once task t_7 is finished, the execution of the mobile application is complete.

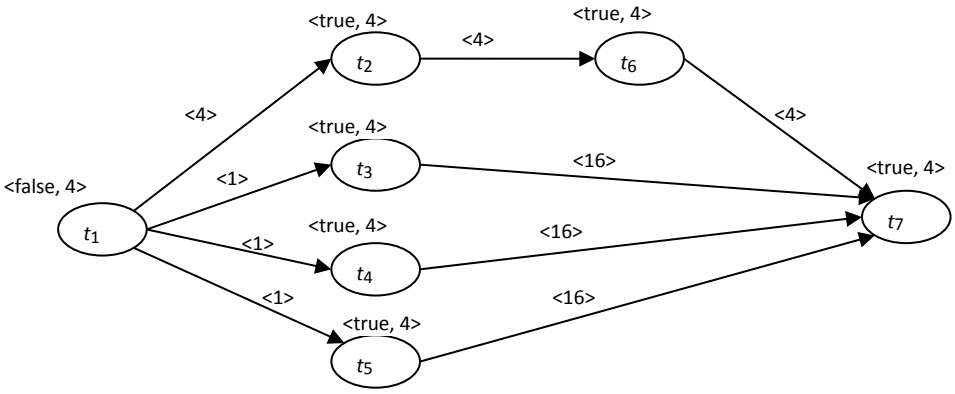

Figure 1: A Simple Workflow Graph

In *Figure 1 above*, the task t_1 is non-offloadable task whereas tasks t_2, t_3, t_4, t_5, t_6 and t_7 are offloadable tasks. Each task in this example needs to execute 4 million of instructions for completion. Each edge between t_1 and t_j is labelled with the amount of data that needs to be transferred between the devices executing the tasks t_i and t_j for communication. For example, if

the tasks t_5 and t_7 are executed on different devices, then 16MB of data needs to be transferred between those devices. The details of the workflow graph in *Figure 1* is as follow:

$$source(t_1) = \{\} \quad and\ sink(t_1) = \{t_2, t_3, t_4, t_5\}$$

$$source(t_2) = \{t_1\}\ and\ sink(t_2) = \{t_6\}$$

$$source(t_3) = \{t_1\}\ and\ sink(t_3) = \{t_7\}$$

$$source(t_4) = \{t_1\}\ and\ sink(t_4) = \{t_7\}$$

$$source(t_5) = \{t_1\}\ and\ sink(t_5) = \{t_7\}$$

$$source(t_6) = \{t_2\}\ and\ sink(t_6) = \{t_7\}$$

$$source(t_7) = \{t_3, t_4, t_5, t_6\}\ and\ sink(t_7) = \{\}$$

$$level(t_1) = 1$$

$$level(t_2) = level(t_3) = level(t_4) = level(t_5) = 2,$$

$$level(t_6) = 3$$

$$level(t_7) = 4.$$

Time-Weighted Workflow Graph for the allocation $\boldsymbol{a = [d_0, d_2, d_1, d_1, d_1, d_0, d_2]}$**:**

Let us refer Example-1 whose workflow is shown in *Figure 1*. Let the mobile user gets 1GB of data per month as part of monthly subscription. Let us assume that out of 1GB, 500MB is unused and the charge for using additional data is 2 cents per MB of data such that

$$\langle \rho, \alpha \rangle = < 500MB, 0.2\ dollars/MB >$$

For the mobile user profile. Further, let's assume that the current state of the mobile device is described as:

$$< b_0, n_0, s_0, pc_0, pd_0, pi_0 > = < 95\%, 1Core, 1000MIPS, 0.5W, 0.25W, 0.15W >$$

The mobile device is currently at 95% battery remaining, only has 1 processing core, $1MIPS$ is the processing speed of the core and $0.5, 0.25$ and $0.15\ watts$ will be the execution, communication, and being idle power consumption. Let us consider that there are two cloud servers $d_1$and d_2 where the offloadable tasks can be allocated. The cloud resources configuration is as follow

$$d_1 = < n_1,\ s_1,\ r_1 > = < 4Core, 2000MIPS, 0.03\ dollars/Min >$$
$$d_2 = < n_2,\ s_2,\ r_2 > = < 4Core, 4000MIPS, 0.05 dollars/Min >$$

d_1 is described as i.e. it has 1 core, 2 MIPS is the speed of the core, and 0.03 dollars is the charge per minute for renting d_1. Similarly, the cloud server d_2 also has 1 core, with MIPS core speed and 0.05 dollars per minute renting charge. The data transfer bandwidth among different resources is assumed to be:

$$bandwidth(d_0, d_1) = 1MBPS$$
$$bandwidth(d_0, d_2) = 2MBPS$$
$$bandwidth(d_1, d_2) = 4MBPS$$

Such that the bandwidth between local mobile device d_0 and cloud server d_1 is 1MBPS and the bandwidth between mobile device d_0 and cloud server d_2 is 2MBPS. Finally, the data bandwidth between two cloud servers is $d_1$and d_2 is 4MBPS.

Let's consider the following offloading allocation $a = [d_0, d_2, d_1, d_1, d_1, d_0, d_2]$ for the tasks $[t_0, t_1, t_2, t_{3,} t_4, t_5, t_6, t_7]$ respectively. The time-weighted workflow graph corresponding to *Figure 1* is shown in *Figure 2*. In *Figure 2*, all the weights (on the vertices and the edges) are in seconds.

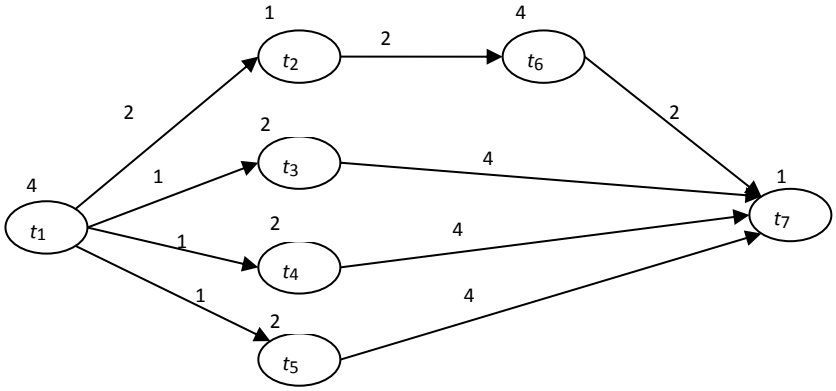

Figure 2: The time-weighted workflow graph corresponding to the offloading allocation a.

The *Figure 2* above shows the time-weighted workflow graph of *Figure 1* for the offloading allocation a. The task t_1 and t_2 are assigned to the local mobile device so their execution time is 4 seconds. The task t_3, t_4 and t_5 execution times are 2 seconds since the cloud server d_1 speed is 2MIPS. Similarly, the task t_2, t_7 execution times are only 1 seconds since cloud server d_2 speed is 4MIPS.

The communication time between edges $e(t_1, t_2), e(t_2, t_6),$ and $e(t_6, t_7)$ is 2 seconds since the bandwidth between mobile device and cloud server d_2 is 2MBPS. Similarly, the communication

time between the edges $e(t_1, t_3), e(t_1, t_4)$, and $e\ (t_1, t_5)$ is 1 seconds since the bandwidth between mobile device and cloud server d_1 is 1MBPS. Finally, the communication time between the edges $e(t_3, t_7), e(t_4, t_7)$, and $e(t_5, t_7)$ is 4 seconds since the bandwidth between two cloud servers d_0 and d_1 is 4MBPS.

3.4 **Evaluating a given allocation without Considering Parallel Execution of Tasks (for the introductory example)**

In this section, for the introductory example in section 3.3, we compute the Response Time, Energy Consumption and Monetary Cost for a given allocation $a = [d_0, d_2, d_1, d_1, d_1, d_0, d_2]$ without considering parallel execution.

3.4.1 Response Time

Corresponding to each offloading allocation *a*, we can compute the response time RT_a of the mobile application, battery energy consumption E_a in the mobile device, and monetary cost C_a that will be incurred for the mobile user for executing the application according to the allocation *a*. The computation of these three measures is given in the next section. The response time for *Figure 2* workflow graph can be calculated using *equation (1).* It is as follow:

$$RT_a = \sum_{t_i \in T} w_i + \sum_{e(t_i,t_j) \in E} w_{ij}$$

$$\sum_{t_i \in T} w_i = (4 + 1 + 2 + 2 + 2 + 4 + 1) = 16$$

$$\sum_{e(t_i,t_j) \in E} w_{ij} = (2 + 1 + 1 + 1 + 2 + 4 + 4 + 4 + 2) = 21$$

$$RT_a = 37sec$$

The Overall response time RT_a for the offloading allocation *a* for *Figure 2* is 37 seconds which includes the *computation time* of 16 seconds and the *communication time* of 21 seconds.

3.4.2 Energy Consumption:

The energy consumption for the offloading allocation *a* is the sum of *computation energy*, *communication energy* and *idle energy*. The computation energy can be calculated using *equation (2)* which is as follow:

$$Computation\ Energy\ = \sum_{\substack{t_i \in T, \\ f_i = d_0}} w_i * pc_0 = (4+4) * 0.5 = 4.0\ Joules$$

Computation energy is equal to the local mobile device assigned tasks (t_1, and t_6) execution time multiply by the user pre-defined computation power pc_0 of the mobile processor which is equal to 0.5 Watts. The total Computation energy for the offloading allocation *a* is 4 Joules. Similarly, the *communication energy* can be calculated by using *equation (3)* which is as follow:

$$Communication\ Energy = \sum_{\substack{e(t_i,t_j) \in E, \\ f_i \neq f_j\ and \\ either\ f_i = d_0 \\ or\ f_j = d_0}} w_{ij} * pd_0$$

$$= 0.25 * (2+1+1+1+2+2) = 2.25\ Joules$$

The *communication energy* is equal to the sum of all edges where local mobile processor is either sending or receiving the offloading data from any other resource i.e. edges $e(t_1, t_2)$,

$e(t_1, t_3), e(t_1, t_4), e(t_1, t_5), e(t_2, t_6)$and $e(t_2, t_7)$ multiply by the user pre-defined communication power pd_0 which is equal to 0.25 Watts. The *communication energy* of the offloading allocation *a* is 2.25 Joules. Finally, the idle energy can be calculated by the *equation (4)* which is as follow:

$$Idle\ Energy = \sum_{\substack{t_i \in T, \\ f_i \neq d_0}} w_i * pi_0 + \sum_{\substack{e(t_i, t_j) \in E, \\ f_i \neq f_j \neq d_0}} w_{ij} * pi_0$$

$$= 0.15 * ((1 + 2 + 2 + 2 + 1) + (4 + 4 + 4)) = 3 \text{ Joules}$$

The *idle energy* is the sum of all computation tasks which are offloaded to any cloud server d_0 and d_1, (t_2, t_3, t_4, t_5, and t_7) multiply by the user pre-defined idle power pi_0. Also, all edges communication time where local mobile processor is not involved i.e. $e(t_3, t_7), e(t_4, t_7)$and $e(t_5, t_7)$ multiply by idle power pi_0 which is 0.15 Watts. The total *idle energy* for the offloading allocation is 3 Joules for the offloading allocation *a.* Thus, the total power of the offloading allocation *a* can be calculated by the *equation (5)*

$$E_a = Computation\ Energy + Transmission\ Energy + Idle\ Energy$$

$$= 4 + 2.25 + 3 = 9.25\ Joules$$

The total energy consumption for the offloading allocation *a* is the sum of *computation energy*, *communication energy*, and *idle energy* which is 9.25 *Joules*.

3.4.3 Monetary Cost:

The computation cost of the offloading allocation *a* can be calculated using the cost *equation (7)* which is as follow:

$$\text{Cost} = C_{\text{a}} \ + \ (\text{D}_{\text{a}} * \alpha), \text{if } \text{D}_{\text{a}} > 0 \ else \ C_{\text{a}}$$

Since the remaining data limit in the user wireless plan is assumed to be 500MB and the offloading allocation *a* does not exceed the remaining data limit so there is no on additional charge on the user wireless network plan, resulting D_{a} is 0. However, the renting of cloud servers, d_1 and d_2, for the application duration of 37 seconds incur monetary processing cost C_a which is as follow.

$$C_a = \ 37 * \left(\frac{0.03}{60}\right) + \ 37 * \left(\frac{0.05}{60}\right) = \ 4.9 \frac{cents}{minute}$$

The monetary cost C_a of the offloading allocation *a* is the renting cost of cloud server d_1 which is 0.03 cents/minute as well as the renting cost of cloud server d_2 which is 0.05 cents/minute multiply by the response time of the application which is 37 seconds. The total monetary cost of the offloading allocation *a* is equal to 4.9 cents/minute.

Thus, the response time of the workflow graph presented in *Figure 2* for the offloading allocation $a = \ [d_0, d_2, d_1, d_1, d_1, d_0, d_2]$ is 37 seconds with the overall energy consumption of 9.25 Joules and the processing cost of 4.9 $cents/minute$

3.5 Near-Optimal Allocation(s) using Genetic Algorithm without Considering Parallel Execution (for the Introductory Example)

In this section, we will apply Genetic Algorithm (GA) to find the near-optimal offloading allocation for the introductory example provided earlier (section 3.3). The GA will start with a set number of offloading solutions' population which is defined by the user. Using the fitness function, it will generate new offloading solutions' genes based on the mutation and crossover operator probability. In the exploration of the near-optimal solution, the GA will consider multi-objectives to find an offloading allocation which has minimum response time, energy consumption and processing cost of the application.

3.5.1 GA Parameters:

The genetic algorithm parameters are as follow:

Initial Population Set: 100 solutions

Number of evaluations to find the near-optimal solution: 10,000

Mutation Algorithm: Uniform Mutation (UM)

Probability of Mutation: 1/Number of Tasks = 1/7 = 0.143

Crossover Algorithm: Subset Crossover (SSX)

Probability of Crossover: 0.9

The GA starts with an initial population of 100 solutions and iterate on the initial population for 10,000 evaluations. Each evaluation generates a new gene, based on the mutation or crossover operator probability. The mutation algorithm is chosen to be a Uniform Mutation (UM) with a probability of 0.03. The uniform mutation mutates each decision variable in the gene by selecting

a new value within its bound uniformly at random [15]. Similarly, the crossover algorithm is chosen to be Subset Crossover (SSX) with a probability of 0.9. The subset crossover swaps half of the non-matching decision variable between two parent's genes [15].

Next, we will show the results of the introductory example (section 3.3) by applying GA. The GA will be applied for three cases: allocate all the tasks on only local mobile processor (no offloading); allocating the tasks among the local mobile processor and one VM (single-site offloading); and allocating the tasks among the local mobile processor and two VMs (multi-site offloading).

3.5.2 No offloading

When applying GA with a local processor, there is only one offloading allocation – which is all tasks must be executed in the local processor. The GA reveals the following offloading allocations for only the local processor resource d_0.

