

A MODEL FOR POWER EFFICIENCY OF MOBILE
DEVICES THROUGH LIGHTWEIGHT METHOD
LEVEL COMPUTATIONAL OFFLOADING



MUSHTAQ ALI

DOCTOR OF PHILOSOPHY

UNIVERSITI MALAYSIA PAHANG

UNIVERSITI MALAYSIA PAHANG

DECLARATION OF THESIS AND COPYRIGHT

Author's Full Name : MUSHTAQ ALI

Date of Birth : 07-04-1977

Title : A Model for Power Efficiency of Mobile Devices through
Lightweight Method Level Computational Offloading

Academic Session : 2017 / 2018

I declare that this thesis is classified as:

- CONFIDENTIAL (Contains confidential information under the Official Secret Act 1997)*
- RESTRICTED (Contains restricted information as specified by the organization where research was done)*
- OPEN ACCESS I agree that my thesis to be published as online open access (Full Text)

I acknowledge that Universiti Malaysia Pahang reserve the right as follows:

1. The Thesis is the Property of Universiti Malaysia Pahang
2. The Library of Universiti Malaysia Pahang has the right to make copies for the purpose of research only.
3. The Library has the right to make copies of the thesis for academic exchange.

Certified by:

(Student's Signature)

KS1796323

New IC/Passport Number
Date: 14 Feb 2018

(Supervisor's Signature)

Dr. Mohamad Fadli Zolkipli
Name of Supervisor
Date: 21 Feb 2018

NOTE : * If the thesis is CONFIDENTIAL or RESTRICTED, please attach with the letter page 2 from the organization with the period and reasons for confidentiality or restriction.

SUPERVISOR'S DECLARATION

We hereby declare that we have checked this thesis and in our opinion, this thesis/project* is adequate in terms of scope and quality for the award of the degree of *Doctor of Philosophy in Computer Sciences.

(Supervisor's Signature)

Full Name : Dr. Mohamad Fadli Zolkipli
Position : Senior Lecturer
Date : 21 Feb 2018



(Co-supervisor's Signature)

Full Name : Prof. Dr. Jasni Mohamad Zain
Position : Professor
Date : 19 Feb 2018

STUDENT'S DECLARATION

I hereby declare that the work in this thesis is based on my original work except for quotations and citation which have been duly acknowledged. I also declare that it has not been previously or concurrently submitted for any other degree at Universiti Malaysia Pahang or any other institutions.

(Student's Signature)

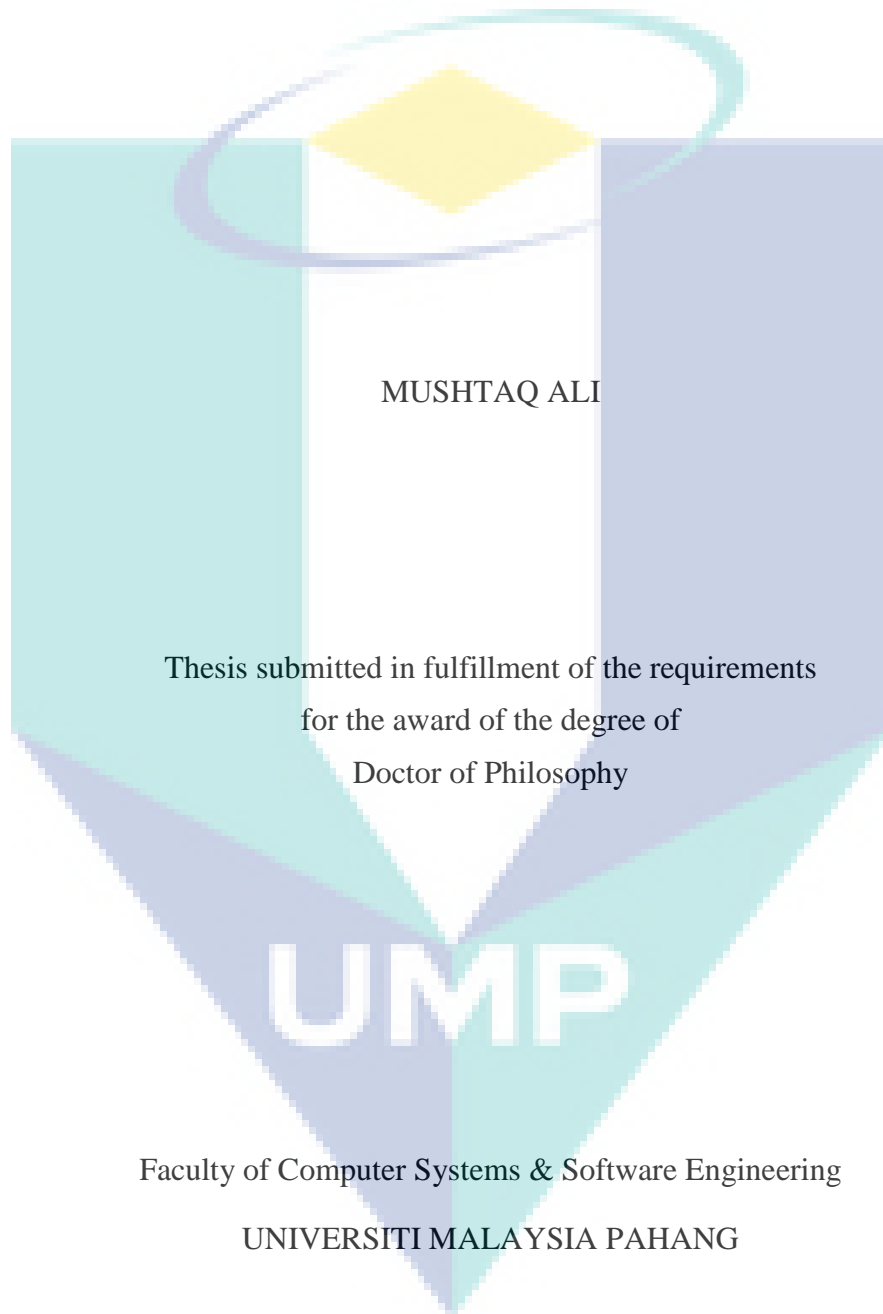
Full Name : MUSHTAQ ALI

ID Number : PCC13009

Date : 14 Feb 2018

UMP

A MODEL FO POWER EFFICIENCY OF MOBILE DEVICES THROUGH
LIGHTWEIGHT METHOD LEVEL COMPUTATIONAL OFFLOADING



FEBRUARY 2018

Dedicated to

“My Mom and Dad (Late)”

I learned from my Mother: “Where there is a will there is a way”

&

I learned from my Father: “Always do the best job, your reputation
worth more than a quick profit”



UMP

ACKNOWLEDGEMENT

All virtues and praise to Almighty Allah, Whose blessings are unlimited. I am thankful to my current research supervisor Dr. Mohamad Fadli Zolkipli who guided me at each step in the beginning till end to pursue my research. I pay my tribute to my previous research supervisor Prof. Dr. Jasni Mohamad Zain for her kind support to deal with all the challenges pertaining to this research. Her wise guidance led my way from my first step of this research till the last one; particularly in the process of data acquisition and lab instruments. Apart from my research supervisors, I am very thankful to Associate Professor Dr. Mazlina Abdul Majid, Assistant Professor Dr. Nawsher Khan and my dear friend Mr. Waris Khan who helped me in starting my PhD journey and supported me throughout. I also received the cooperation of Dr. Mansoor Abdullateef Abdulgaber, Assistant Professor at University Malaysia Pahang and Dr. Muhammad Shiraz Assistant Professor in Federal Urdu University Islamabad Pakistan. I would like to extend my gratitude to my uncle Noor Rahman Afghani without whom guidance I would not be here today. I am very thankful to my brother Ashraf Ali who supported me and taught me how to tackle the hard time. My thanks further goes to university friends Dr. Gran Badshah, Shahid Anwar and Riaz ul Haq, who helped me in overcoming the hurdles during my studies. The prime credit goes to the University Malaysia Pahang that funded this work under research grant GRS 110334. I also wish to express my sincere gratitude and appreciation to my family specially my wife and two beautiful kids Manahil Ali and Abdul Moiz, who endured my absence with patience and never let me have any idea of those problems that they faced persistently during the whole period I remained abroad.