Table 1: GA with no offloading

Name	Offloading Allocation	Response Time (Sec)	Energy (Joules)	Cost (cents/min)
a_1	$[d_0, d_0, d_0, d_0, d_0, d_0, d_0]$	28.0	14.00	0.00

The allocation a_1 determines the $RT_a = 28\ seconds, E_a = 14.0\ Joules, Cost = 0\ cents/min$. Thus, the processing of all task in the *example 1* will required 28 seconds which will consume 14 Joules of mobile power and there will not be any additional processing cost since there is no offloading is possible for this configuration.

3.5.3 Single Site Offloading

In this configuration of a local processor with one VM, since each element can have a value of either 0 (Local) or 1 (VM), the available offloading allocations are $2^7 = 128$. GA will start with the initial population of 100 solution and generate new solution to find the near-optimal solution. In this configuration task t_0 is forced to be executed in the local processor, since *example 1* refer to a mobile application and offloading should start from the local processor. All other tasks of the application will be processed in the local processor or the cloud server. The GA provides the following offloading allocations:

Table 2: GA Offloading with one VM

Name	Offloading Allocation	Response Time (Sec)	Energy (Joules)	Cost (cents/min)
a_1	$[d_0, d_1, d_1, d_1, d_1, d_1, d_1]$	23.0	05.55	1.15
a_2	$[d_0, d_0, d_0, d_0, d_0, d_0, d_0]$	28.0	14.00	0.00

The GA suggest the two offloading allocations a_1 and a_2 for the user to choose based on their application objective. First offloading allocation, a_1, which provides the minimum $RT_{a1} = 23\ seconds, and\ E_{a1} = 5.55\ Joules$, however the user will be charged a processing fee of $Cost = 1.15\ cents/min$. Similarly, the offloading allocation a_2 does not have any processing cost, $Cost = 0.00\ cents/min$. However, the application takes longer to complete $RT_{a2} = 28\ seconds$ and it will consume $E_{a2} = 14.0\ Joules$ of the mobile battery.

3.5.4 Multi-Site offloading

In the configuration with the local processor with two VMs, each element of the offloading solution can have three states, 0 (local), 1 (VM_1), 2 (VM_2) so there are $3^7 = 2187$ different available

offloading allocations. The GA starts with the initial population of 100 solutions and generates a new gene using either crossover or mutation. It observes the behavior of the new gene to explore the near-optimal solution. Like the local processor with one VM configuration, the task t_0 is forced to be processed in the local processor, and all other tasks can be processed in the local processor or any available cloud server. The GA provides two different offloading allocations:

Table 3:GA Offloading with two VM

Name	Offloading Allocation	Response Time (Sec)	Energy (Joules)	Cost (cents/min)
a_1	$[d_0, d_2, d_2, d_2, d_2, d_2, d_2]$	13.5	03.76	1.125
a_3	$[d_0, d_0, d_0, d_0, d_0, d_0, d_0]$	28.0	14.00	0.00

The GA suggests two different offloading allocations a_1, $and\ a_2$, for the user to choose based on their current state of the mobile device. If the user wants to optimize the response time and energy consumption of the mobile application, the user can choose offloading allocation a_1 which produces $RT_{a1} = 13.5\ seconds$, $E_{a1} = 3.775\ Joules$ but require $Cost = 1.125 cents/min$. Likewise, if the user is hesitant to spend any additional cost on the application, they can use the allocation a_2 which does not have any processing cost, $Cost = 0.00 cents/min$. However, the application takes longer to complete at $RT_{a2} = 28\ seconds$ and it will consume $E_{a2} = 14.0\ Joules$ of the mobile battery.

3.6 Our Model considering Parallel Execution of Tasks

In this section, we describe our new algorithm to evaluate an allocation that considers both (external and internal) parallel execution of the application tasks. The external parallelism is among different available computing resources, and internal parallelism among different

processing cores of a single computing resource. The goal is to explore for an offloading allocation with minimum Response Time, Energy Consumption, and Monetary Cost.

The model of the internal parallelism of a device d_u(where u = 0, 1, 2, …K) with r_u cores is shown in *Figure 3* below,

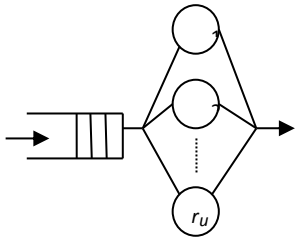

Figure 3: The device du modeled as a multi-server queueing station with ru number of servers

The *Figure 3* above represents a device d_u as a single multi-server queueing station that consists of a job queue and r_u number of identical servers. The r_u cores of the queuing station perform internal parallelism for the device d_u. Similarly, for the model of external parallelism, we assume that the parallelism can exist among different devices and each device maintains its own queue.

3.6.1 Definitions:

1. Each action which is required in the workflow graph shown in the *Figure 2* above**,** is referred to a *Job.*
2. A *job* can be scheduled to execute on any available device.
3. There can be three kinds of jobs with respect to the execution of task t_i:

   3.1. *receiveJob* (t_j, t_i):

   The execution of this job represents *receiving* of data—produced by task t_j—by the device hosting task t_i. This data will be needed for executing the task t_i. This job is relevant when tasks t_j and t_i are hosted in different devices.

3.2. *executeJob*(t_i):

The execution of this job represents execution of the task t_i.

3.3. *sendJob* (t_i, t_j):

The execution of this job represents *sending* of data—produced by task t_i—from the device hosting task t_i to the device hosting task t_j. This data will be needed for executing the task t_j. This job is relevant when tasks t_i and t_j are hosted in different devices.

4. Each job has a *depth*. The depth of a job captures its dependencies on other jobs. For example, a job with depth 2 will need information from one or more jobs at depth 1
5. Each job has *arrival time, start time, service time* and *end time.*
   5.1. The *arrival time* denotes the time when the job can be started to run if a server in the scheduled device is free. Otherwise, if all the servers are busy, then the job must wait in the queue of the scheduled device.
   5.2. The *start time* denotes the time instant when one of the servers of the scheduled device starts processing the job.
   5.3. The *service time* denotes the time needed for processing the job.
   5.4. The *end time* denotes the time instant when the job processing is complete, i.e. *end time* = *start time + processing time*.
6. Each core in a device can be either in *busy* state or *idle* state.
   6.1. The core is in *busy* state means that it is busy processing a job.
   6.2. The core is in *idle* state means that the core is idle.
7. Each core has *computation time* and *communication time* associated with it
   7.1. The *computation time* denotes the time the core spends in executing tasks.
   7.2. The *communication time* denotes the time the core spends in sending or receiving data.

3.6.2 Job Generation

The process of generating, assigning appropriate time and processing *Execution_Jobs, Receiving_Jobs, and Send_Jobs* are as follows:

Step-1:

In this step, we generate jobs and schedule them in relevant devices. We set the arrival times, service times and depth for the jobs.

totalJobsList = {}

For each $t_i \in T$ processed in the order of increasing levels:

1. *receiveJobsList*(t_i) = {}
2. For each task $t_j \in source(t_i)$ where $f_j \neq f_i$:
   2.1. Schedule a *receiveJob*(t_j, t_i) on the device f_i such that:
      2.1.1. the *arrival time* of this job will be the *start time* of *sendJob*(t_j, t_i) (the job *sendJob*(t_j, t_i) should already be scheduled on device f_j)
      2.1.2. the *service time* of this job will be w_{ji},
      2.1.3. the depth of this job will be the depth of *sendJob*(t_j, t_i) plus 1.
   2.2. Add the job *receiveJob*(t_j, t_i) to the two lists *receiveJobsList*(t_i) and *totalJobsList*.
3. Schedule a *executeJob*(t_i) on the device f_j such that:
   3.1. The *arrival time* of this job will be the maximum of the *end time*s of the jobs in the *receiveJobsList*(t_i)
   3.2. The *service time* of this job will be w_i
   3.3. The depth of this job will be 3**level*(t_i)-2.
4. Add the job *executeJob*(t_i) to the list *totalJobsList*.

5. For each task $t_k \in sink(t_i)$ where $f_k \neq f_i$:

   5.1. Schedule a *sendJob*(t_i, t_k) on the device f_i such that:

      5.1.1. The *arrival time* of this job will be the *end time* of the *executeJob*(t_i)

      5.1.2. The *service time* of this job will be w_{ik}

      5.1.3. The depth of this will be the depth of *executeJob*(t_i) plus 1.

   5.2. Add the job *sendJob*(t_i, t_k) to the list *totalJobsList*.

Step-2:

In this step, we process all the jobs to set the start time and end time for each of them.

Process each job in the *totalJobsList* in the order of increasing depths as follows:

1. Obtain the time instant *t* when one of the cores of the job's scheduled device will be free. Let the core which will be free at time *t* be *p*.

   1.1. If the *arrival time* of this job is greater or equal to *t*, then set the *start tim*e of this job = its *arrival time*. Otherwise, set the *start tim*e of this job = *t*.

   1.2. Set the *end tim*e of this job = its *start time* + its *service time*. Let the time period, *end tim*e of this job minus *start tim*e of this job, be *b*.

   1.3. Set the state of the core *p* to be *busy* for the time period *b*. If this job is an *executeJob*, add the time period *b* to the *computation time* of core *p*, otherwise add *b* to the *communication time* of core *p*.

Step-3:

Compute the required measures as follows:

1. Compute the response time RT_a as:

1.1. RT_a = maximum of the *end time*s of the jobs which are at the highest depth.

2. Compute the energy consumption E_a on the mobile device d_0, as follows:

2.1. Energy consumption of each core *p* can be computed as (*computation time* of *p*) * pc_0 + (*communication time* of *p*) * pd_0 + (RT_a – *computation time* of *p* - *communication time* of *p*) * pi_0

2.2. $E_a = \sum_{p=1}^{n_0}(\text{Energy consumption of each core } p)$

3. Compute the monetary cost, C_a, using equation (7)

3.1. Since the value of RT_a may be different when parallel execution is considered, the cost C_a may be different in case of parallel execution as well.

For a given allocation *a*, the values of the three performance measures will likely vary when considering or ignoring parallelism in the execution. Hence, the near-optimal allocation(s) that minimizes one or more of these measures may also vary as well when parallelism is considered.

3.6.3 External Parallel Execution

External parallel execution involves multiple task processing at the same instance of time among different available resources. The ideal parallel execution, without considering latency and communication overhead losses, can be represented by the following equation below [38]

$$RT_{Parallel} = {}^{RT_{Sequential}}/_{N} \qquad (viii)$$

Where $RT_{Parallel}$ refer to response time of an application, $RT_{Sequential}$ refer to the sequential time of an application and N refer to identical available computing resources. In this section, the gain of external parallel execution will be analyzed for the same introductory *example 1* mentioned

above. To model the external parallelism, we assume that the parallelism can exist among the devices and each device maintains its own queue.

3.7 Evaluating a given allocation Considering Only External Parallelism (for the introductory example)

The effect of external parallel execution for the time-weighted workflow graph with the offloading allocation $a = [d_0, d_2, d_1, d_1, d_1, d_0, d_2]$ is represented by the table of jobs for each available computing resource. The table 4, 5, and 6 below represents the schedule of jobs, **i.e**. the depth, *service_time*, *arrival_time*, *start_time* and *end_time* for each available computing resource d_0, d_1 and d_2 respectively. In this section, all the devices have one processor, thus no internal parallel execution is considered. However, external parallelism among the different processing resources is considered since they can execute tasks in parallel. The *Table 4* below represents the jobs which are handled by the resource Mobile Device d_0.

Table 4: Jobs scheduled in the Mobile Device d_0

Job Number	Job Type	Job Depth	Service Time (Sec)	Arrival Time (Sec)	Start Time (Sec)	End Time (Sec)
0	*execute*(t_1)	1	4	0	0	4
1	*sendJob*(t_1, t_2)	2	2	4	4	6
2	*sendJob*(t_1, t_3)	2	1	4	6	7
3	*sendJob*(t_1, t_4)	2	1	4	7	8
4	*sendJob*(t_1, t_5)	2	1	4	8	9
5	*receiveJob*(t_2, t_6)	6	2	7	9	11
6	*execute*(t_6)	7	4	11	11	15
7	*sendJob*(t_6, t_7)	8	2	15	15	17

From *Table 4* above, we see that the mobile device d_0 throughput total 8 jobs in which only one is *Receive_Job* and two are *Execute_Jobs* and total five jobs are *Sent_Jobs*. Similarly, the device d_0 spend 8 seconds for *computation*, and 9 secs for *communication*. All jobs except Job number 5 are started processing on their arrival time however, job number 5 (*receiveJob*(t_2, t_6)) arrives at the time = 7 seconds but wait in the queue until time = 9 when the mobile device d_0 becomes available. The total time for all the jobs in the mobile device d_0 is 17 seconds.

The *Table 5* below shows the jobs schedule for cloud server d_1

Table 5: Jobs scheduled in the Cloud Server d_1

Job Number	Job Type	Job Depth	Service Time (Sec)	Arrival Time (Sec)	Start Time (Sec)	End Time (Sec)
0	*receiveJob*(t_1, t_3)	3	1	6	6	7
1	*receiveJob*(t_1, t_4)	3	1	7	7	8
2	*receiveJob*(t_1, t_5)	3	1	8	8	9
3	*execute*(t_3)	4	2	7	9	11
4	*execute*(t_4)	4	2	8	11	13
5	*execute*(t_5)	4	2	9	13	15
6	*sendJob*(t_3, t_7)	5	4	11	15	19
7	*sendJob*(t_4, t_7)	5	4	13	19	23
8	*sendJob*(t_5, t_7)	5	4	15	23	27

The *Table 5* above shows the jobs schedule in the Cloud Server d_1. Server d_1 throughput total 9 jobs which includes equal number of three jobs for *ReceiveJobs*, *ExecuteJobs*, and *SendJobs.* The server d_1 spends three seconds on *ReceiveJobs*, 6 seconds on *executeJobs* and 12 seconds on the *SentJobs*. In server d_1, Job number 3-8 encounter the server in busy state and wait for its availability in the queue. Job 3 (*execute*(t_3)) must wait for 2 seconds in the queue before it can be processed. Job 4, 5, 6, 7, 8 required the queue wait time to be 3, 4, 4, 6, 8 seconds respectively. The total time the cloud server d_1 takes to process all the jobs is 27 seconds.

The *Table 6* below represents the Jobs schedule for Cloud Server d_2

Table 6: Jobs scheduled in the Cloud Server d_2

Job Number	Job Type	Job Depth	Service Time (Sec)	Arrival Time (Sec)	Start Time (Sec)	End Time (Sec)
0	*receiveJob*(t_1, t_2)	3	2	4	4	6
1	*execute*(t_2)	4	1	6	6	7
2	*sendJob*(t_2, t_6)	5	2	7	7	9
3	*receiveJob*(t_3, t_7)	6	4	15	15	19
4	*receiveJob*(t_4, t_7)	6	4	19	19	23
5	*receiveJob*(t_5, t_7)	6	4	23	23	27
6	*receiveJob*(t_6, t_7)	9	2	15	27	29
7	*execute*(t_7)	10	1	29	29	30

The *Table* 6 above shows the jobs schedule in the Cloud Server d_2. Server d_2 throughput total 8 jobs which includes five *ReceiveJobs,* and only one *ExecuteJobs* and one *SendJobs* from task t_2. Similarly, t_7 is the last task in the workflow graph so it does not contain any *SendJobs.* The server d_2 spends 16 seconds on *ReceiveJobs*, 2 seconds on *executeJobs* and 2 seconds on the *SentJobs*. In server d_2, Job 6 must wait for availability in the queue otherwise all jobs are processed upon their arrival time. The total time the cloud server d_2 takes to process all the jobs is 27 seconds.

As per *Step*-3 of our algorithm, we compute:

$$RT_a = end\ time\ of\ job\ (execute(t7)) = 30\ seconds$$

Since this job *execute*(t_7) has the highest depth in the task graph so the response time will be the end time of job *execute*(t_7). The energy consumption can be calculated using *equation (5).* where *computation time* is 8 seconds, *communication time* is 9 seconds and *idle time* is 13 seconds

$$E_a = (0.5) * 8 + (0.25) * 9 + 0.15 * (30 - 17) = 8.2\ Joules$$

The energy consumption for the offloading allocation a with external parallel execution is 8.2 *Joules*. Finally, the cost can be calculated using *equation (7)*

$$C_a = 30 * \left(\frac{0.03}{60}\right) + 30 * \left(\frac{0.05}{60}\right) + 0 = 0.04\ cents/Minutes.$$

Thus, the allocation $a = [d_0, d_2, d_1, d_1, d_1, d_0, d_2]$ with the server queue takes 30 seconds to process and it requires 8.2 Joules of energy with only the processing cost of only 4 cents/Minutes. The following *Table 7* below represents all three GA objectives for offloading allocation $a = [d_0, d_2, d_1, d_1, d_1, d_0, d_2]$ for no queue model (i.e. without parallelism) and for queue based model (i.e. with external parallelism).

Table 7: External Parallel Execution Gain

Allocation	Config	Response Time (Sec)	Energy Consumption (Joules)	Processing Cost (Cents/Min)
$[d_0, d_2, d_1, d_1, d_1, d_0, d_2]$	No Queue	37	9.25	4.9
	With Queue	30	8.2	4.0

Thus, the above *Table 7* shows that introducing the queue in the work-flow graph reduces the response time to 30 seconds from 37 seconds. Also, the energy consumption on the mobile processor is reduced to 8.2 Joules instead of 9.25 Joules. Finally, the processing cost is reduced to 4.0 cents/minute instead of 4.9 cents/minutes.

3.8 Near-Optimal Allocation(s) using Genetic Algorithm Considering only External Parallelism (for the Introductory Example)

Let's introduce the genetic algorithm on the above queuing model of the work flow graph presented in *Figure 2* with the following GA parameters:

Initial Population Set: 100 solutions

Number of evaluations to find the near-optimal solution: 10,000

Mutation Algorithm: Uniform Mutation (UM)

Probability of Mutation: 1/Number of Tasks = 1/7 = 0.143

Crossover Algorithm: Subset Crossover (SSX)

Probability of Crossover: 0.9

Framework: MOEA Framework

As in non-queuing model, we will be performing the GA on the following three configurations.

3.8.1 No offloading:

When applying GA with only the mobile processor configuration, there is only one offloading allocation – which is all tasks must be executed in the local processor. The results of the offloading allocation of the work-flow graph for *Figure 2* above are shown in *Table 8* below:

Table 8: External Parallel Execution with no offloading

Name	Offloading Allocation	Response Time (Sec)	Energy (Joules)	Cost (cents/min)
a_1	$d_0, d_0, d_0, d_0, d_0, d_0, d_0$	28.0	14.00	0.00

Since there is no internal parallel execution in the configuration of *no offloading*, so the result of parallel execution queueing model are same as in no queue model (*Figure 2*). The GA reveals that

response time of the queuing model for allocation a_1 is $RT_{a1} = 28\,seconds, E_{a1} = 14.0\,Joules, C_{a1} = ¢0$.

3.8.2 Single Site Offloading:

The number of available offloading allocation in parallel execution model and no queue model are same ($2^7 = 128$). However, the parallel execution model can save the queue of jobs for each processor until the processor is not available for execution. GA will start with the initial population of 100 solution and generates new solution on each iteration to find the near-optimal solution. The task t_0 is forced to be executed in the local processor as it is in the no queue model. The genetic algorithm provides three solutions after exploration. As such, the user will decide which objective is more important for their current mobile device configuration. The following *Table 9* below represents the offloading allocation suitable in the presence of only one VM.

Table 9: External Parallel Execution with single-site offloading (one VM)

Name	Offloading Allocation	Response Time (Sec)	Energy (Joules)	Cost (cents/min)
a_1	$d_0, d_0, d_1, d_1, d_1, d_1, d_1$	19.0	06.15	0.95
a_2	$d_0, d_1, d_1, d_1, d_1, d_1, d_1$	21.0	05.05	1.05
a_3	$d_0, d_0, d_0, d_0, d_0, d_0, d_0$	28.0	14.00	0.00

The above *Table 9* presents the offloading allocation in the presence of one VM. The offloading allocation a_1 provides the lowest response time which is 19 seconds. However, it consumes 6.15 joules of energy with the processing of 0.95 cents/min. The user would choose this allocation if the user is concerned about the application response time, has a decent charge on their mobile device and is willing to pay the additional cost of 0.95 cents/min. The allocation a_2 is the power saving option, which minimizes the energy with a small trade-off on the response time and the

operating cost. This allocation provides the response time to be around 19 seconds with the energy consumption to be lowest as 5.05 Joules and the operating cost is 1.05 Cents/min. The offloading allocation a_3 does not add any additional operating cost on the application execution and process all task in the local mobile device d_0. This allocation provides the response time to be 28 seconds with energy consumption of 14 Joules and no additional operating cost.

3.8.3 Multi-Site Offloading

In this configuration of a local processor with two VMs, each element of the offloading solution can have three states, 0 (local), 1 (VM_1), 2 (VM_2) so there are $3^7 = 2187$ different available offloading allocations as it is in non-queuing model. It is important to note that, in this model, each resource can store the queue for the upcoming jobs to process them in parallel. The GA starts with the initial population of 100 solutions and generates a new gene using either crossover or mutation. It observes the behavior of the new gene to explore the near-optimal solution. Like local processor with one VM configuration, the task t_0 is forced to be processed in the local processor, and all other tasks can be processed in the local processor or any available cloud server. The GA provides four different offloading allocations after the exploration of the problem. The offloading allocations are as follow:

Table 10: External Parallel Execution with multi-site offloading (two VMs)

Name	Offloading Allocation	Response Time (Sec)	Energy (Joules)	Cost (cents/min)
a_1	$d_0, d_2, d_2, d_2, d_2, d_2, d_2$	15.0	04.15	1.25
a_2	$d_0, d_0, d_1, d_1, d_1, d_1, d_1$	19.0	06.15	0.95
a_3	$d_0, d_1, d_1, d_1, d_1, d_1, d_1$	21.0	05.05	1.05
a_4	$d_0, d_0, d_0, d_0, d_0, d_0, d_0$	28.0	14.00	0.00

The *Table 10* above presents the GA suggested code offloading allocations for the work-flow graph with two VMs. The allocation a_1 provides the lowest possible response time of 15 seconds

and the lowest energy consumption of 4.15 Joules, with the trade-off of an exceptionally high additional operating cost of 1.25 cents/minute. The GA also provide allocations to the user which are average for all three objectives and the user can may use them if they are suitable for their current mobile configuration.

3.9 Evaluating a given allocation Considering both Internal and External Parallelism (for the introductory example)

In this section, we will enhance further the queuing model presented in *Example 1* for the offloading allocation $a = [d_0, d_2, d_1, d_1, d_1, d_0, d_2]$ with the implementation of internal parallel execution. We will be assuming that each available computing resource has 2 cores which can be used for parallel execution. There are three available resources for the offloading allocation *a*, the sequence of the jobs for each resource are shown in the tables below.

Table 11: Jobs scheduled in the Mobile Device d_0

Job Number	Job Type	Processor	Job Depth	Service Time (Sec)	Arrival Time (Sec)	Start Time (Sec)	End Time (Sec)
0	*execute*(t_1)	P_1	1	4	0	0	4
1	*sendJob*(t_1, t_2)	P_1	2	2	4	4	6
2	*sendJob*(t_1, t_3)	P_2	2	1	4	4	5
3	*sendJob*(t_1, t_4)	P_2	2	1	4	5	6
4	*sendJob*(t_1, t_5)	P_1	2	1	4	6	7
5	*receiveJob*(t_2, t_6)	P_1	6	2	7	7	9
6	*execute*(t_6)	P_1	7	4	9	9	13
7	*sendJob*(t_6, t_7)	P_1	8	2	13	13	15

From *Table 11*, shows the jobs schedule for the mobile device d_0. We see that the processor P_1 of mobile device d_0 handles total 6 jobs which includes one *Receive_Job*, two *Execute_Jobs*, and total

three *Sent_Jobs*. Similarly, the processor P_2 handles only two *Sent_Jobs*. Further, P_1 spends 8 seconds for *computation*, 7 seconds for *communication* but P_2 does not perform any *computation*, and only spend 2 seconds for communication. All jobs except Job number 4 are started processing upon their arrival time however, Job number 4 (*sendJob*(t_1, t_5)) arrives at the time = 4 seconds but wait in the queue for 2 seconds until any processor (P_1 *or* P_2) is available. The total time for all the jobs in the mobile device d_0 is 15 seconds.

Table 12: Jobs scheduled in the Cloud Server d_1

Job Number	Job Type	Processor	Job Depth	Service Time (Sec)	Arrival Time (Sec)	Start Time (Sec)	End Time (Sec)
0	*receiveJob*(t_1, t_3)	P_1	3	1	4	4	5
1	*receiveJob*(t_1, t_4)	P_1	3	1	5	5	6
2	*receiveJob*(t_1, t_5)	P_1	3	1	6	6	7
3	*execute*(t_3)	P_2	4	2	5	5	7
4	*execute*(t_4)	P_1	4	2	6	7	9
5	*execute*(t_5)	P_2	4	2	7	7	9
6	*sendJob*(t_3, t_7)	P_1	5	4	7	9	13
7	*sendJob*(t_4, t_7)	P_2	5	4	9	9	13
8	*sendJob*(t_5, t_7)	P_1	5	4	9	13	17

The *Table 12* above shows the jobs schedule in the Cloud Server d_1. The processor P_1 of Cloud Server d_1 throughput total 6 jobs which includes three *ReceiveJobs*, one *ExecuteJobs*, and two *SendJobs*. The processor P_1 perform 2 seconds on *computation* and 11 seconds on communication. However, the processor P_2 handles total three jobs which includes two *ExecuteJobs* and one *SendJobs*. The processor P_2 spends 4 seconds on *computation* and 4 seconds on *communication*. In server d_1, only Job number 6 (*sendJob*(t_3, t_7)) encounter the server in busy state and wait for its availability in the queue for 2 seconds. The total time the cloud server d_1 takes to process all the jobs is 17 seconds.

Table 13: Jobs scheduled in the Cloud Server d_2

Job Number	Job Type	Processor	Job Depth	Service Time (Sec)	Arrival Time (Sec)	Start Time (Sec)	End Time (Sec)
0	*receiveJob*(t_1, t_2)	P_1	3	2	4	4	6
1	*execute*(t_2)	P_1	4	1	6	6	7
2	*sendJob*(t_2, t_6)	P_1	5	2	7	7	9
3	*receiveJob*(t_3, t_7)	P_1	6	4	9	9	13
4	*receiveJob*(t_4, t_7)	P_2	6	4	9	9	13
5	*receiveJob*(t_5, t_7)	P_1	6	4	13	13	17
6	*receiveJob*(t_6, t_7)	P_2	9	2	13	13	15
7	*execute*(t_7)	P_1	10	1	17	17	18

The *Table 13* above shows the jobs scheduled in the Cloud Server d_2. The processor P_1 of Cloud Server d_2 throughput total 6 jobs which includes three *ReceiveJobs*, two *ExecuteJobs*, and only one *SendJobs.* The processor P_1 perform 2 seconds on *computation* and 12 seconds on communication. However, the processor P_2 handles only two *SendJobs* and spends 6 seconds on communication. In server d_2, all jobs start upon their arrival and no job wait in the queue for the processor availability. The total time the cloud server d_2 takes to process all the jobs is 18 seconds.

As per *Step*-3 of our algorithm, we compute:

$$RT_a = end\ time\ of\ job\ (execute(t7)) = 18\ seconds$$

The job *execute*(t_7) has the highest depth in the workflow graph thus the response time of the offloading allocation a is the end time of this job which is only 18 seconds. The energy consumption can be calculated using *equation (5).* where *computation time* is 8 seconds, *transmission time* is 9 seconds and *idle time* is only 3 seconds

$$E_a = (0.5) * 8 + (0.25) * 9 + 0.15 * (18 - 15) = 6.7\ Joules$$

The energy consumption for the offloading allocation a with external and internal parallel execution is reduced to 6.7 *Joules*. Finally, the cost can be calculated using *equation (7)*

$$C_a = 18 * \left(\frac{0.03}{60}\right) + 18 * \left(\frac{0.05}{60}\right) + 0 = 0.024\ dollars/Minutes.$$

The renting cost of cloud server d_1 is 0.03 and renting cost of cloud server d_2 is 0.05 in the model specification in section 3.3 above. Since the communication data for the offloading allocation *a* does not exceed the remaining wireless network data allowance so there D_a is 0. The total cost of the offloading allocation is 2.4 cents/ minutes.

Thus, the allocation $a = [d_0, d_2, d_1, d_1, d_1, d_0, d_2]$ with the server queue takes 18 seconds to process and it requires 6.7 Joules of energy with only the processing cost of only 2.4 cents/Minutes. The following *Table 14* below represents all three GA objectives for offloading allocation $a = [d_0, d_2, d_1, d_1, d_1, d_0, d_2]$ for no queue model (i.e. without parallelism) and for queue based model (i.e. with external parallelism).

Table 14: External and Internal Parallel Execution Gain

Offloading Allocation	Config	Response Time (Sec)	Energy Consumption (Joules)	Processing Cost (Cents/Min)
$[d_0, d_2, d_1, d_1, d_1, d_0, d_2]$	No Queue	37	9.25	4.9
	Queuing Model with External Parallel	30	8.2	4.0
	Queuing Model with Internal and External Parallel	18	6.7	2.4

Thus, the above *Table 14* shows that performing internal and external parallel execution reduces the response time to 18 seconds from 37 seconds from no queue model. Also, the energy consumption on the mobile processor is reduced to 6.7 Joules instead of 9.25 Joules from no queue model. Finally, the processing cost is reduced to 2.4 cents/minute instead of 4.9 cents/minutes from no queue model.