The logo of Universiti Malaysia Pahang (UMP) is a large, stylized letter 'V' shape. The left side of the 'V' is light blue, the right side is light green, and the bottom point is a darker blue. The letters 'UMP' are written in white, bold, sans-serif font across the center of the 'V'.

ABSTRAK

Peranti mudah alih telah menjadi satu bahagian penting dalam kehidupan seharian kita. Walaubagaimanapun, masa yang terhad pada bateri mengurangkan masa operasinya. Untuk menangani masa yang terhad pada bateri, pemunggahan pengiraan (computational offloading) digunakan untuk melepaskan tugas intensif daripada peranti mudah alih ke pelayan jauh bagi melaksanakan tugas itu dari jauh dan menjimatkan hayat bateri. Rangka kerja pemunggahan pengiraan berdasarkan penghijrahan Virtual Machine (VM), aplikasi keseluruhan penghijrahan atau tahap kaedah (method level) tradisional pemunggahan adalah sumber yang intensif dan memakan masa. Pembahagian yang dinamik pada aplikasi, pelaksanaan tugas pada pelayan awan, panggilan perkhidmatan oleh SOAP dan tiada penentu mekanisme untuk parameter yang telah ditetapkan, mengubah rangka kerja tahap kaedah pengiraan (method level computational) menjadi tidak cekap untuk penjimatan tenaga. Dalam usaha untuk menangani kekurangan rangka kerja tahap kaedah pengiraan pemunggahan (method level computational offloading), rangka kerja tahap kaedah yang ringan (lightweight method) telah dicadangkan. Empat komponen yang berbeza digunakan dalam rangka kerja yang dicadangkan bagi menghapuskan kelemahan rangka kerja yang dibangunkan sebelum ini. REST digunakan untuk panggilan perkhidmatan yang berasaskan JSON dan ia menghapuskan SOAP yang berasaskan XML. Oleh sebab itu, REST adalah satu pendekatan ringan. REST juga mengurangkan saiz data komunikasi sehingga 100% berbanding dengan panggilan perkhidmatan SAOP. Pelayan tumpang (Surrogate server) dikonfigurasi pada jarak hop tunggal yang mengurangkan RTT dan seterusnya mengurangkan penggunaan kuasa. Aplikasi itu dibahagikan di peringkat tahap kaedah (method level) yang sama dengan rangka kerja tahap kaedah sebelumnya tetapi pembahagian berlaku di peringkat kod sumber statik. Satu mekanisme khusus untuk pemilihan parameter yang telah ditetapkan adalah penting untuk dipertimbangkan sebelum setiap offload. Parameter yang telah ditetapkan terdiri daripada tahap bateri, jenis rangkaian dan masa pelaksanaan telah mengesahkan penjimatan tenaga semasa pemunggahan. Rangka kerja yang dicadangkan telah dilaksanakan dalam persekitaran pengkomputeran awan mudah alih yang sebenar. Masa Pelaksanaan dan Penggunaan Tenaga oleh Pelaksanaan Tempatan dan Pemunggahan Secara Tradisional ditanda aras untuk menyiasat dan mengesahkan pelaksanaan rangka kerja tahap kaedah ringan (lightweight method level) yang dicadangkan. Prototaip dibangunkan dengan tiga komponen REST-Offload, Pelaksanaan Tempatan dan Traditional-Offload serta ia diuji dalam persekitaran awan mudah alih sebenar untuk Masa Pelaksanaan dan Penggunaan Tenaga. Hasilnya telah menunjukkan bahawa penyelesaian yang dicadangkan telah mengurangkan penggunaan sumber pada peranti mudah alih. REST-Offload adalah amat berguna jika dibandingkan dengan kedua-dua Pelaksanaan Tempatan dan kaedah Pemunggahan Tradisional. Ia mengurangkan kira-kira 50% Masa Pelaksanaan dan kira-kira 38% Penggunaan Tenaga.

ABSTRACT

Mobile devices have become an integral part of our daily lives. However, the restricted battery timing curtails longer operational hours. To tackle the limited battery timing issue, a technique, computational offloading is used. In computational offloading, the intensive tasks are offloaded from mobile devices to remote server in order to execute the task remotely and save battery life. Computational offloading frameworks/models based on VM migration, whole application migration, or traditional method level offloading are resources intensive and time consuming. The dynamic partitioning of application, execution of task at cloud server, service call by Simple Object Access Protocol (SOAP) and no defined mechanism for predefined parameters, make the previous method level computational frameworks/models inefficient for energy saving. In order to address the inefficiencies of previous method level computational offloading frameworks/models, a lightweight method level computational offloading model is proposed. Four distinct components are deployed in the proposed model which eliminates the shortcomings of previously developed frameworks/models. A Representational State Transfer (REST) based technique developed for calling the remote services which is based on JSON instead of XML, and hence is lightweight. REST also reduces the size of communication data at approximately 100% as compared to SAOP service call. Surrogate server is configured at a single hop distance which reduces the RTT and ultimately reduces the power consumption. The application is partitioned at method level by a novel dynamic technique in source code, which counters the inefficiencies of existing partitioning techniques. A mechanism for selection of predefined parameters is defined. These parameters are important to consider before each offload. The predefined parameters consist of battery level, network type, and execution time which affirms the energy saving during offloading. The proposed framework is implemented in the real mobile cloud computing environment. Execution time and energy consumption of both local execution and traditional offloading are benchmarked in order to investigate and validate the performance of the proposed lightweight method level model. The prototype is developed with three components which are REST-Offload, Local Execution and Traditional-Offload and then tested in real mobile cloud environment for Execution Time and Energy Consumption. The result of this research indicates that the proposed solution diminishes resources utilization. The REST-Offload is significantly useful compared to both Local Execution and Traditional Offloading methods. It reduces about 50% Execution Time and approximately 38% Energy Consumption.

TABLE OF CONTENTS

	Page
DECLARATION	
TITLE PAGE	
ACKNOWLEDGEMENT	iii
ABSTRAK	iv
ABSTRACT	v
TABLE OF CONTENTS	vi
LIST OF TABLES	x
LIST OF FIGURES	xii
LIST OF ABBREVIATION	xv
CHAPTER 1 INTRODUCTION	
1.1 Overview	1
1.2 Motivation	1
1.3 Problem Background	3
1.4 Problem Statement	6
1.5 Research Questions	7
1.6 Goal and Objectives	8
1.7 Scope of Research	8
1.8 Thesis Organization	9
CHAPTER 2 LITERATURE REVIEW	
2.1 Overview	11
2.2 Background	11
2.2.1 Mobile Computing	12
2.2.2 Cloud Computing	13
2.2.3 Mobile Cloud Computing	16
2.3 Approaches for Augmenting Mobile's Resources	17

2.3.1	Generation of New Hardware Resources	18
2.3.2	Execute Program Slowly	18
2.3.3	New Applications for Resources Constrained Devices	19
2.3.4	Hardware and Software Management Techniques	19
2.4	Taxonomy of Batteries Augmentation Techniques	21
2.4.1	Hardware Level Augmentation	22
2.4.2	Software Level Augmentation	23
2.5	An Analysis of CPU Clock Time, Execution Time and Power Consumption	29
2.6	Computational Offloading	31
2.6.1	Energy Saving Computational Offloading	32
2.6.2	Metrics of Computational Offloading Approaches	34
2.6.3	Taxonomy of Computational Offloading / Cyber foraging Approaches	37
2.7	Related Works	42
2.7.1	Previous Research on Enhancing Mobile's Efficiency	43
2.8	Review of Computational Offloading Frameworks	57
2.8.1	Whole Application Migration Frameworks	57
2.8.2	Virtual Machine (VMs) Migration Frameworks	58
2.8.3	Method Level Migration Frameworks	58
2.9	Analytical Analysis of Method Level Computational Offloading Frameworks	65
2.10	Summary	70
CHAPTER 3 METHODOLOGY		
3.1	Overview	72
3.2	Research Approach	72
3.2	Research Phases	76
3.2.1	Planning Phase	78
3.2.2	Analysis, Design and Implementation Phase	79
3.2.3	Evaluation Phase	87
3.2.4	Comparative Analysis	88
3.3	Summary	89