3.10 Near-Optimal Allocation(s) using Genetic Algorithm Considering both Internal and External Parallelism (for the Introductory Example)

Let's introduce the genetic algorithm on the above of the work flow graph presented in *example 1* above for the near-optimal offloading allocation. The genetic algorithm parameters are as follow:

Initial Population Set: 100 solutions

Number of evaluations to find the near-optimal solution: 10,000

Mutation Algorithm: Uniform Mutation (UM)

Probability of Mutation: 1/Number of Tasks = 1/7 = 0.143

Crossover Algorithm: Subset Crossover (SSX)

Probability of Crossover: 0.9

Framework: MOEA Framework

As in previous configurations, the GA will be performed on the following three configurations. To visualize the gain of internal parallel execution, we will be assigning two processor to each resource processing the following three configurations, *No_Offloading, Offloading_With_1VM, and Offloading_With_2VM.*

3.10.1 No offloading

When applying GA with only the mobile processor configuration, provides only one offloading allocation – which is all tasks must be executed in the local processor. The offloading allocation details of the task graph is shown in the *Table 15* below.

Table 15: Internal and External Parallel Execution with no offloading

Name	Offloading Allocation	Local Processor Throughput (sec)	Response Time (Sec)	Energy (Joules)	Cost (cents/min)
a_1	$d_0, d_0, d_0, d_0, d_0, d_0, d_0$	$P_{1_{Comp}} = 20$ $P_{1_{Trans}} = 0$ $P_{2_{Comp}} = 8$ $P_{2_{Trans}} = 0$	20	15.8	0

The above *Table 15* above shows that there is only one offloading allocation for the configuration of only local mobile device with two processor. The processor P_1 throughput 5 compute jobs which takes 20 seconds to complete. Similarly, processor P_2 handles only 2 jobs which takes 8 seconds to complete. Since there is no external parallel execution among resources so there are no communication jobs in this configuration. The allocation a_1 reduces the response time to 20 seconds and energy consumption to 15.8 joules with no operating cost.

3.10.2 Single Site Offloading

In this configuration, there are two available resources, local mobile device and cloud VM1 each with two available processors. The *task 0* of this allocation is forced to be executed in the local mobile device and all other resources can be assigned to any resource. The results of this configuration are shown in the *table 16* below.

Table 16: Internal and External Parallel Execution for single-site offloading (with one VM)

Name	Offloading Allocation	Local Processor Throughput	VM_1 Processor Throughput	RT (Sec)	Energy (Joules)	Cost (cents/ min)
a_1	$d_0, d_0, d_0, d_0, d_0, d_0, d_0$	$P_{1_{Comp}} = 20$ $P_{1_{Trans}} = 0$ $P_{2_{Comp}} = 8$ $P_{2_{Trans}} = 0$	$P_{1_{Comp}} = 0$ $P_{1_{Trans}} = 0$ $P_{2_{Comp}} = 0$ $P_{2_{Trans}} = 0$	20.00	15.80	0.00
a_2	$d_0, d_1, d_1, d_1, d_1, d_1, d_1$	$P_{1_{Comp}} = 4$ $P_{1_{Trans}} = 4$ $P_{2_{Comp}} = 0$ $P_{2_{Trans}} = 3$	$P_{1_{Comp}} = 8$ $P_{1_{Trans}} = 4$ $P_{2_{Comp}} = 4$ $P_{2_{Trans}} = 3$	16.00	6.90	1.60

The *Table 16* above presents two possible offloading allocation a_1 and a_2 for this configuration. The allocation a_1 does not offload any task to the VM and execute all task in the mobile device and the results are same as no offloading configuration. However, the allocation a_1 offload all task to the VM_1 except task 0 which is forced to mobile device. The mobile device throughput 1 compute and 4 sends job. The processor P_1 of mobile device spends 4 seconds on *computation* and 4 seconds on *communication* and P_2 spends on 3 seconds on *communication*, no *computation* is performed by processor P_2 of mobile device. The VM_1 throughput 6 *compute_jobs* and 4 *receive jobs*. The processor P_1 of VM_1 spends 8 seconds on *computation* and 4 seconds on *communication* and processor P_2 spends 4 seconds on *computation* and 3 seconds on *communication*. The response time, energy consumption and operating costs are 16 Seconds, 6.90 Joules and 1.60 cents/min respectively.

3.10.3 Multi-Site Offloading

In this configuration, there are two cloud servers added to the mobile device each has two processors for internal parallel execution. The offloading allocation for this configuration is shown in the *Table 17* below.

Table 17: Internal and External Parallel Execution for multi-site offloading (with two VMs)

Name	Offloading Allocation	Local Processor Throughput	VM_1 Processor Throughput	VM_2 Processor Throughput	RT (Sec)	Energy (Joules)	Cost (cents/ min)
a_1	$[d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0]$	$P_{1_{Comp}} = 20$ $P_{1_{Trans}} = 0$ $P_{2_{Comp}} = 8$ $P_{2_{Trans}} = 0$	$P_{1_{Comp}} = 0$ $P_{1_{Trans}} = 0$ $P_{2_{Comp}} = 0$ $P_{2_{Trans}} = 0$	$P_{1_{Comp}} = 0$ $P_{1_{Trans}} = 0$ $P_{2_{Comp}} = 0$ $P_{2_{Trans}} = 0$	20.00	15.80	0.00
a_2	$[d_0, d_2, d_2, d_2, d_2, d_2, d_2, d_2]$	$P_{1_{Comp}} = 4$ $P_{1_{Trans}} = 2$ $P_{2_{Comp}} = 0$ $P_{2_{Trans}} = 1.5$	$P_{1_{Comp}} = 0$ $P_{1_{Trans}} = 0$ $P_{2_{Comp}} = 0$ $P_{2_{Trans}} = 0$	$P_{1_{Comp}} = 3$ $P_{1_{Trans}} = 2$ $P_{2_{Comp}} = 3$ $P_{2_{Trans}} = 1.5$	9.50	4.60	1.58

The *Table 17* above presents the offloading allocation of offloading with two VM configuration. Like offloading with one VM, GA explore two offloading allocation a_1 and a_2. The offloading allocation a_1 perform all tasks to mobile device to reduce the operating cost as in no offloading allocation. However, offloading allocation a_2, executes all task to the VM_2 except task 0 which is forced to compute in the mobile device. The mobile device throughput 1 compute and 4 *sends job*. The processor P_1 of mobile device spends 4 seconds on computation and 2 seconds on *communication* and P_2 spends 1.5 seconds on *communication*, no *computation* is performed by P_2 of mobile device. The VM_2 throughput 6 *compute_jobs* and 4 *receive jobs*. The processor P_1 of VM_2 spends 3 seconds on *computation* and 2 seconds on *communication* and processor P_2 spends

3 seconds on *computation* and 1.5 seconds on *communication*. The response time, energy consumption and operating costs are 9.50 Seconds, 4.60 Joules and 1.58 cents/min respectively.

3.11 Summary

In the section, we will be summarizing the gain of all three configurations without queue, (Section 3.5) external parallel execution (Section 3.8) and internal and external parallel execution (Section 3.10). The *Table 18* below represents the near-optimal offloading allocation for all three configurations along with the GA objectives values. It is observed that GA objectives do not change in the *No_offloading* scenario for the *with_queue* and *external parallel execution* configurations since the execution is sequential in both scenarios. In a multi-objective offloading problem there is no global best solution, it is dependent on the user mobile status. If the user wants to run the application on performance mode, an offloading allocation which provides low response time should be chosen. Similarly, if the user wants to perform the execution in the power saving mode, they can pick the allocation with lowest response time. If the user wants to save the additional processing cost, they can use *No_Offloading* scenario.

Table 18: Summary of Results for the Introductory Example

Config	**Offloading Scenario**	**Offloading Allocation**	**RT** (Sec)	**Energy** (Joules)	**Cost** ₡/min)
No Parallel Execution Without Queue	No Offloading	$d_0, d_0, d_0, d_0, d_0, d_0, d_0$	28.0	14.00	0.00
	Single-Site Offloading	$d_0, d_0, d_0, d_0, d_0, d_0, d_0$	28.0	14.00	0.00
		$d_0, d_1, d_1, d_1, d_1, d_1, d_1$	23.0	05.55	1.15
	Multi-Site Offloading	$d_0, d_0, d_0, d_0, d_0, d_0, d_0$	28.0	14.00	0.00
		$d_0, d_2, d_2, d_2, d_2, d_2, d_2$	13.5	03.76	1.125
External Parallel Execution	No Offloading	$d_0, d_0, d_0, d_0, d_0, d_0, d_0$	28.0	14.00	0.00
	Single-Site Offloading	$d_0, d_0, d_0, d_0, d_0, d_0, d_0$	28.0	14.00	0.00
		$d_0, d_1, d_1, d_1, d_1, d_1, d_1$	21.0	05.05	1.05
		$d_0, d_0, d_1, d_1, d_1, d_1, d_1$	19.0	06.15	0.95
	Multi-Site Offloading	$d_0, d_0, d_0, d_0, d_0, d_0, d_0$	28.0	14.00	0.00
		$d_0, d_1, d_1, d_1, d_1, d_1, d_1$	21.0	05.05	1.05
		$d_0, d_0, d_1, d_1, d_1, d_1, d_1$	19.0	06.15	0.95
		$d_0, d_2, d_2, d_2, d_2, d_2, d_2$	15.0	04.15	1.25
External and Internal Parallel Execution	No Offloading	$d_0, d_0, d_0, d_0, d_0, d_0, d_0$	20.0	15.80	0.00
	Single-Site Offloading	$d_0, d_0, d_0, d_0, d_0, d_0, d_0$	20.0	15.80	0.00
		$d_0, d_1, d_1, d_1, d_1, d_1, d_1$	16.0	6.90	1.60
	Multi-Site Offloading	$d_0, d_0, d_0, d_0, d_0, d_0, d_0$	20.0	15.80	0.00
		$d_0, d_2, d_2, d_2, d_2, d_2, d_2$	9.50	4.600	1.58

Chapter 4: Case Study

In this chapter we will analyze a real-world face recognition application problem to answer the following questions:

- Does consideration of parallel execution of different tasks of an application while solving the offloading allocation problem influence the near-optimal solution?
- What is the effect of multi-core devices on the near-optimal solution of the offloading allocation problem?

4.1. Mobile Application Specification

In this section, the proposed code offloading framework will be evaluated for external and internal parallel execution using a real-world face recognition based on the call graph presented in [27] shown in *Figure 4* below

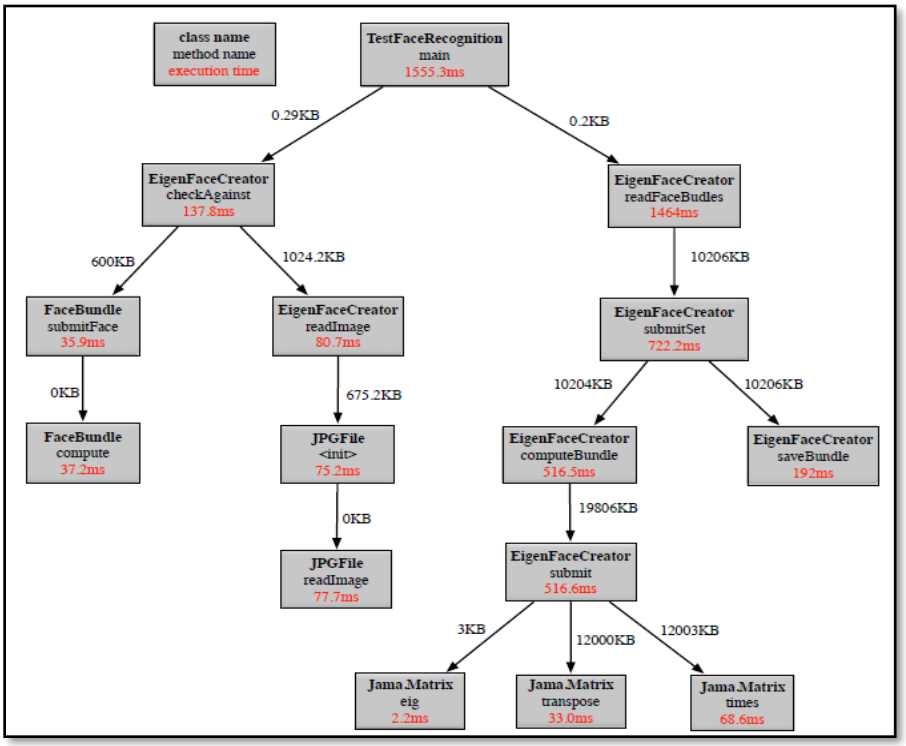

Figure 4: Call graph of the face recognition application

The *Figure 4* above represents the call graph of the face recognition application which is built upon an open source code to implement the Eigen face recognition algorithm [27]. The call graph is structured by analyzing the application with Soot tool and building a network and energy profiler. Each step in the call graph has two bold lines where the first line represents the class name and bottom bold line with colon represents the method name of that class for the application. The execution time for each step is presented in ms (milli-seconds) and the data transfer between two steps is presented in KB (killo-bytes). For the analysis of our project, we convert, the call graph into a work-flow graph which is shown in *Figure 5* below.

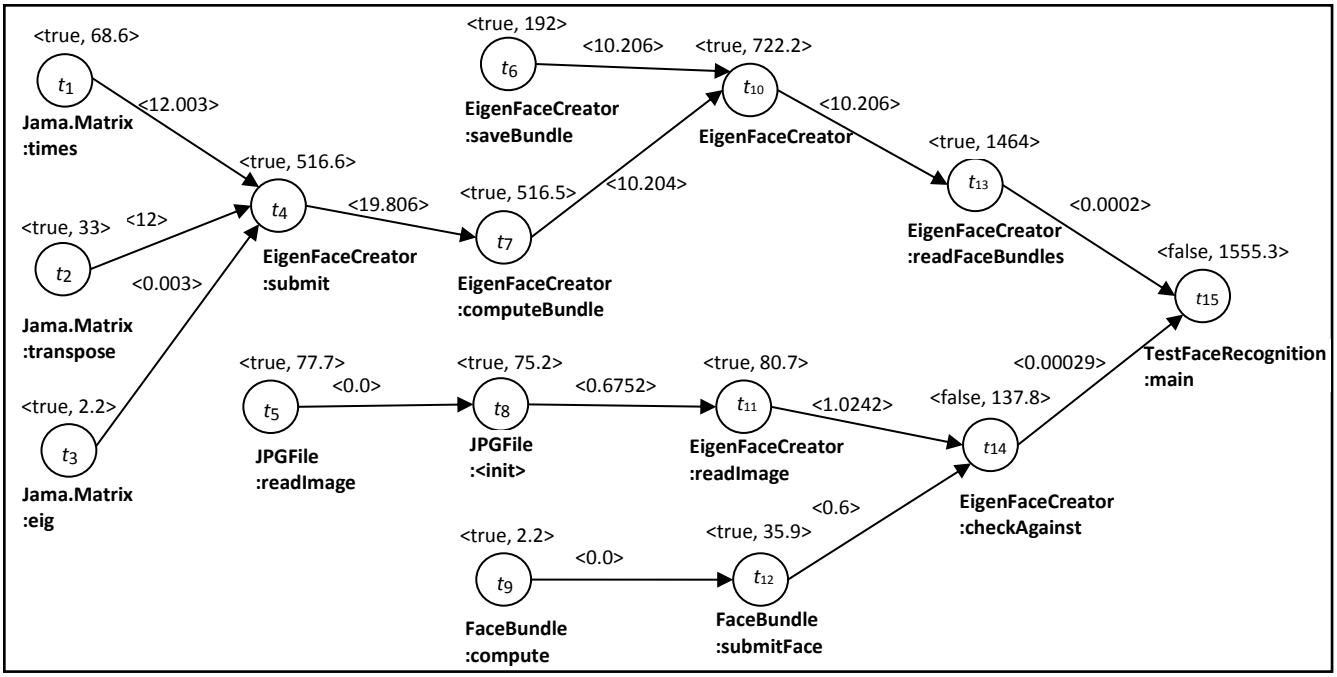

Figure 5: Workflow - Graph of a Face Recognition Application

The *Figure 5* above represents the work-flow graph of the face recognition example from [27]. It has total 15 tasks $[t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, t_9, t_{10}, t_{11}, t_{12}, t_{13}, t_{14}, t_{15}]$ which must be executed to complete the application. Each task t_i, in the graph below is represented with a tuple of two $<o_i, w_i>$ where o_i is the type (*true* for *offloadable*, *false* for *non-offloadable*) of t_i and w_i is

the amount of CPU cycles MI (in million instructions) required for execution of task t_i. The $<w_{ij}>$ for each edge $e(t_i, t_j)$ is the amount of data (in MB) that needs to be transferred between the tasks t_i and t_j for communication. The communication data transfer is zero if the tasks t_i and t_j are executed on the same device. The execution times w_i for each task t_i where $i < 0 < N$ of the application is converted in MI from ms. we assume that the mobile device has one core with processing speed of 1000 MIPS. Using this assumption, we convert all tasks execution time from milliseconds to million instructions. As an example, the conversion of the task t_1 (Jama.Matrix :time) which is has 68.6ms execution time will be as follow.

$$execution_{Instructions} = 68.6\ ms * 1000\,\frac{MI}{s} = 68.6\ MI$$

Thus, the execution of task t_1 will requires 68.6 million instructions to process. All tasks are independent and have the ability to process in any resource except task t_{14} and t_{15} (main and check against). These two tasks are not offloadable and they are forced to be processed in the local mobile processor. Based on the above conversion, we simplify the model as shown in *Figure 6* below, for the offloading allocation [d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0] (all tasks are allocated to the local mobile device).

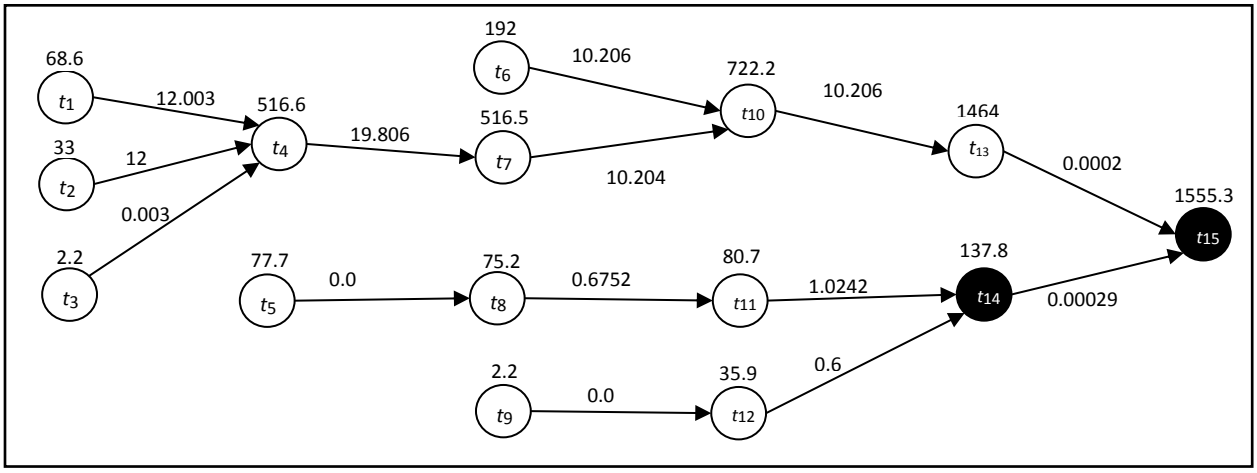

Figure 6: Simplified Work-Flow graph of Face Recognition Application for offloading allocation [d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0]

The *Figure 6* above represents the simplified work-flow graph of the face recognition application. The task which are not offloadable are represented in black specifically, task t_{14} and t_{15} in *Figure 6* above.

4.2. Model Specification

In this section, the specification about mobile device, mobile user profile, cloud servers, GA configurations and system configuration will be specified.

4.2.1. Mobile Device:

The mobile device d_0 is modeled as a tuple of five $< b_0, n_0, s_0, pc_0, pd_0, pi_0 >$ where b_0 is the current battery percentage of the mobile device, n_0 is the number of processors in the mobile device, and for each processor s_0 is the processing speed of that processor (in million instructions per second), pc_0 is the computation power consumption, $pd_0$is the power consumption for communication(send and receive data), and pi_0 is the power consumption while the device is idle.

$< b_0, n_0, s_0, pc_0, pd_0, pi_0 >$ = <10%, 1 core, 1000MIPS, 0.9W, 1.3W, 0.3W>

For the analysis it is assumed that the mobile device d_0 is currently at 10% and there are total 4 available cores for the execution. The number of instruction that can be processed in each core is 1000 MIPS. The computation energy, communication energy and idle energy is 0.9W, 1.3W and 0.3W respectively.

4.2.2. Mobile User Profile:

The mobile user profile is the specification the user network package which is represented as tuple of two $\langle \rho, \alpha \rangle$ where ρ is the current remaining amount of data left from the fixed portion and α is the monetary rate for the additional amount of data (in dollars per MB).

$$\langle \rho, \alpha \rangle = \langle 1024\ MB, 0.3\ dollars/MB \rangle$$

For the analysis of this project, we assume ρ to be 1024 MB and α to be 0.3 dollars per MB.

4.2.3. Cloud Server d_1:

The specification of the cloud server d_1 is represented as a tuple of three $< n_c,\ s_c,\ r_c >$ where n_c is the number of available cores in the cloud server, s_c is the processing speed of each core (in million instructions per second), and r_c is the monetary rate of renting the cloud server from the cloud provider (in dollars per minute).

$$< n_c,\ s_c,\ r_c > = < 4\ core, 2000MIPS, 0.6\ dollars/min >$$

Thus, the cloud server d_1 has 4 available cores and each core is twice as fast as the local mobile processor which can process 2000 million instructions per minute with the total renting cost of 0.6 dollars / mins.

4.2.4. Cloud Server d_2:

Similarly, the Cloud Server d_2 is also represented as a tuple of three$< n_c,\ s_c,\ r_c >$. with the following specifications:

$$< n_c,\ s_c,\ r_c > = < 4\ core, 4000MIPS, 0.6\ dollars/min >$$

The cloud server d_2 has 4 available cores and each core is 4 times as fast as the mobile processor which has the ability to process 4000 million instructions per minute with the total renting cost of 1.2 dollars / mins

4.2.5. Device to Device Bandwidth:

In the analysis of our framework, we consider the maximum resource availability of two cloud servers and the network bandwidth between any two resources is as follow:

$$bandwidth(d_0, d_1) = 1MBPS$$

$$bandwidth(d_0, d_2) = 1MBPS$$

$$bandwidth(d_1, d_2) = 1MBPS$$

Thus, the network bandwidth between local mobile device d_0 and cloud server d_1 as well as cloud server d_2 is 1MBPS. Similarly, the network bandwidth between two cloud resources d_1 and d_2 is also 1MBPS.

4.2.6. Genetic Algorithm Configurations:

The GA is designed based on NSGA-II optimization algorithm which introduces fast non-dominated sorting and uses more computation efficient crowding distance metric during survival selection as compare to NSGA. [15]. The NSGA-II has the ability of binary tournament selection with Pareto dominance and crowding distance, subset crossover and uniform mutation operators. We apply subset crossover to apply crossover on the GA solutions and uniform mutation to mutate GA solution. The probability of applying the subset crossover operator to a decision variable is 0.9 and the probability of applying the uniform mutation operator to a decision variable is 1 /

(number of decision variables, i.e. number of tasks) = 1/15. The initial population size GA solutions is 1000 solution. The GA performs 100,000 evaluations of solutions with the set probability of crossover and mutation to find the near-optimal solution. The GA is set to optimize three objective function, response time, energy consumption and processing cost.

4.2.7. System Configurations:

The evaluations of all configuration were performed on Windows 10 (Redstone 4) operating system. The hardware consists of Intel Core i7-6800k CPU with total 6 cores and processor base frequency of 3.4 GHz. The system has 16 GB of total Random-Access Memory (RAM).

4.3. Results and Discussion:

In this section, based on the above mobile face recognition application (section 4.1) and the model specifications presented in section 4.2, each code offloading sub-set will be evaluated on the following three cases to observe the effect of external parallel execution.

- *No-offloading*:

  In this case, we assume that there is no cloud server available for computation offloading. Thus, all the tasks must execute locally in the mobile device d_0.

- *Single-Site offloading*:

  In this case, we assume that there is one cloud server d_1 available for computation offloading so each task which is offloadable can be executed either in mobile device d_0 or the cloud server d_1.

- *Multi-Site offloading*:

  In this case, we assume that there are two cloud servers d_1 and d_2 available for computation offloading. Thus, the execution of all offloadable tasks can take place in any of the available resource, d_0, d_1 and d_2.

Similarly, the effect of internal parallel execution will be represented through the following three configurations:

- *1-Core in each computing resource*:

  It is assumed that all available resources can only perform the execution in only 1-core of the processor.

- *2-core in each computing resource*:

  In this configuration, the processor can perform the internal parallel execution among only two cores.

- *4-core in each computing resource*

  This configuration allows the processor to divide the work-load among 4-core to reduce the throughput time of the applications tasks.

4.3.1. Including or excluding parallel execution to find the near-optimal offloading allocation

This section tries to answer the question: *Does consideration of parallel execution of different tasks of an application while solving the offloading allocation problem influences the near-optimal solution?* We assume every device has one processing core. We therefore consider only external parallelism while comparing parallel versus non-parallel execution in this section. In general, for each evaluation, GA optimizes the problem based on three objectives, reduced response time, and energy consumption with no additional processing cost. For each solution, GA produces pareto-optimal results for each of the application objectives. It is up to the user to pick the right pareto-optimal solution based on the current mobile device configuration. In this research will be looking at all three objectives individually.

4.3.1.1. Response Time:

The response time relates to the performance of the application and optimization of response time refer to reducing the time required to process all the task of the application. The pareto-optimal solution for the total reduced response time on all three cases is shown in *Table 19* below.

Table 19: Near-optimal Solution and Minimum Corresponding Response Time

Case	**Considering Parallel exec. of tasks** <minimum response time> [near-optimal solution, i.e. the near-optimal offloading allocation of 15 tasks on available devices]	**Without Considering Parallel exec. of tasks** < minimum response time> [near-optimal solution, i.e. the near-optimal offloading allocation of 15 tasks on available devices]
No-offloading	<5.5149 Sec> [d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0]	<5.5149 Sec> [d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0]
Single-site offloading	<3.3130 Sec> [d_1,d_1,d_1,d_1,d_0,d_1,d_1,d_0,d_0,d_1,d_0,d_0,d_1,d_0,d_0]	<3.7001 Sec> [d_1,d_1,d_1,d_1,d_1,d_1,d_1,d_0,d_1,d_1,d_0,d_0,d_1,d_0,d_0]
Multi-Site offloading	<2.4342 Sec> [d_2,d_2,d_2,d_2,d_1,d_2,d_2,d_0,d_1,d_2,d_0,d_0,d_2,d_0,d_0]	<2.7925 Sec> [d_2,d_2,d_2,d_2,d_2,d_2,d_2,d_0,d_2,d_2,d_0,d_0,d_2,d_0,d_0]

Table 19 above shows the application response time based pareto-optimal offloading solutions (i.e. the offloading allocation of the fifteen tasks on the computing devices) for the three cases *No-offloading*, *Single-Site offloading*, and *Multi-Site offloading*. Each case is evaluated while considering versus not considering the parallel execution of tasks. The *Table 19* above represents

that the *No-Offloading* case for both scenarios of considering or not considering parallel execution, the minimum response time is same 5.5149 Seconds. This is because in *No-offloading* case, all the tasks must be allocated in the mobile device and that the device consists of only one core forcing the tasks to execute sequentially. Thus, no parallel execution is performed in both scenarios.

In *Single-Site offloading* case, the external parallel execution provides the gain of 11.68% compare to sequential implementation which reduces the application response time to 3.3130 Seconds from 3.7001 Seconds. The GA offloading solution for the single-site parallel execution is [d_1, d_1, d_1, d_1, d_0, d_1, d_1, d_0, d_0, d_1, d_0, d_0, d_1, d_0, d_0] and [d_1, d_1, d_1, d_1, d_1, d_1, d_1, d_0, d_1, d_1, d_0, d_0, d_1, d_0, d_0] is offloading solution for not considering any parallel execution among local mobile device d_0 and cloud server d_1.

Similarly, for *Multi-Site offloading* case, the external parallel execution provides the gain of 12.83% by reducing the application response time to 2.4342 Seconds as compare to 2.7925 Seconds for the sequential execution. The offloading solution to achieve the minimum response time with parallel execution among different available resources is [d_2, d_2, d_2, d_2, d_1, d_2, d_2, d_0, d_1, d_2, d_0, d_0, d_2, d_0, d_0]. Similarly, the offloading solution for not considering parallel execution and performing all tasks in sequential manners is [d_2, d_2, d_2, d_2, d_2, d_2, d_2, d_0, d_2, d_2, d_0, d_0, d_2, d_0, d_0].

In conclusion, the two cases *Single-Site* and *Multi-Site offloading*, provides a gain of 11.68% and 12.83% respectively for considering external parallel execution among different resources as compare sequential execution. It is also observed that the task allocation for both cases is different in both scenarios.

4.3.1.2. Energy Consumption:

The energy consumption is aligned with the power saving mode of the application. It is suitable for the scenarios when the mobile device current battery percentage b_0 is low and user wants the application to consume as less battery as possible. As it is mentioned in the section 4.2.1 (mobile device) the current battery level is assumed to be 10 % remaining so the user chooses this option to make the battery lasts longer. The near-optimal solution for the optimized energy consumption objective is shown in the *Table 20* below.