CHAPTER 4 DESIGN AND IMPLEMENTATION

4.1	Overview	90
4.2	Computational Offloading and Execution Time	90
4.3	The Model	92
4.3.1	Mobile Component	97
4.3.2	Server Component	102
4.3.3	Communication	103
4.4	Operational Logic	103
4.4.1	Operational Logic of REST-Offload Model	103
4.4.2	Application Execution Flow of REST-Offload Model	105
4.4.3	Proposed Algorithm for the Selection of Pre-defined Parameters	109
4.5	Evaluation of the Model	112
4.5.1	Experiment Setup	113
4.5.2	Prototype	117
4.5.3	Data Collection and Data Processing	118
4.5.4	Data Collected by Executing Application at Local Mobile Device	119
4.5.5	Data Collected by Offloading Application using Traditional Offloading Techniques	122
4.5.6	Data Collected by Offloading Application using REST-Offload	124
4.6	Summary	126

CHAPTER 5 RESULTS AND DISCUSSION

5.1	Overview	127
5.2	Analysis of Application Execution at Local Mobile Device	127
5.3	Analysis of Application Executed through Traditional Offloading Methods	131
5.4	Analysis of Application Execution using REST-Offload Method	136
5.5	Comparison of ET/TT of Matrix Multiplication B/W Local Execution, Traditional Offloading and REST-Offload	140
5.5.1	Execution Time (ET) Result Comparison of Samsung Galaxy A5	140
5.5.2	Execution Time (ET) Result Comparison of ASUS Zenfone5	142
5.6	Comparison of Energy Consumption Cost of Matrix Multiplication between Local Execution, Traditional Offloading and REST-Offload	145

5.6.1	Energy Consumption (EC) Result Comparison of Samsung Galaxy A5	145
5.6.2	Energy Consumption (EC) Result Comparison of ASUS Zenfone5	148
5.7	Comparison of Execution Time (ET) and Energy Consumption (EC) between REST-Offload and DCOF	151
5.8	ET and EC Comparison of Samsung Galaxy A5 with ASUS Zenfone5 for all Three Scenarios	153
5.9	Efficiency Comparison of REST-Offload against Existing Frameworks	160
5.9.1	Efficiency Comparison of Execution Time	161
5.9.2	Efficiency Comparison of Energy Consumption	162
5.10	Specification Comparisons of REST-Offload and Existing Approaches	163
5.11	Threat to Validity	166
5.11.1	Usage Scenario	166
5.11.2	<i>DuTs</i> Threats	167
5.11.3	Single Hop Surrogate Threats	170
5.11.4	Energy and Time Estimating Tools Threats	170
5.9	Summary	171
CHAPTER 6 CONCLUSION AND FUTURE WORK		
6.1	Overview	173
6.2	Discussion	173
6.2.1	Revisit of the Research Objectives	174
6.2.2	Contribution of the Research	177
6.2.3	The Scalability of the Research	178
6.5	Limitations and Future Work	178
REFERENCES		180

LIST OF TABLES

Table 2.1	A Comparison of advancement between Static Servers and Mobile Devices with a gap of 5 years	16
Table 2.2	Analysis of the Previous Research Work on Resources Augmentation of Mobile Devices	54
Table 2.3	Comparative Review of Computational Offloading Models	60
Table 2.4	Summary of Method Level Computational Offloading Frameworks	66
Table 3.1	Energy Consumption Cost EC1 of Offloading Matrix Multiplication Service in Traditional Computational Offloading	82
Table 3.2	Time Consumption Cost TC1 of Offloading Matrix Multiplication Service in Traditional Computational Offloading	85
Table 3.3	The Experiment Setup of Previous Research Works	87
Table 4.1	The Hop-Count and Average Delay	93
Table 4.2	Local Execution Time of Prototype Application at Samsung Galaxy A5	120
Table 4.3	Local Execution Time of Prototype Application at ASUS Zenfone5	120
Table 4.4	Energy Consumption (EC) of Prototype Application at Samsung Galaxy A5 through Local Execution	121
Table 4.5	Energy Consumption of Prototype Application at ASUS Zenfone5 through Local Execution	121
Table 4.6	Execution Time of Prototype Application Execution through Traditional Offloading using Samsung Galaxy A5	122
Table 4.7	Execution Time of Prototype Application Execution through Traditional Offloading using ASUS Zenfone5	122
Table 4.8	Energy Consumption Cost of Prototype Application Execution through Traditional Offloading using Samsung Galaxy A5	123
Table 4.9	Energy Consumption Cost of Prototype Application Execution through Traditional Offloading using ASUS Zenfone5	123
Table 4.10	Execution Time of Prototype Application Execution through REST-Offload Method using Samsung Galaxy A5	125
Table 4.11	Execution Time of Prototype Application Execution through REST-Offload Method using ASUS Zenfone5	125
Table 4.12	Energy Consumption Cost of Prototype Application Execution through REST-Offload Method using Samsung Galaxy A5	125

Table 4.13	Energy Consumption Cost of Prototype Application Execution through REST-Offload Method using ASUS Zenfone5	125
Table 5.1	Comparison of ET / TT of Samsung Galaxy A5 between Local Execution, Traditional Offloading and REST-Offload	140
Table 5.2	P% of ET / TT of REST-Offload Method for Samsung Galaxy A5 using the equation, $Y = P\% * X$, against Local Execution and Traditional Offloading	142
Table 5.3	Comparison of ET of ASUS Zenfone5 between Local Execution, Traditional Offloading and REST-Offload	143
Table 5.4	P% of ET / TT of REST-Offload Method for ASUS Zenfone5 using the equation, $Y = P\% * X$, against Local Execution and Traditional Offloading	144
Table 5.5	Comparison of Energy Consumption Cost between Local Execution, Traditional Offloading and REST-Offload for Galaxy A5	145
Table 5.6	P% of Energy Consumption of REST-Offload Method for Samsung Galaxy A5 using the equation, $Y = P\% * X$, against Local Execution and Traditional Offloading	147
Table 5.7	Comparison of Energy Consumption Cost between Local Execution, Traditional Offloading and REST-Offload for ASUS Zenfone5	148
Table 5.8	P% of Energy Consumption of REST-Offload Method of ASUS Zenfone5 using the equation, $Y = P\% * X$, against Local Execution and Traditional Offloading	150
Table 5.9	Comparison of ET between REST-Offload and DCOF	151
Table 5.10	Comparison of EC between REST-Offload and DCOF	152
Table 5.11	ET Comparison of Samsung Galaxy A5 and ASUS Zenfone5.	154
Table 5.12	Specifications of Samsung Galaxy A5 and ASUS Zenfone5	155
Table 5.13	EC Comparison of Samsung Galaxy A5 and ASUS Zenfone5.	158
Table 5.14	Efficiency Comparison of Execution Time	161
Table 5.15	Efficiency Comparison of Energy Consumption	162
Table 5.16	Comparative Analysis of REST-Offload against Others	164