Table 20: Near-optimal Solution for Energy Consumption

Case	**Considering Parallel exec. of tasks** <minimum energy consumption> [near-optimal solution]	**Without Considering Parallel exec. of tasks** < minimum energy consumption> [near-optimal solution]
No-offloading	<4.9634 J> $[d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0]$	<4.9634 J> $[d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0]$
Single-site offloading	<2.1422 J> $[d_1,d_1,d_1,d_1,d_1,d_1,d_0,d_1,d_1,d_0,d_0,d_1,d_0,d_0]$	<3.8746 J> $[d_1,d_1,d_1,d_1,d_1,d_1,d_1,d_0,d_1,d_1,d_0,d_0,d_1,d_0,d_0]$
Multi-Site offloading	<1.8614 J> $[d_2,d_2,d_2,d_2,d_1,d_2,d_2,d_0,d_1,d_2,d_0,d_0,d_2,d_0,d_0]$	<2.7856 J> $[d_2,d_2,d_2,d_2,d_2,d_2,d_2,d_0,d_2,d_2,d_0,d_0,d_2,d_0,d_0]$

The *Table 20* above represents the application energy consumption based pareto-optimal offloading allocations for the scenarios when the parallel execution is performed among computation resources verses sequential execution. The energy consumption for the *no-offloading* case, is same *4.9634 Joules* for both scenarios since there is only one resource (local mobile device d_0) with only one available core. Thus, there is no external or internal parallel execution is performed in the case of *no-offloading*.

In *Single-site offloading* case, the parallel execution consumes 44.71% less energy as compare to no parallel execution among different resources. The energy consumption for the parallel execution scenario is *2.1422 Joules* as compare to *3.8746 Joules* when the parallel execution is ignored. The near-optimal offloading allocation for the application tasks in parallel execution and

sequential implementation scenario is same for the energy consumption which is [d_1, d_1, d_1, d_1, d_1, d_1, d_1, d_0, d_1, d_1, d_0, d_0, d_1, d_0, d_0].

Similarly, in *Multi-Site offloading*, the parallel execution scenario consumes 33% less energy as compare to sequential implementation. It is also observed, the GA near-optimal solution for the *Multi-Site offloading* case is different in both scenarios. The near-optimal offloading allocation with parallel execution is [d_2, d_2, d_2, d_2, d_1, d_2, d_2, d_0, d_1, d_2, d_0, d_0, d_2, d_0, d_0] which consumes total energy of 1.8614 Joules as compare to sequential implementation of 2.7856 Joules of energy consumption from allocation [d_2, d_2, d_2, d_2, d_2, d_2, d_2, d_0, d_2, d_2, d_0, d_0, d_2, d_0, d_0].

Thus, the two scenario reveals that if the user current mobile device's battery percentage is low, the user should consider parallel execution to consume less power during execution. The *Single-Site offloading*, and *Multi-Site offloading* consumes 44.71 % and 33.00 % less power as compare to not considering parallel execution.

4.3.1.3. Monetary Cost:

Cost with Response Time:

The processing cost is incurred by the mobile user can also be chosen as the objective function. However, the minimum monetary cost (zero dollars) intuitively refers to the *No-offloading* case. Thus, the GA optimization for any scenario for this object always leads towards the *No-offloading* case. In order to solve the near-optimal offloading allocation problem in terms of monetary cost as the objective function makes sense *only when* other objectives (such as response time, energy

consumption) are also considered as well. Let's consider the processing cost with respect to response time and energy consumption individually.

The processing cost is any additional cost which is required to achieve the near-optimal response time and energy consumption. The *Table 21* below represents the cost objective with respect to response time.

Table 21: Cost Objective with respect to Response Time

Case	**Considering Parallel exec. of tasks** <minimum response time> [near-optimal solution, i.e. the near-optimal offloading allocation of 15 tasks on available devices]	**Cost (Cents)**	**Without Considering Parallel exec. of tasks** < minimum response time> [near-optimal solution, i.e. the near-optimal offloading allocation of 15 tasks on available devices]	**Cost (Cents)**
No-offloading	<5.5149 Sec> [d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0]	0.0¢	<5.5149 Sec> [d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0]	0.0¢
Single-site offloading	<3.3130 Sec> [d_1,d_1,d_1,d_1,d_0,d_1,d_1,d_0,d_0,d_1,d_0,d_0,d_1,d_0,d_0]	3.3¢	<3.7001 Sec> [d_1,d_1,d_1,d_1,d_1,d_1,d_0,d_1,d_1,d_0,d_0,d_1,d_0,d_0]	3.7¢
Multi-Site offloading	<2.4342 Sec> [d_2,d_2,d_2,d_2,d_1,d_2,d_2,d_0,d_1,d_2,d_0,d_0,d_2,d_0,d_0]	4.9¢	<2.7925 Sec> [d_2,d_2,d_2,d_2,d_2,d_2,d_2,d_0,d_2,d_2,d_0,d_0,d_2,d_0,d_0]	5.6¢

Based on the *Table 21* above, it is safe to conclude that parallel execution of the tasks does not just improve the response time of the application. It also reduces the processing cost of the near-optimal solution. The cost for no offloading case is same 0.0¢ for both scenarios. However, the single site offloading case requires 10.81% less cost for the parallel execution scenario as compare to sequential execution. Similarly, the percentage difference of cost for *Multi-Site offloading* case is 12.5% so the parallel execution of task will incur 12.5% less operating cost as compare to not considering parallel execution.

Cost with Energy Consumption:

In this section, the relationship between operating cost and the application energy consumption will be discussed. The required operating cost with respect to mobile device energy consumption is shown the *Table 22* below.

Table 22: Near-optimal Energy Consumption with Cost

Cases	**Considering Parallel exec. of tasks** <minimum energy consumption> [near-optimal solution]	**Cost (Cents)**	**Without Considering Parallel exec. of tasks** < minimum energy consumption> [near-optimal solution]	**Cost (Cents)**
No-offloading	<4.9634 J> [d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0]	0.0¢	<4.9634 J> [d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0]	0.0¢
Single-Site offloading	<2.1422 J> [d_1,d_1,d_1,d_1,d_1,d_1,d_1,d_0,d_1,d_1,d_0,d_0,d_1,d_0,d_0]	3.4¢	<3.8746 J> [d_1,d_1,d_1,d_1,d_1,d_1,d_1,d_0,d_1,d_1,d_0,d_0,d_1,d_0,d_0]	3.7¢
Multi-site offloading	<1.8614 J> [d_2,d_2,d_2,d_2,d_1,d_2,d_2,d_0,d_1,d_2,d_0,d_0,d_2,d_0,d_0]	7. 3¢	<2.7856 J> [d_2,d_2,d_2,d_2,d_2,d_2,d_2,d_0,d_2,d_2,d_0,d_0,d_2,d_0,d_0]	5. 6¢

Based on *Table 22*, one can easily visualize the behavior of cost and energy consumption for parallel execution and non-parallel execution. Similar to the response time, the no-offloading case does not require any additional operating cost since all tasks are processed in the local processor. Further, the gain of parallel execution in single-site offloading is 8.108 % as compare to no parallel execution scenario. Finally, the parallel execution for the *Multi-Site offloading* case requires 23.287% more operating cost as compare to no-offloading scenario.

Hence, it is safe to conclude that parallel execution of the application tasks optimizes all three objectives (response time, energy consumption and cost) as compare to sequential execution.

4.3.2. Evaluating the effect of multi-core devices on near-optimal offloading allocation:

This section tries to answer the question: *What is the effect of multi-core devices on the near-optimal solution of the offloading allocation problem?* We consider both internal and external parallelism here. In order to visualize the effect of internal parallel execution, we extend the three cases of (*No-offloading, Singe-Site offloading, Multi-Site offloading*) with three more configurations 1-core, 2-core and 4-core of the processor. The renting rate of any cloud server d_i (i = 1, 2) with m cores is $m * r_i$ where r_i is the renting rate of d_i with one core. Similar to external parallel execution (section 4.3.1) there is not a single best solution that minimizes all the three objectives at the same time since a small improvement in one objective may deteriorate at least

one other objective [8]. Instead, we will have a Pareto-optimal set of solutions. Pareto optimality considers a solution to be better or worse in comparison to another solution only if it is better with respect to all objectives or worse with respect to all objectives. Any two solutions are non-dominated if neither dominates the other, i.e. neither one is better than the other. The set of all non-dominated solutions is captured by the Pareto-optimal set of solutions [15]. For each pair of case (*Case-1, Case-2* and *Case-3*) and core (*1-core, 2-cores* and *4-cores*), each pareto-optimal set contains a **bold** value representing the minimum value of an objective function among the solutions. For example, corresponding to the Pareto-optimal set for the case-core pair (*Case-1, 1-core*), the solution 11 yields the minimum energy consumption of 4.9634 Joules, the solution 12 yields the minimum response time of 4.8431seconds with no additional cost, we will be looking at three cases (*No-offloading, Singe-Site offloading, Multi-Site offloading)* individually to observe the gain of internal parallel execution.

4.3.2.1.Case1: No Offloading:

In this section, the gain of internal parallel execution on case 1 (No offloading) will be analyzed. In case1, there is only local processor available for execution and all tasks are forced to process in the local processor resulting no additional processing cost. The only one possible solution for this case is solution [d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0] all task are forced to process in the local mobile device. However, varying the number of processor cores introduces internal parallel execution which makes a significant improvement on the response time with the trade-off of energy consumption. The GA pareto-optimal solutions for the case1- no offloading are shown in the *Table 23* below.

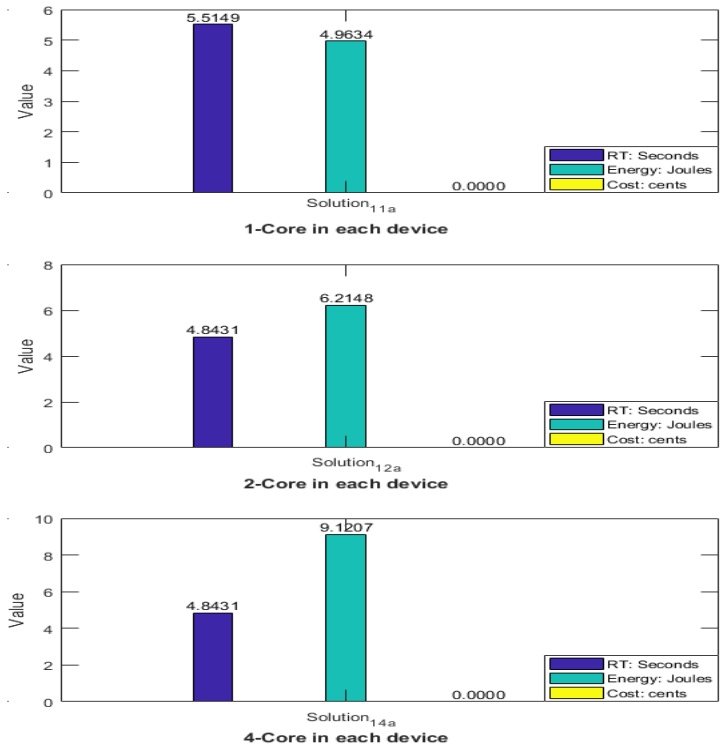

Table 23:Offloading Allocation for No-Offloading

1-core in each device	2-cores in each device	4-cores in each device
11a[d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0]	**12a**[d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0]	**14a**[d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0]

Figure 7: Internal Parallel Execution for No-Offloading

The *Figure 7* above shows the pareto-optimal solution for the case 1-No offloading for 1-core, 2-core and 4-core labelled as 11a, 12a, and 14a. For the 1-core processing, there is no internal parallel execution and all tasks are executed sequentially which results in *5.5419 Seconds* of response time with the energy consumption of *4.9634 Joules* without adding any additional processing cost. Similarly, for the 2-core processing, the response time and energy consumption is *4.8431 Seconds* and *6.2148 Joules* respectively which shows the gain of 12.18% in response time with the trade-off of 25.12% more energy consumption as compare to 1 core processing scenario. It is observed in the *Figure 7* above that the response time for the 2-core processing reaches its near-optimal

value and does not improve any further by adding any additional core for the face recognition example. Hence the 4-core processing yields the same response time as it is in 2 core processing *4.8431 Seconds*. However, the energy consumption value increases for any 4-core processing scenario. The energy loss is due to processor being in idle state when there is not enough application task available. Thus, for the no offloading case the energy consumption is minimum for the 1-core processing and the response time is near-optimal for the 2-core processing and there is no additional processing cost.

4.3.2.2.Case 2: Single Site offloading:

In case2, Single Site offloading, there are two available resources, local mobile processor one and cloud server. In this case, all tasks except t_{14} and t_{15} can be either offloaded to the cloud server or to be process in the local mobile device. The pareto-optimal solutions for the case2- single site offloading scenario are shown in the *Table 24* below.

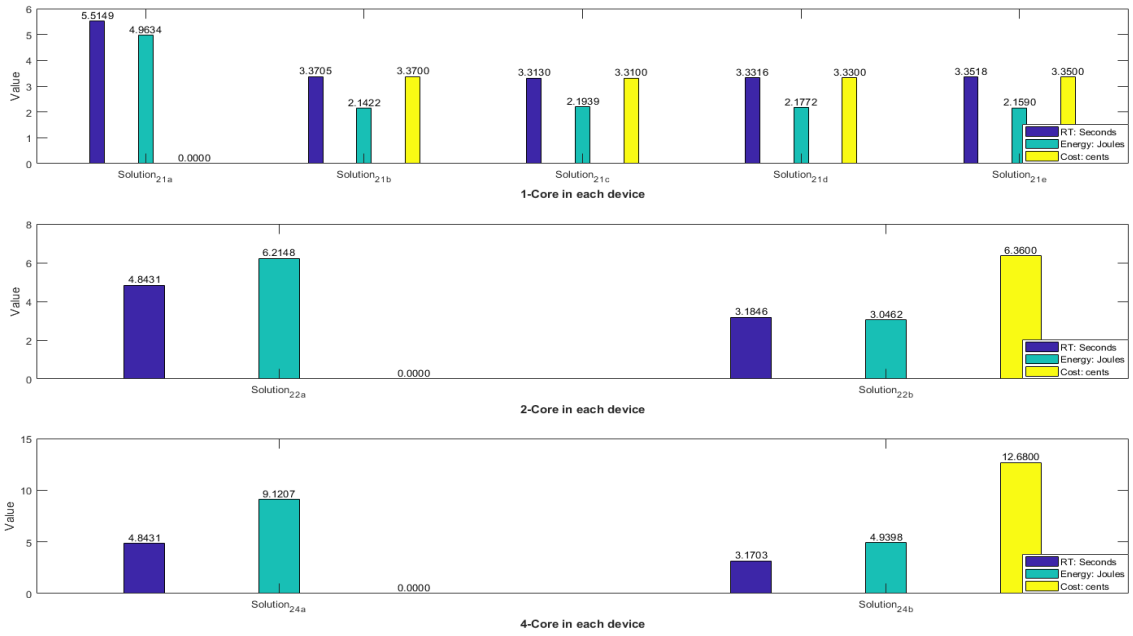

Table 24:Offloading Allocation for Single-Site Offloading

1-core in each device	2-core in each device	4-core in each device
21a.[d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0]	**22a.**[d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0]	**24a.**[d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0]
21b.[d_1,d_1,d_1,d_1,d_1,d_1,d_1,d_1,d_0,d_1,d_1,d_0,d_0,d_1,d_0,d_0]	**22b.**[d_1,d_1,d_0,d_1,d_1,d_1,d_1,d_0,d_1,d_1,d_0,d_0,d_1,d_0,d_0]	**24b.**[d_1,d_1,d_0,d_1,d_1,d_1,d_1,d_0,d_1,d_1,d_0,d_0,d_1,d_0,d_0]
21c.[d_1,d_1,d_1,d_1,d_0,d_1,d_1,d_0,d_0,d_1,d_0,d_0,d_1,d_0,d_0]		
21d.[d_1,d_1,d_1,d_1,d_0,d_1,d_1,d_0,d_1,d_1,d_0,d_0,d_1,d_0,d_0]		
21e.[d_1,d_1,d_1,d_1,d_1,d_1,d_1,d_0,d_0,d_1,d_0,d_0,d_1,d_0,d_0]		

Figure 8: Internal Parallel Execution for Single-Site Offloading

The *Figure 8* above shows the pareto-optimal solutions for single site offloading when executed in 1-core, 2-core and 4-core configuration. Let's analyze each configuration individually to conclude the gain of internal parallel execution.

4.3.2.2.1. 1-Core each Resource:

The execution of 1 core in both resources yields 5 pareto-optimal solution labeled as 21a, 21b, 21c, 21d, 21e in the *Table 24* above. There is no internal parallel execution in this configuration, however, all solutions except 21a, utilizes the external parallel execution between the mobile device and cloud server. There is no best solution, it is up to the user to pick the solution more suitable for

their current situation. As an example, if the mobile battery is at 100% and the user is willing to pay 3.31¢ on the processing cost then the best option for the user to pick 21c. [d_1, d_1, d_1, d_1, d_0, d_1, d_1, d_0, d_0, d_1, d_0, d_0, d_1, d_0, d_0] which provides the minimum response time of *3.3130 Seconds* with the energy consumption of *2.1939 Joules* and the additional processing cost of 3.31¢. However, if the user main goal is to save the battery consumption as much as possible regardless of the additional processing cost or the application response time, the best option for the user would be 21b [d_1, d_1, d_1, d_1, d_1, d_1, d_1, d_0, d_1, d_1, d_0, d_0, d_1, d_0, d_0] which yields the minimum energy consumption of *2.1422 Joules* with the response time to be *3.3705 Seconds* with the processing cost of 3.35¢. Further, if the main goal is to minimize the response time as well as the energy consumption then the user has to decide between 21d and 21e depending on the additional processing cost which the user wants to pay.

4.3.2.2.2. 2-Core each Resource:

In this scenario, each computing resource has the ability to perform internal parallel execution among both cores of the processors as well as the external parallel execution between both resources. Each core of the processor can handle the computation or communication of application tasks jobs depending on their arrival or the processor availability. The GA reveals there are two pareto-optimal solutions in this scenario labelled as 22a and 22b in the *Table 24* above. The solution 22a is pareto-optimal for the cost objective and provides the user an option to process all tasks of the application without adding any additional processing cost. However, the solution 22b provides the minimum response time and energy consumption of *3.1846 Seconds* and *3.0462 Joules* respectively with an additional processing cost of 6.36¢. In this scenario, the response time is further enhanced by 3.88% as compare to minimum response time of the pareto-optimal solution

in 1-core scenario (21b). However, energy consumption increases by 42.20% as compared to minimum energy consumption of the pareto-optimal solution (21b) in 1-core scenario. The additional processing cost of the application also increases to 6.36¢ as compare 1-core pareto-optimal solution (21b) which requires the maximum processing cost.

4.3.2.2.3. 4-Core each Resource:

In this scenario, each computing resources has the ability to execute application tasks parallel among the different processors (internal parallel execution). Each processor can perform computation or communication based on their availability and the application tasks jobs arrival. Similar to 2-core scenario, the GA also reveals to 2 pareto-optimal solution in this case labelled as 24a and 24b in the *Table 24* above. This case further enhanced the application response time with the trade-off of energy consumption and processing cost. The solution 24a provides the minimum operating cost the application however, the solution 24b provides the response time of *3.1703 Seconds* with the energy consumption of *4.9398 Joules* and the additional processing cost of 12.68¢. In this case, the gain of the response time is just 0.45% however, the trade-off of energy consumption and cost is 38.33 % and 49.84% respectively as compare to 2-core scenario. Since the loss of energy consumption and cost is much higher than the gain in the response time so it is safe to conclude that this case is only for those users whose main goal is to reduce the application response time without prioritizing the energy consumption or the additional processing cost of the application.

4.3.2.3.Case 3: Multi-Site Offloading:

In this case, there are three computation resources, local mobile device d0 and cloud server d1 and d2. In order to observe the gain of the parallel execution, each resource is executed in 1-core, 2-core, and 4-core configuration. We will be looking at each configuration individually. The *table*

25 below shows the results of the three objectives (response time, energy consumption and cost) for all three configurations (1-core, 2-core, 4-core).

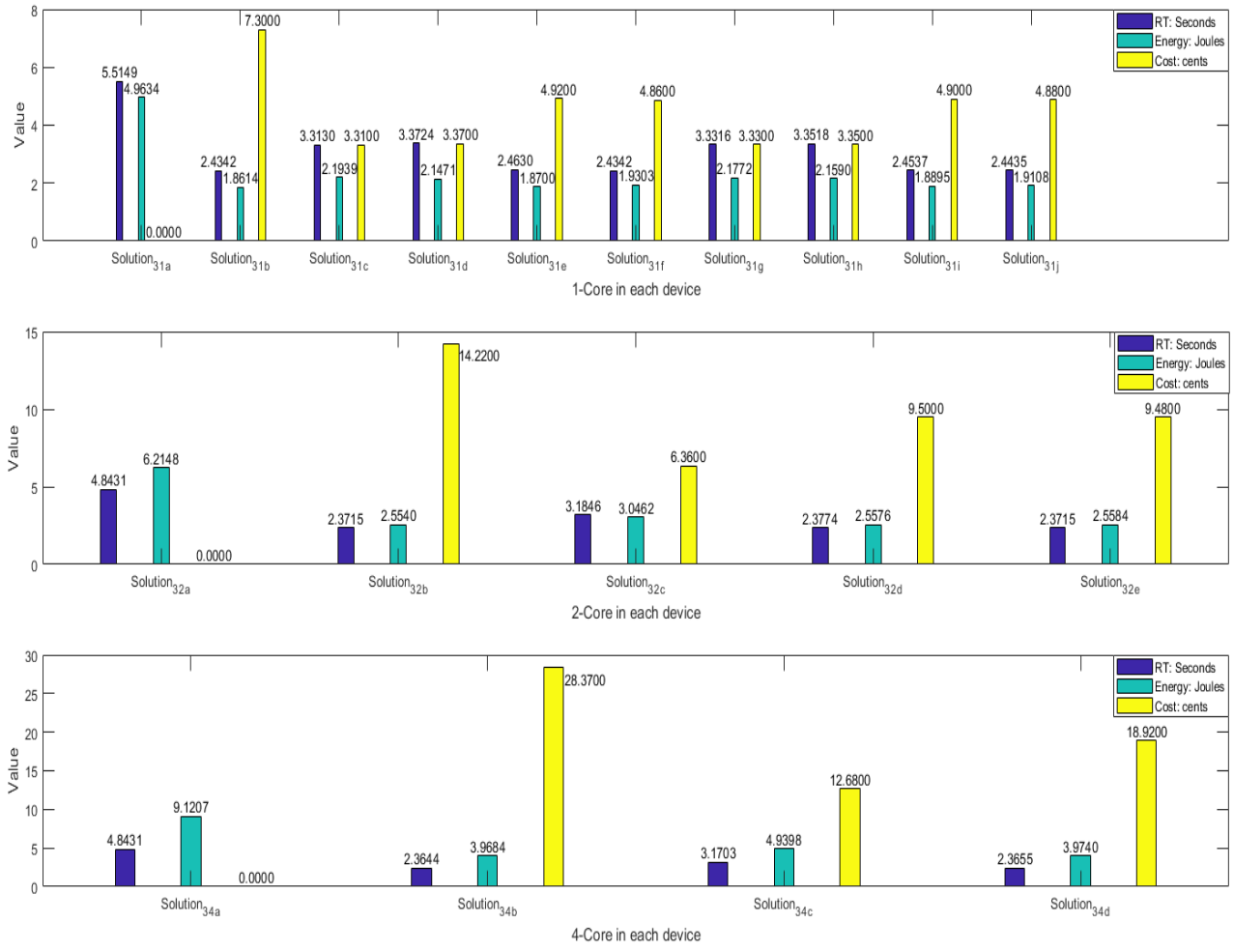

Table 25: Offloading Allocation for Multi-Site Offloading

1-core in each device	2-core in each device	4-core in each device
31a.[d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0]	**32a.**[d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0]	**34a.**[d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0,d_0]
31b.[d_2,d_2,d_2,d_2,d_2,d_1,d_2,d_2,d_0,d_1,d_2,d_0,d_0,d_2,d_0,d_0]	**32b.**[d_2,d_2,d_1,d_2,d_2,d_2,d_2,d_0,d_1,d_2,d_0,d_0,d_2,d_0,d_0]	**34b.**[d_2,d_2,d_1,d_2,d_1,d_2,d_2,d_0,d_2,d_2,d_0,d_0,d_2,d_0,d_0]
31c.[d_1,d_1,d_1,d_1,d_0,d_1,d_1,d_0,d_0,d_1,d_0,d_0,d_1,d_0,d_0]	**32c.** [d_1,d_1,d_0,d_1,d_1,d_1,d_1,d_0,d_1,d_1,d_0,d_0,d_1,d_0,d_0]	**34c.**[d_1,d_1,d_0,d_1,d_1,d_1,d_1,d_0,d_1,d_1,d_0,d_0,d_1,d_0,d_0]
31d.[d_1,d_1,d_0,d_1,d_1,d_1,d_1,d_0,d_1,d_1,d_0,d_0,d_1,d_0,d_0]	**32d.**[d_2,d_2,d_2,d_2,d_2,d_2,d_2,d_0,d_2,d_2,d_0,d_0,d_2,d_0,d_0]	**34d.**[d_2,d_2,d_0,d_2,d_2,d_2,d_2,d_0,d_2,d_2,d_0,d_0,d_2,d_0,d_0]
31e.[d_2,d_2,d_2,d_2,d_2,d_2,d_2,d_0,d_2,d_2,d_0,d_0,d_2,d_0,d_0]	**32e.**[d_2,d_2,d_0,d_2,d_2,d_2,d_2,d_0,d_2,d_2,d_0,d_0,d_2,d_0,d_0]	
31f. [d_2,d_2,d_2,d_2,d_0,d_2,d_2,d_0,d_0,d_2,d_0,d_0,d_2,d_0,d_0]		
31g.[d_1,d_1,d_1,d_1,d_0,d_1,d_1,d_0,d_1,d_1,d_0,d_0,d_1,d_0,d_0]		
31h.[d_1,d_1,d_1,d_1,d_1,d_1,d_1,d_0,d_0,d_1,d_0,d_0,d_1,d_0,d_0]		
31i. [d_2,d_2,d_2,d_2,d_2,d_2,d_2,d_0,d_0,d_2,d_0,d_0,d_2,d_0,d_0]		
31j. [d_2,d_2,d_2,d_2,d_0,d_2,d_2,d_0,d_2,d_2,d_0,d_0,d_2,d_0,d_0]		

Figure 9: Internal Parallel Execution for Multi-Site Offloading

4.3.2.3.1. 1-Core each Resource:

In this scenario, since there is only 1 available processor for all three computing resources so there is no internal parallel execution. However, the external parallel is all pareto-optimal solutions except 31a. The GA explores 10 pareto-optimal solution for the (case3, 1-core) scenario. The solution 31b and 31f produces the lowest possible application response time of *2.4342 Seconds*. However, 31f consume more battery of *1.9303 Joules* as compare to *1.8614 Joules* from 31b but requires less operating cost of *4.86¢* as compare to *7.30¢* in 31b. The 31a does not offload any task to the cloud server and does not require any additional processing cost. All other pareto-optimal solution are intermediate solutions for the user to choose based on their current device configuration.

4.3.2.3.2. 2-Core each Resource:

In this scenario, each computing resource contains two processors and has the ability to perform internal parallel execution all the independent application tasks based on their availability. There are 5 different pareto-optimal solutions in this configuration which are labelled as 32a, 32b, 32c, 32d, and 32e in the *Table 25* above. The solution 32b and 32d both provides the same response time of the application of *2.3715 Seconds* but different in the energy consumption and the processing cost. The solution 32b provides the minimum energy consumption of *2.5554 Joules* with the trade-off of the processing cost of *14.22¢*. However, the solution 32e provides the low operating cost with slightly higher energy consumption. The user can choose the any solution from these pareto-optimal solutions to meet their desired goals based on their current situation of the mobile device.

4.3.2.3.3. 4-Core each Resource:

In this scenario, each computing resource has 4 available processor for the internal parallel execution of the tasks based on their availability. The GA reveals 4 different pareto-optimal solution for the (case3, 4-core) situation which are labelled as 34a, 34b, 34c, 34d in *table 25* above. The solution 34b, provide the lowest response time and energy consumption 2364.4 and 3968.4 respectively among all pareto-optimal solution with the trade-off of high operating cost of 28.37¢. The gain of the response time as compare to 2-core scenario, is 0.3% as compare to energy loss of 55.38%. Thus, this solution provides the lowest response time but consumes more mobile energy and requires high operating cost as compare to all other cases.

4.3.3. Summary:

In this section, we summarize the gain of internal and external execution with respect to response time, energy consumption and monetary cost of the application. The *Table 26* below combines the results for all three cases (No-Offloading, Single-Site Offloading and multi-Site offloading) for the scenarios of 1-core, 2-core and 4-core in each computing resource. Based on the results form *Table 26*, it is safe to conclude that the response time of an application reduces with both external and internal parallel execution as compare to sequential execution. Similarly, the energy consumption is reduced through external parallel execution. However, for the internal parallel execution, there are several different cores to the resource thus the execution division requires more energy consumption to complete the tasks as compare to sequential execution. Finally, offloading monetary cost is consider with respect to response time and energy consumption separately. The user can choose the offloading cost with respect to response time and energy

consumption by paying an additional monetary cost for using cloud resource and network service on the mobile device.