LIST OF FIGURES

Figure 2.1	Framework of Mobile Computing	13
Figure 2.2	Service-Oriented Layered Architecture of Cloud Computing	15
Figure 2.3	A Typical Framework of Mobile Cloud Computing	17
Figure 2.4	Mobile's Resources Enhancing Approaches	21
Figure 2.5	Taxonomy of Mobile's Battery Augmentation Techniques	22
Figure 2.6	Clone-Cloud Framework	29
Figure 2.7	Relationship between B, D and C	33
Figure 2.8	Metrics of Cyber Foraging	34
Figure 2.9	Execution Flow of Computational Offloading	35
Figure 2.10	Taxonomy of Cyber Foraging/Computational Offloading	37
Figure 2.11	CloneCloud- Architectural and System Framework	44
Figure 2.12	Cloudlet Architectural Model	46
Figure 2.13	Context-Aware Power Manager	49
Figure 2.14	CALEEF Architectural Model	50
Figure 3.1	Research Flowchart	73
Figure 3.2	Operational Model	77
Figure 3.3	EC Cost of Matrix Multiplication in Traditional Offloading	82
Figure 3.4	ET Cost of Matrix Multiplication in Traditional Offloading	85
Figure 4.1	Computational Offloading and Execution Time/Turnaround Time	92
Figure 4.2	Mobile Devices Connected to Remote Servers through Single hop, limited multi-hop and multi-hop	94
Figure 4.3	Model of REST-Offload	96
Figure 4.4	Operational Logic of REST-Offload Model	104
Figure 4.5	Application Execution Flow of the Proposed Model	107
Figure 4.6	Flow of the Selection of Predefine Parameters	110
Figure 4.7	The Environment of Experimental Offloading Scenario	114
Figure 4.8	A Screen Short of Monsoon Power Monitor Application Tools	116
Figure 4.9	Use of Monsoon Power Monitor in Experiments for Power Consumption Readings	117
Figure 5.1	Execution Time (ms) of Matrix Multiplication in Local Mobile Device Samsung Galaxy A5	128

Figure 5.2	Execution Time (ms) of Matrix Multiplication in Local Mobile Device ASUS Zenfone5	129
Figure 5.3	Local Energy Consumption (J) of Matrix Multiplication by Samsung Galaxy A5	130
Figure 5.4	Local Energy Consumption (J) of Matrix Multiplication using ASUS Zenfone5	131
Figure 5.5	Execution Time of Matrix Multiplication in Traditional Offloading by Samsung Galaxy A5	133
Figure 5.6	Execution Time of Matrix Multiplication in Traditional Offloading by ASUS Zenfone5	133
Figure 5.7	Energy Consumption (J) of Matrix Multiplication in Traditional Offloading by Samsung Galaxy A5	134
Figure 5.8	Energy Consumption (J) of Matrix Multiplication in Traditional Offloading by ASUS Zenfone5	135
Figure 5.9	Execution Time (ms) of Matrix Multiplication in REST-Offload using Samsung Galaxy A5	136
Figure 5.10	Execution Time (ms) of Matrix Multiplication in REST-Offload using ASUS Zenfone5	138
Figure 5.11	Energy Consumption (J) of Matrix Multiplication in REST-Offload using Samsung Galaxy A5	138
Figure 5.12	Energy Consumption (J) of Matrix Multiplication in REST-Offload using ASUS Zenfone5	139
Figure 5.13	Execution Time (ms) Comparison of Matrix Multiplication of all Three Scenarios using Samsung Galaxy A5	141
Figure 5.14	Execution Time (ms) Comparison of Matrix Multiplication of all Three Scenarios using ASUS Zenfone5	143
Figure 5.15	Energy Consumption (J) Comparison of Matrix Multiplication of all Three Scenarios using Samsung Galaxy A5	147
Figure 5.16	Energy Consumption (J) Comparison of Matrix Multiplication of all Three Scenarios using ASUS Zenfone5	149
Figure 5.17	ET (ms) Comparison of Matrix Multiplication of between DCOF and REST-Offload	152

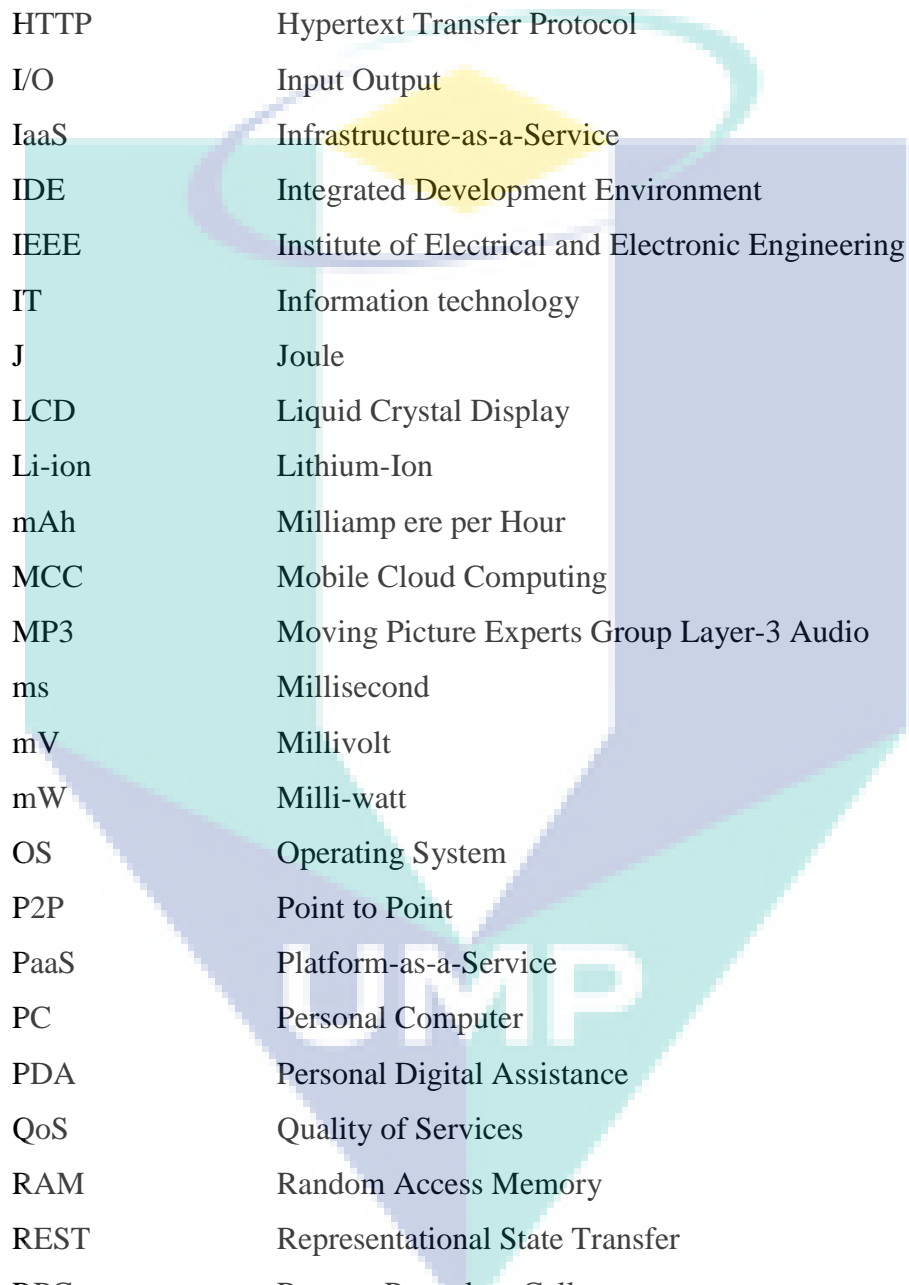
Figure 5.18	EC (J) Comparison of Matrix Multiplication of between DCOF and REST-Offload	153
Figure 5.19	Execution Time (ms) Comparison of Galaxy A5 and ASUS Z5 in Local Execution	156
Figure 5.20	Execution Time (ms) Comparison of Galaxy A5 and ASUS in Traditional Offloading	156
Figure 5.21	Execution Time (ms) Comparison of Galaxy A5 and ASUS in REST-Offload	157
Figure 5.22	Energy Consumption (J) Comparison of Galaxy A5 and ASUS Z5 in Local Execution	159
Figure 5.23	Energy Consumption (J) Comparison of Galaxy A5 and ASUS Z5 in Traditional Offloading	159
Figure 5.24	Energy Consumption (J) Comparison of Galaxy A5 and ASUS Z5 in REST-Offload	160
Figure 5.25	Efficiency Comparison of Execution Time	161
Figure 5.26	Efficiency Comparison of Energy Consumption	163