Table 26: EFFECT OF THE NUMBER OF CORES IN EACH DEVICE ON NEAR-OPTIMAL OFFLOADING ALLOCATION

Case	1-core in each device				2-cores in each device				4-cores in each device			
	Near-optimal Soln.	**Resp. Time (Seconds)**	**Energy (Joules)**	**Cost (¢)**	**Near-optimal Soln.**	**Resp. Time (Seconds)**	**Energy (Joules)**	**Cost (¢)**	**Near-optimal Soln.**	**Resp. Time (Seconds)**	**Energy (Joules)**	**Cost (¢)**
No-offloading (Case-1)	**11.**[d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0]	5.5149	4.9634	0¢	**12.**[d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0]	4.8431	6.2148	0¢	**14.**[d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0]	4.8431	9.1207	0¢
Single-site offloading (Case-2)	**21a.**[d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0]	5.5149	4.9634	**0¢**	**22a.**[d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0]	4.8431	6.2148	**0¢**	**24a.**[d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0]	4.8431	9.1207	**0¢**
	21b.[d_1, d_1, d_1, d_1, d_1, d_1, d_1, d_0, d_1, d_1, d_0, d_0, d_1, d_0, d_0]	3.3705	**2.1422**	3.37¢	**22b.**[d_1, d_1, d_0, d_1, d_1, d_1, d_1, d_0, d_1, d_1, d_0, d_0, d_1, d_0, d_0]	**3.1846**	**3.0462**	6.36¢	**24b.**[d_1, d_1, d_0, d_1, d_1, d_1, d_1, d_0, d_1, d_1, d_0, d_0, d_1, d_0, d_0]	**3.1703**	**4.9398**	12.68¢
	21c.[d_1, d_1, d_1, d_1, d_0, d_1, d_1, d_0, d_0, d_1, d_0, d_0, d_1, d_0, d_0]	**3.3130**	2.1939	3.31¢								
	21d.[d_1, d_1, d_1, d_1, d_0, d_1, d_1, d_0, d_1, d_1, d_0, d_0, d_1, d_0, d_0]	3.3316	2.1772	3.33¢								
	21e.[d_1, d_1, d_1, d_1, d_1, d_1, d_1, d_0, d_0, d_1, d_0, d_0, d_1, d_0, d_0]	3.3518	2.1590	3.35¢								
Multi-Site offloading (Case-3)	**31a.**[d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0]	5.5149	4.9634	**0**	**32a.**[d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0]	4.8431	6.2148	**0¢**	**34a.**[d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0, d_0]	4.8431	9.1207	**0¢**
	31b.[d_2, d_2, d_2, d_2, d_1, d_2, d_2, d_0, d_1, d_2, d_0, d_0, d_2, d_0, d_0]	**2.4342**	**1.8614**	7.30¢	**32b.**[d_2, d_2, d_1, d_2, d_2, d_2, d_2, d_0, d_1, d_2, d_0, d_0, d_2, d_0, d_0]	**2.3715**	**2.5540**	14.22¢	**34b.**[d_2, d_2, d_1, d_2, d_1, d_2, d_2, d_0, d_2, d_2, d_0, d_0, d_2, d_0, d_0]	**2.3644**	**3.9684**	28.37¢
	31c.[d_1, d_1, d_1, d_1, d_0, d_1, d_1, d_0, d_0, d_1, d_0, d_0, d_1, d_0, d_0]	3.3130	2.1939	3.31¢	**32c.**[d_1, d_1, d_0, d_1, d_1, d_1, d_1, d_0, d_1, d_1, d_0, d_0, d_1, d_0, d_0]	3.1846	3.0462	6.36¢	**34c.**[d_1, d_1, d_0, d_1, d_1, d_1, d_1, d_0, d_1, d_1, d_0, d_0, d_1, d_0, d_0]	3.1703	4.9398	12.68¢
	31d.[d_1, d_1, d_0, d_1, d_1, d_1, d_1, d_0, d_1, d_1, d_0, d_0, d_1, d_0, d_0]	3.3724	2.1471	3.37¢	**32d.**[d_2, d_2, d_2, d_2, d_2, d_2, d_2, d_0, d_2, d_2, d_0, d_0, d_2, d_0, d_0]	2.3774	2.5576	9.50¢	**34d.**[d_2, d_2, d_0, d_2, d_2, d_2, d_2, d_0, d_2, d_2, d_0, d_0, d_2, d_0, d_0]	2.3655	3.9740	18.92¢
	31e.[d_2, d_2, d_2, d_2, d_2, d_2, d_2, d_0, d_2, d_2, d_0, d_0, d_2, d_0, d_0]	2.4630	1.8700	4.92¢	**32e.**[d_2, d_2, d_0, d_2, d_2, d_2, d_2, d_0, d_2, d_2, d_0, d_0, d_2, d_0, d_0]	**2.3715**	2.5584	9.48¢				
	31f.[d_2, d_2, d_2, d_2, d_0, d_2, d_2, d_0, d_0, d_2, d_0, d_0, d_2, d_0, d_0]	**2.4342**	1.9303	4.86¢								
	31g.[d_1, d_1, d_1, d_1, d_0, d_1, d_1, d_0, d_1, d_1, d_0, d_0, d_1, d_0, d_0]	3.3316	2.1772	3.33¢								
	31h.[d_1, d_1, d_1, d_1, d_1, d_1, d_1, d_0, d_0, d_1, d_0, d_0, d_1, d_0, d_0]	3.3518	2.1590	3.35¢								
	31i.[d_2, d_2, d_2, d_2, d_2, d_2, d_2, d_0, d_0, d_2, d_0, d_0, d_2, d_0, d_0]	2.4537	1.8895	4.90¢								
	31j.[d_2, d_2, d_2, d_2, d_0, d_2, d_2, d_0, d_2, d_2, d_0, d_0, d_2, d_0, d_0]	2.4435	1.9108	4.88¢								

Chapter 5: Conclusion

5.1. Conclusion:

The increasing demand of mobile devices in our daily lives requires the development of intensive applications on that platform. However, the physical structure, limited computation capability and dependence on battery consumption make the development of an intensive applications challenging on the mobile devices. Mobile cloud computing offers code offloading framework as a medium between mobile device and cloud server to mitigate these challenges. These challenges can be further reduced through multi-site computation offloading. This thesis attempts to solve the problem of multi-site computation offloading for mobile applications by introducing a multi-state decision variable. The states of the decision variable are equal to the number of available computation resources at the time of offloading. Our work goes beyond existing approaches by considering parallel execution of tasks during offloading decision in contrast to others who primarily focused on sequential executions. Unlike prior work in computation offloading, our work considers the effect of *Internal and External parallelism* on the offloading allocation. The assignment of tasks on multiple different resources for parallel execution refer to *External Parallelism*. Similarly, the assignment of tasks on the different cores of a single resource refer to *Internal Parallelism*. Further, we proposed a multi-objective code offloading algorithm to meet a user's need for application computation. Our multi-objective algorithm computes the response time, energy consumption and monetary cost by considering the effect of external and internal parallelism on each offloading allocation. We used Genetic Algorithm to optimize the offloading allocation and to find near-optimal solution(s) with respect to response time, energy consumption and monetary cost. The Genetic Algorithm invokes our proposed algorithm to evaluate the fitness of each offloading solution and produce pareto-optimal offloading allocations for each objective.

The user can choose any pareto-optimal solution based on their objective and offloading needs. The gain of our multi-objective algorithm between external and sequential as well as internal and sequential is verified through a real-world face recognition application from [27]. The results show that accounting for the effect of parallel execution yields a better near-optimal solution for the allocation problem as compared to excluding parallelism in the analysis.

5.2. Future Work

Our proposed code offloading framework performs the parallel implementation of the parallel path and the parallel execution itself is an open problem. The framework can be further enhanced by addressing some of these parallel execution limitations. The future research work to address these limitations are as follow:

- In the internal parallel execution, the data is accessed from the memory and cache by all processors of the single device. In the current framework, the data access time from the memory and cache is not included in the response time of the application.

- In regards to VM (virtual machine), there is an initialization time of each VM which is currently not considered in the calculation. It is assumed that each VM is already initialized and it is ready to receive the offloading tasks of an application from the mobile device.

- In the external parallel execution, the communication among different devices is considered to be constant. However, the data exchange rate between the mobile device and the cloud server continuously changes based on the user wireless network plan and the geographic location. The dramatic change in the wireless connection needs to be addressed in the current framework.

- In the current framework, it is assumed that the mobile device is always connected to the internet and there is no sudden interrupt in the wireless connection. Further research is required to handle unexpected disconnection of the mobile device or cloud servers from the network.

- The current state of this framework is heavily dependent on the user input for the model specification, mobile device and cloud server configurations. The user manually has to enter all the details before performing the simulation. A user interface can be created to gather the model specifications from the mobile device profiler and cloud server's APIs.

References

[1] An energy-efficient cloud-based offloading decision algorithm for mobile devices. (2012). *SCIENTIA SINICA Informationis*. doi:10.1360/112011-922

[2] Atayero, A. A., & Feyisetan, O. (n.d.). Security Issues in Cloud Computing: The Potentials of Homomorphic Encryption. *Journal of Emerging Trends in Computing and Information Sciences, 2 (10). Pp. 546-552. ISSN 2079-8407*.

[3] Berg, F., Durr, F., & Rothermel, K. (2015). Increasing the efficiency of code offloading through remote-side caching. *2015 IEEE 11th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*. doi:10.1109/wimob.2015.7348013

[4] Chen, L., Ho, Y., Kuo, W., & Tsai, M. (2015). Intelligent file transfer for smart handheld devices based on mobile cloud computing. *International Journal of Communication Systems*,*30*(1). doi:10.1002/dac.2947

[5] Cheng, Z., Li, P., Wang, J., & Guo, S. (2015). Just-in-Time Code Offloading for Wearable Computing. IEEE Transactions on Emerging Topics in Computing, 3(1), 74-83. doi:10.1109/tetc.2014.2387688

[6] Chun, B., Ihm, S., Maniatis, P., Naik, M., & Patti, A. (2011). Clonecloud: Elastic Execution between Mobile Device and Cloud. *Proceedings of the Sixth Conference on Computer Systems - EuroSys 11*. doi:10.1145/1966445.1966473

[7] Cuervo, E., Balasubramanian, A., Cho, D., Wolman, A., Saroiu, S., Chandra, R., & Bahl, P. (2010). Maui: Making Smartphones Last Longer with Code Offload. *Proc. ACM MobiSys 2010,San Francisco, CA*. doi:10.1145/1814433.1814441

[8] Deb, K., & Jain, H. (2014). An Evolutionary Many-Objective Optimization Algorithm Using Reference-Point-Based Nondominated Sorting Approach, Part I: Solving Problems With Box Constraints. *IEEE Transactions on Evolutionary Computation,18*(4), 577-601. doi:10.1109/tevc.2013.2281535

[9] Deng, S., Huang, L., Taheri, J., & Zomaya, A. Y. (2015). Computation Offloading for Service Workflow in Mobile Cloud Computing. *IEEE Transactions on Parallel and Distributed Systems,26*(12), 3317-3329. doi:10.1109/tpds.2014.2381640

[10] Eason, G., Noble, B., & Sneddon, I. N. (1955). On Certain Integrals of Lipschitz-Hankel Type Involving Products of Bessel Functions. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences,247*(935), 529-551. doi:10.1098/rsta.1955.0005

[11] Ellouze, A., Gagnaire, M., & Haddad, A. (2015). A Mobile Application Offloading Algorithm for Mobile Cloud Computing. 2015 3rd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering. doi:10.1109/mobilecloud.2015.11

[12] Flores, H., Hui, P., Tarkoma, S., Li, Y., Srirama, S., & Buyya, R. (2015). Mobile code offloading: From concept to practice and beyond. *IEEE Communications Magazine,53*(3), 80-88. doi:10.1109/mcom.2015.7060486

[13] Gordon, M. S., Jamshidi, D. A., Mahlke, S., Mao, M. Z., & Chen, X. (n.d.). COMET: Code Offload by Migrating Execution Transparently. *10th USENIX Symposium on Operating Systems Design and Implementation (OSDI '12),*93-106.

[14] Goudarzi, M., Zamani, M., & Toroghi Haghighat, A. (2016). A genetic-based decision algorithm for multisite computation offloading in mobile cloud computing. *International Journal of Communication Systems,30*(10). doi:10.1002/dac.3241

[15**]** Hadka, D. (n.d.). Beginner's Guide to the MOEA Framework. MOEA Framework User Guide.

[16] Hudik, Martin, and Michal Hodon. "Performance Optimization of Parallel Algorithms." Journal of Communications and Networks, vol. 16, no. 4, 2014, pp. 436–446., doi:10.1109/jcn.2014.000074.

[17] Kosta, S., Aucinas, A., Hui, P., Mortier, R., & Zhang, X. (2012). ThinkAir: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading. *2012 Proceedings IEEE INFOCOM*. doi:10.1109/infcom.2012.6195845

[18] Kumar, K., Liu, J., Lu, Y., & Bhargava, B. (2012). A Survey of Computation Offloading for Mobile Systems. Mobile Networks and Applications, 18(1), 129-140. doi:10.1007/s11036-012-0368-0

[19] L. Jiao, R. Friedman, X. Fu, S. Secci, Z. Smoreda and H. Tschofenig, "Cloud-based computation offloading for mobile devices: State of the art, challenges and opportunities," 2013 Future Network & Mobile Summit, Lisboa, 2013, pp. 1-11.

[20] Maxwell, J. C. (n.d.). Electricity And Magnetism. *A Treatise on Electricity and Magnetism,*Xxxi-Xxxiv. doi:10.1017/cbo9780511709333.002

[21] Niu, R., Song, W., & Liu, Y. (2013). An Energy-Efficient Multisite Offloading Algorithm for Mobile Devices. International Journal of Distributed Sensor Networks, 9(3), 518518. doi:10.1155/2013/518518

[22] Park, J., Kim, H., Jeong, Y., & Lee, E. (2013). Two-phase grouping-based resource management for big data processing in mobile cloud computing. *International Journal of Communication Systems,27*(6), 839-851. doi:10.1002/dac.2627

[23] Shiraz, M., Gani, A., Ahmad, R. W., Shah, S. A., Karim, A., & Rahman, Z. A. (2014). A Lightweight Distributed Framework for Computational Offloading in Mobile Cloud Computing. *PLoS ONE*,*9*(8). doi:10.1371/journal.pone.0102270

[24] Sinha, K., & Kulkarni, M. (2011). Techniques for Fine-Grained, Multi-site Computation Offloading. *2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. doi:10.1109/ccgrid.2011.69

[25] Sivanandam, S. N., & Deepa, S. N. (2010). *Introduction to genetic algorithms*. Berlin: Springer.

[26] Terefe, M. B., Lee, H., Heo, N., Fox, G. C., & Oh, S. (2016). Energy-efficient multisite offloading policy using Markov decision process for mobile cloud computing. *Pervasive and Mobile Computing*,*27*, 75-89. doi:10.1016/j.pmcj.2015.10.008

[27] Wu, Huaming, et al. "An Optimal Offloading Partitioning Algorithm in Mobile Cloud Computing." Quantitative Evaluation of Systems Lecture Notes in Computer Science, 2016, pp. 311–328., doi:10.1007/978-3-319-43425-4_21.

[28] Yang, L., Cao, J., Yuan, Y., Li, T., Han, A., & Chan, A. (2013). A framework for partitioning and execution of data stream applications in mobile cloud computing. ACM SIGMETRICS Performance Evaluation Review, 40(4), 23. doi:10.1145/2479942.2479946

[29] Sheikh, I. & Das, O. (2018). Effect of Parallel Execution on Multi-site Computation Offloading in Mobile Cloud Computing. *Submitted to the 26th IEEE International Symposium on the Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS 2018)*.