UMP

LIST OF ABBREVIATIONS

3D	Three Dimensions
3G	Third Generations
4G	Fourth Generations
ADB	Android Debug Bridge
AMOLED	Active Matrix Organic Light Emitting Diode
AP	Access Point
API	Application Programming Interface
AWS	Amazon Web Services
BTS	Base Transceiver Station
BW	Bandwidth
CALEEF	Context Aware Light Weight Energy Efficient Framework
CasCap	Cloud Assisted Context-Aware Power Management
CC	Cloud Computing
CDB	Context Detection Block
CEO	Chief Executive Officer
CF	Cyber Foraging
CI	Confidence Interval
CPU	Central Processing Unit
DiET	Distributed Execution Transformer
DOCs	Documents
DPM	Dynamic Power Management
DuT	Device under Test
EC2	Elastic Cloud Computing
ECC	Energy Consumption Cost
ECC	Elastic Cloud Computing
EDGE	Enhanced Data for GSM Evolution
ET	Execution Time
FA	Fidelity Adaptation
FTP	File Transfer Protocol
GB	Giga Byte
GHz	Giga Hurts



GPRS	Global Packet Radio Service
GPS	Global Positioning System
GPU	Graphical Processing Unit
GSM	Global System for Mobiles
HTC	High Tech Computer Corporation
HTTP	Hypertext Transfer Protocol
I/O	Input Output
IaaS	Infrastructure-as-a-Service
IDE	Integrated Development Environment
IEEE	Institute of Electrical and Electronic Engineering
IT	Information technology
J	Joule
LCD	Liquid Crystal Display
Li-ion	Lithium-Ion
mAh	Milliamp ere per Hour
MCC	Mobile Cloud Computing
MP3	Moving Picture Experts Group Layer-3 Audio
ms	Millisecond
mV	Millivolt
mW	Milli-watt
OS	Operating System
P2P	Point to Point
PaaS	Platform-as-a-Service
PC	Personal Computer
PDA	Personal Digital Assistance
QoS	Quality of Services
RAM	Random Access Memory
REST	Representational State Transfer
RPC	Remote Procedure Calls
RSD	Relative Standard Deviation
RTT	Round Trip Time
S3	Simple Storage Services
SaaS	Software-as-a-Service

SD	Standard Deviation
SDK	Software Development Kit
SID	Smart Internet Device
SOAP	Simple Object Access Protocol
T_0	Zero Throughputs
TC	Time Cost
TCP	Transfer Control Protocol
T_M	Maximum Throughputs
T_N	Normal Throughputs
TT	Turnaround Time
UDP	User Datagram Protocol
UI	User Interface
UMTS	Universal Mobile Telecommunication System
USB	Universal Serial Bus
VM	Virtual Machine
WAN	Wide Area Network
Wi-Fi	Wireless Fidelity
Wi-Max	Worldwide Interoperability for Microwave Access
WLAN	Wireless Local Area Network
WWW	World Wide Web

UMP

CHAPTER 1

INTRODUCTION

1.1 Overview

This chapter presents the theoretical framework and motivations for the proposed research. It discusses the problem statement, states the objectives and describes the methodology used for the proposed research. The chapter is divided into six sections. Section 1.2 highlights the motivations for the proposed research by explaining the importance of the proposed work and significance of the proposed solution. Section 1.3 discusses the problem background. Section 1.4 states the problem statement. Section 1.5 discusses the research question. Section 1.6 highlights the research objectives. Section 1.7 summarizes the research scope. Section 1.8 outlines the layout of the thesis.

1.2 Motivation

Current smartphone evolved from Personal Digital Assistants (PDAs) and Cell Phones and gradually being enhanced capabilities with each coming year. The rapid growth of Smart Internet Devices (SIDs), especially smartphones in terms of swelling functionalities such as graphics, high speed processing, storage capacity, and sensing features, has led to the device being the first choice of communication tool for people across generations. Moreover, the explosion of smart mobile applications such as YouTube, Facebook, Twitters, Google Maps and a wide range of other functionalities such as sensors, cameras, navigators, sounds and text editors has been the factors of mobile device being an integral part of our daily lives.

Furthermore, Smartphone usage has experienced significant growth in the recent years. The latest survey conducted by Cisco in 2015 shows that the mobile-connected devices are more than the population of people on earth and by the current pace the increase will be 1.5 mobile devices per capita in 2019 ("Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update 2014–2019 White Paper," 2015). Despite the numbers and increasing popularity of Smartphones, there is always a tension remains in between the increasing demands for Smartphone features (market demand) and Smartphone performance.

The modern mobile devices comprise features such as advance displays, high processing speed, network adapters (Wi-Fi, 3G, 4G), powerful data storage, advanced 3D graphics which reflects Moore's Law "the integrated circuits double every two years". Conversely, the practice of all these features boosts the energy depletion of the portable devices. The functionalities and extra features are burdening the resources-constrained mobile device especially on limited battery power, while the battery improvement remained historically slow, curtails the operational time of mobile devices to few hours.

Consistent with Moore's law, the capacity of mobile battery hardly doubles in a decade (Mack, 2011). The improvement of battery technologies cannot keep pace with the rapid growth of energy demand required by the new power-hungry mobile applications. Also, due to emerging high computational intensive applications, for instance, speech recognizers, natural language translators, online video games, and wearable sensors in the mobile computing environment coupled with increased user's expectations, while battery life, limited processing and storage memory remain a big challenge (Shiraz *et al.*, 2013).

To sum up, mobile devices have turned to an integral part of our daily lives due to mobility, convenience and convergence; however, the operational time of mobile devices especially of smartphones curtail to few hours only due to the diverse functionalities such as, natural language translation, playing games, browsing, audio/video, touch screen and sensing features. In spite of all the recent advancements, mobile devices are still low potential computing devices due to the limitation in CPU speed, memory capacity and battery power (Bheda & Lakhani, 2013). Besides, in pervasive computing, smartphones have brought a new rich user experience, but the

hardware performance is still inadequate due to limited capacity, thus restraining potential applications too. Therefore, the battery of SIDs in particular smartphones needs improvement, as the explosion of new mobile applications, multiple sensors and wireless interfaces drain battery swiftly.

1.3 Problem Background

Cloud computing, as a rich pool of computing resources facilitates to augment the computing capabilities and energy constraints of resource-limited mobile devices by providing a leased infrastructure, platform and software applications as services. Mobile devices thus utilize the served cloud resources to replenish the limitation of processing and energy hassles. The process where mobile devices approach cloud resources through mobile cloud applications is termed as Mobile Cloud Computing (MCC). Furthermore, researchers have been attempting to curtail the battery consumption by adopting different techniques either on the application side or on hardware (managing resources) side. Amongst many other proposed solutions, the technique utilized to minimize the computational load of mobile devices is to offload the complex tasks to a remote server for processing (Son & Lee, 2017; Wolski *et al.*, 2008) and is called computational offloading. The concept of computational offloading is not new (Yang *et al.*, 2008a). This concept dates back to the concepts of load balancing in the early 70s once used in distributed systems.

However, always offloading to remote servers is not energy saving (Kumar & Lu, 2010). If the computation required is low or if data needed to be exchanged is large whilst the available bandwidth is small, then execution of the task at mobile device is energy efficient rather than offloading the task. In order to reduce Round Trip Time (RTT) and save mobile's resources (energy, processing time) while accessing the resources of cloud servers, the compact size, mobility nature and wireless access medium of mobile devices always requires a lightweight (easy to execute locally or required less computations) framework or model to process the computational intensive tasks faster. Thus, computational offloading can be energy efficient only if all the necessary parameters are considered and a lightweight communication procedure is adopted. It is argued that the current computational offloading solutions for MCC are similar extensions of the traditional computational offloading frameworks and models for mobile

computing (MC). Additionally, the current approaches of computational offloading are computational heavy and resources hungry. An offloading approach can be efficient only if it considers the required Computation “C”, the size of Data to be exchange “D” and the available Bandwidth “B”, while most of the approaches previously proposed are resources intensive, as they failed to encompass the three basic parameters (Kumar & Lu, 2010). The three basic parameters, C, D and B, are the backbones for any computational offloading model to surface efficient results. Unfortunately, all the three parameters are avoided partially or fully to design the computational offloading frameworks (Griera Jorba, 2013). For instance, if there is no any mechanism to deal with the huge size of data to be exchanged both sides (transmitting and receiving), it will hit the power more, due to longer communication time. Similarly, if the required computation “C” is small enough then it is better to process locally rather than to offload while if the available bandwidth “B” is limited then the offloading time increases, which ultimately hits the battery life.

Some of the recent research techniques proposed for computational offloading (Chun *et al.*, 2010; Cuervo *et al.*, 2010; Lu *et al.*, 2011; Moghimi *et al.*, 2012) need runtime migration of computational tasks and configuration of ad-hoc resources platforms. Additionally, most of these techniques are time consuming (increased execution time) and resource intensive (occupy CPU for longer time) which ultimately drain power. The concept proposed by Sathan (2009) is named as context-aware computational offloading, where embedded sensors (additional sensors) are deployed to gather contextual information before making an offloading decision. It must be noted that using sensors itself is a power intensive process. Furthermore, in case of any sensor failure, the system needs to restart and restore itself to the last working state, thus is a time consuming and power draining process too.

In addition, the utilization of cloud resources in traditional offloading frameworks/models takes place at VM level, application level, task level, class level and method level. Several of the traditional computational offloading frameworks/models have developed outsourcing running instances of mobile applications (Cuervo *et al.*, 2010; Huang *et al.*, 2010). The method of outsourcing running instances to remote servers incorporates additional costs of running application’s states saving on mobile devices and then reconfiguration of application based on the saved states on remote services. This whole process requires additional mobile resources. Moreover, continuous

synchronization is needed for the management of runtime distributed platforms between mobile devices and remote servers; as a result, the mobile device needs to be in active state which is ultimately a power consuming process. VM level computational offloading is an example of outsourcing running instances to remote servers. Offloading based on VM level involves extra computation at local device. This is due to the process of creating an instance of the virtual machine on mobile device; and then pausing the running application and creating a state file. The file needs to save the memory states of the application; then to encapsulate the state file into VM instance and offload application with state file to remote computing environment (Shiraz *et al.*, 2013). This whole process increases computation at the mobile device by using maximum resources for a long time. In addition to that, the transfer of state file along with the application encapsulated into VM also increases the size of data to be exchanged, which ultimately increases RTT. Furthermore, runtime transmission of data files and binary codes of the application used in some of the traditional approaches increase the data transmission overhead in wireless network medium. It causes longer RTT which ultimately drains power at local device (Kosta *et al.*, 2012).

Similarly, offloading basis on class level, task level, and application level increases size of the communication data because every application has some lightweight methods that can be processed locally, while the heavy methods can be offloaded and hence executed remotely. Subsequently, if the whole application migrates for processing remotely, it also offloads the lightweight methods and thus increases the communication data size. Likewise, class level offloading migrates the whole class along with the lightweight methods and task level offloading sends the whole task to remote servers, which requires delegating the lightweight methods also, for the completion of the task. Thus, the traditional offloading frameworks and models utilize maximum resources of the mobile device prior to offloading and therefore, increases the size of communication data during offloading, which is a computational and resources intensive process. In addition, extra resources management is needed to handle offloading whilst adopting the VM migration or the whole application migration. Amongst the previous proposed computational offloading approaches, one such approach is a method level computational offloading approach, which has to counter the huge size of data communication. This approach in computational offloading reduces the size of communication data “D” partially to the smallest offload-able unit (method) which precisely divides application

into light and heavy methods. Hence, it reduces the unnecessary components of application to offload. However, the dynamic partitioning of application at method level, remote processing of the application, and huge size of data carrier files, cause additional computations at mobile device, as a result, turns the traditional computational offloading models to resources intensive for mobile devices.

1.4 Problem Statement

The approaches available in literature use VM (Virtual Machine) deployment and management, which are resource intensive in terms of overhead transfer and consumes extra battery (Chun *et al.*, 2011; Satyanarayanan *et al.*, 2009). The computational offloading approach based on VM migration is amongst the cloud based application processing mechanisms, which takes in encapsulation of mobile application in VM instance and delegates the instance to cloud node. The challenging part of computational offloading based on VM migration is the heavyweight resource intensive method which requires additional mobile's computing resources to manage and deploy VM at remote server node (Shiraz *et al.*, 2013). In order to decrease communication data size "D" the concept of method level was developed, where the delegation of intensive parts is based on "methods", the smallest unit of application. Only the intensive methods are identified to be either runtime or compile time, and then the methods annotated as "light" are processed locally while the methods annotated as "heavy" are delegated for remote processing.

The three most recent researchers used the concept of method level offloading: Rim (2006) used *DiET*, Kosta (2012) developed *ThinkAir*, and Shiraz (2013) developed *EECOF*. All of the research works mentioned minimized the offload-able part to the smallest unit but the process of partitioning application into methods adopted is either time consuming or resources intensive. For instance, Rim (2006) used *DiET* as a slim code generator, which takes offload-able source code as input and generator byte codes to reduce the size of data to communicate. *DiET* itself is a process of capturing mobile's resources for longer time due to generating byte codes; as a result, it hits the battery power. The concept used by Kosta (2012) and Shiraz (2013), is the dynamic partitioning of applications into methods. The partition of application at runtime (dynamic) requires the application to be genius enough to decide based on the previous offloaded pattern or

context gathered about the surrounding execution environment and divide the application into light and heavy parts. The process of dynamic partitioning causes additional computation at mobile device. Moreover, utilizing cloud resources and services used by Kosta (2012) and Shiraz (2013) for augmenting efficiency of mobile devices in terms of execution time and battery life hits by the long run RTT of distant cloud. Hence, the previous computational offloading solutions based on the dynamic partitioning techniques and then offloading the intensive tasks to distant cloud are not fully effective in making computational offloading an energy saving solution; as it causes additional computations and increases the size of data to be transferred and received. The focus of this research is to address the overhead local execution, the size of data to reduce for transmitting as well as to address the long run RTT of distant cloud by proposing a new lightweight method level computational offloading model.

Summing up, the traditional frameworks/models are not successful enough in minimizing the transfer load (data size), since it generates additional computation overhead due to dynamic partitioning, runtime migration and states file transfer. Consequently, computational offloading becomes an energy intensive and time-consuming solution. In addition, the delegation of intensive tasks to cloud server increases RTT due to multi-hop distance, which ultimately affects the results in terms of reducing battery consumption. Hence, by computational intensive procedure of traditional offloading frameworks/models, the offloading of complex tasks to distant cloud servers is not always energy saving. Therefore, the proposed offloading solution takes place by a dynamic partitioning of application at method level, which also carries some static features in order to reduce the overhead computations. By using the novel partitioning technique, the computational intensive methods will be identified during compile time before taking an offloading decision and then executing the methods at surrogate machine connected at a single hop distance.

1.5 Research Questions

1. How to increase the performance of existing applications partitioning techniques for the purpose of reduction of computations and handling of the dynamic network changes?
2. How does cloud server/surrogate machine affect RTT?

3. What are the possible offloading methods that reduce the communications data size?

1.6 Goal and Objectives

The goal of this study is to develop a model for power efficiency of mobile devices through lightweight method level computational offloading. In order to achieve the goal, the research objectives are identified as:

1. To develop a novel dynamic application partitioning method for the reduction of computations and handling of the dynamic network changes.
2. To design the cloud server/surrogate machine for the execution of intensive tasks in order to reduce long run RTT.
3. To develop and evaluate a lightweight offloading method for reduction of communication data size.

1.7 Scope of Research

The scope of the work is to develop a model for addressing the limitations of mobile devices, especially the limited battery timing. The research is limited to android devices produced after 2015 owing to the fact that the device under test (*DuTs*) used in this research are manufactured in 2015 onwards.

Furthermore, different OS generally requires different approaches to tackle the limitation issues of mobile battery. Android is Linux-based, comparatively open source and is more PC-like than iOS. The interface and basic features are generally more customizable from top to bottom. However, iOS is completely different in features and uniform in design, which runs by apple device only. Android is the world's most commonly used platform and used by many different phone manufacturers. Both android and iOS have their own play stores (Google Play or Apple App Store) which possess their own compatible applications only. Therefore, due platform differences, feature and application differences, both the platforms require different approaches to tackle the issues in limitations. This study is limited to android OS as the experiments are planned

to be conducted with android OS only and therefore it will not be applicable for Symbian and IOS.

The communication medium will be Wi-Fi and 4G for offloading the intensive task for remote processing. 3G is excluded as it has a limited bandwidth communication medium which is energy intensive rather energy saving. The model will be developed to test two parameters only, which are Execution Time (ET) and Energy Consumption (EC). Both the parameters ET and EC are inter-related as, if ET increases, it increases the EC too. The efficiency of devices will always be measured by considering these two parameters.

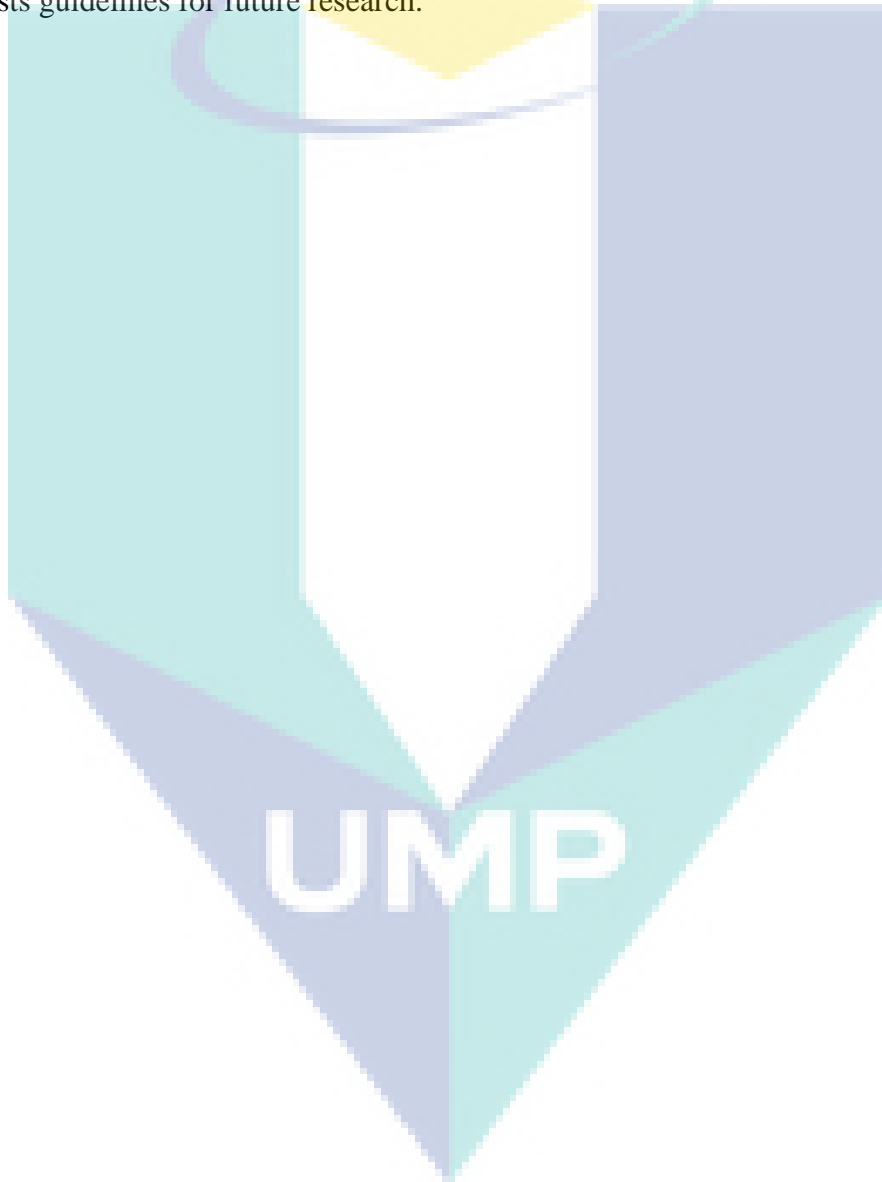
1.8 Thesis Organization

This thesis is divided into six chapters.

Chapter 1: Introduction: This chapter introduces the theoretical framework and motivation for the proposed research. It discusses the problem statements, states the objectives, scope and describes the methodology used for the proposed research as well the thesis layout. Chapter 2: Literature Review: This chapter presents the epistemology of mobile cloud computing and reviews the state-of-the-art in application offloading for mobile cloud computing. It classifies current offloading frameworks / models on the basis of thematic taxonomy and compares current offloading models on the basis of significant parameters. The challenges to traditional offloading models and issues in cloud-based application processing for MCC are also identified. Chapter 3: Methodology: This chapter consists of the research methodology carried out to achieve the research goal. It is starting from the analysis of the problem (gaps analysis) and then designing of the research question and research objectives. The next phase of this chapter includes the model designing and implementation. The implemented model is then to test in real mobile cloud environment, while the last phase presents the comparative analysis of the proposed model.

Chapter 4: Design and Implementation: This chapter proposes REST-Offload lightweight computational offloading model for delegating computational intensive task of mobile applications. It explains the model and proposed algorithms for computational

offloading. It gives details of different predefined parameters taken in account before offloading any task. Chapter 5: Results and Discussion. This chapter discusses the collected results with comparison with the traditional offloading systems and of local execution. It reports the tools used to collect data with special emphasis on computational intensity and of power consumption. Chapter 6: Conclusion: concludes the thesis with a commentary on the review of the research objectives. It highlights the outcomes of the research work cum the importance of the proposed solution. It states the limitations and suggests guidelines for future research.



CHAPTER 2

LITERATURE REVIEW

2.1 Overview

This chapter presents a review of the theoretical framework for Mobile Cloud Computing (MCC) that has been developed for saving energy of mobile devices. The chapter is organized into 9 sections. Section 2.2 gives a detailed background of Mobile Computing. Section 2.3 comprises different approaches adopted to augment mobile resources. This is followed by Section 2.4 which discusses limited battery problem of mobile devices and presents a taxonomy of mobile device's battery augmentation techniques. Meanwhile, Section 2.5 provides an analysis of CPU Clock time, Execution Time and Power Consumption. Next is Section 2.6 which provides the definition, detailed background and the hypothesis of computational offloading. Section 2.7 then discusses previous research works focusing on efficiency of limited resources of mobile devices and compares these frameworks based on a few defined parameters while Section 2.8 provides a review of current computational offloading frameworks as well as a comparative analysis of frameworks and establish a connection with the proposed solution. Section 2.9 presents analytical analysis of Method Level Computational Offloading Frameworks. Finally, section 2.10 concludes the chapter.

2.2 Background

This section discusses the background of cloud computing (CC), mobile computing (MC) and mobile cloud computing (MCC). Furthermore, it discusses the different approaches adopted to augment battery life of SIDs. This section also critically analyses the traditional computational approaches.

2.2.1 Mobile Computing (MC)

In 1990s, ideas of ubiquitous computing (i.e., mobile computing) were defined as technologies that would bring human computer interaction to an absolutely new level. The pervasive nature of smartphones has been proposed by Mark Weiser (1991) with the concept of ubiquitous computing as noted by (Saha *et al.*, 2003) “After the mainframe era, where people used to share a single machine, personal computers where one-to-one human computer interactions took place, the next era will be ubiquitous computing (the era of calm technology) where the technology will disappear”. Weiser (1991) had hoped for a world to be created where people could use and interact with computers without thinking about them (psychologically disappeared). Ubiquitous computing has provided a complete freedom from the mental presence to experience the rich number of services using the internet.

Mobile computing has progressed rapidly and become one of the powerful trends in the development of IT, commerce and industrial field. It has revolutionized, how people work and deal with their daily lives. In addition, with the development of wireless technology like WiMAX, Ad Hoc Network, Wi-Fi, 3G and 4G, users could surf internet much easier and would not be limited by any physical link with a static position or place as before. Thus, mobile devices have been accepted by an increasing number of people as their first choice for communication, at work and for entertainment in their everyday lives. The transmission of data without the connectivity of any physical link is one of the basic features of mobile computing. This has given rise to the increasing number of users of mobile computing.

Gartner, which is a famous analytical and consulting firm, predicted that by 2013, mobile devices would replace PCs to be the most common web access tools in the top ten strategic technology trends and by 2015, the firm predicted that smartphones would dominate over 80% of the mobile phone mature markets (Orlando, October 8, 2013). Furthermore, Business Insider's (BI) Intelligence (2015) had anticipated that the use of smartphones at a global scale would have significantly increased from 5% in 2009 to 22% by the end of 2013, that is, an increase of nearly 1.3 billion smartphones within four years.

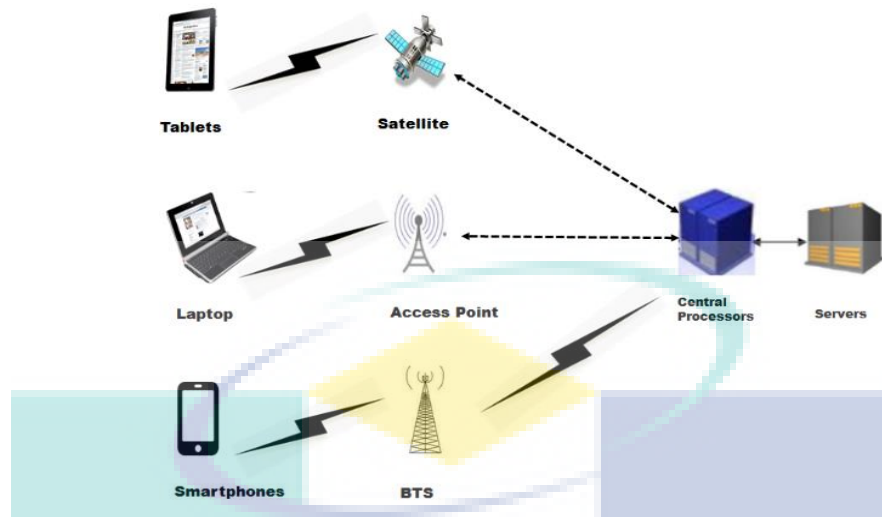


Figure 2.1 Framework of Mobile Computing

Mobile computing is defined by Wikipedia as “mobile computing is a form of human computer interaction, while the computer is expected to be transported during normal usage”. There are three main components which collectively form mobile computing, namely, the hardware, the software and the communication (Qi *et al.*, 2012). Hardware refers to the actual mobile device (e.g., smartphone and laptop or their components) whereas software refers to the number of applications running in the mobile device, such as the antivirus, internet browsers and games. Meanwhile, communication includes the setup of the mobile networks, protocols and the delivery of data between the devices. Figure 2.1 shows a framework of mobile computing. The central processors in the mobile network receive user request through Base Station and pass it to the servers for the required services. In response, the servers release the desired services and the central processors deliver services back to the user.

2.2.2 Cloud Computing (CC)

In the history of computing, a stepwise evolution can be seen from mainframe computing until cloud computing (CC). The feature of unlimited availability of resources makes CC a superior distributed computing model than grid computing. CC provides the ultimate solution of keeping pace with the development of technology and that is the magic of Moore’s Law (Qi & Gani, 2012). Since 2007, CC has become popular and most

significantly researched topic. Due to the different perspective of numerous developers and organizations, it is difficult to define cloud computing in a distinct way.

Consulting Firm Accenture has set a useful, brief definition of CC as “the dynamic provisioning of IT capabilities (hardware, software, or services) from third parties over a network” (Adrees *et al.*, 2016). According to the convention set by the National Institute of Standards & Technology (NIST):

Cloud Computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management efforts or service provider interaction (Mell et al., 2011).

The main objective of the Cloud Computing Model is to increase the capabilities of client devices by augmenting the proficiencies of the device’s own resources through accessing cloud infrastructure and software instead of possessing them. In CC, the services provided by the service providers over internet are commoditized like traditional utilities such as water, electricity and telephone. Consumers avail the resources on demand fashion and they pay as they use (Buyya *et al.*, 2009).

Amazon Web Services (AWS), Google Apps Engine, Aneka and Microsoft Azure are examples of public utility computing which are delivered at low cost by the Cloud providers (e.g., Google, Amazon and Salesforce). AWS allows infrastructure and software as services, which enable users to manage virtualized resources in Cloud datacentre. This decreases the hardware and software costs and the extra efforts of an organization in providing services. AWS also allows the utilization of Simple Storage Services (S3), the unlimited storage capacity for personal data in cloud datacentre by online file storage web services. The computation is performed on the data by Elastic Cloud Computing or EC2 (Kristensen, 2007). It was believed that Amazon S3 had a reputed storage of more than two trillion objects as of April 2013 ("Aamazon S3," March 14, 2006).

Meanwhile, Google Apps Engine provides a unique powerful application development platform in cloud data centres. The well-known development tools like Java and Python are used by Apps Engine for the independent development of applications (Chun *et al.*, 2013). As for Microsoft, its Windows Azure is an open and extensible cloud computing platform for the development, deployment and operation of applications and services in datacentre. Azure offers a simple, widespread, and a powerful platform for the designing of web applications and services (Windows Azure, 2010).

The service-oriented Cloud Computing model basically consists of four layers. These layers are Data Centres, Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS), as shown in Figure 2.2. The cloud physical resources are the hardware resources in datacentres. To access the physical resources, virtual machines are installed. Hypervisor (middleware) is used to access the physical resources (Hardware) and is responsible for the placement and management of virtual machines. Both layers consist of physical resources and virtual resources which fall in the category of IaaS. The third layer, namely, PaaS, comprises the application hosting platform, which provides a cloud programming environment and monitoring tools such as admission control, QoS negotiation and pricing and billing. The fourth layer, SaaS, consists of all the cloud applications running on virtual machine instances in a complete secluded form.

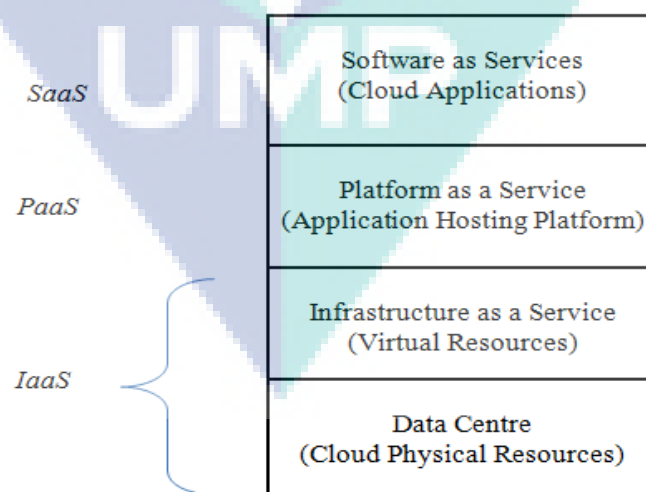


Figure 2.2 Service-Oriented Layered Architecture of Cloud Computing

2.2.3 Mobile Cloud Computing (MCC)

The CEO of Google Eric Schmidt was reported to have anticipated this in 2010 as: “based on Cloud Computing services development, mobile phones [would] become increasingly complicated and [would] evolve to [become] a portable super computer” (Qi & Gani, 2012). Aepona defines MCC as “a new distributed computing paradigm for mobile applications whereby the storage and the data processing are migrated from the SID’s to resources rich and powerful centralized computing data centres in computational clouds” (Shiraz *et al.*, 2013). The terms resource rich and resource scarce are used for static computers and mobile devices, respectively. The mobile device was initially considered for limited use. The lightweight and mobility features have turned the device into a computing tool that is resources limited. Therefore, the performance of a mobile device is restricted by its limitations. These limitations, as noted by Satyanarayanan *et al.*, (1996), are the mobility features of the device’s inherent problems such as low connectivity, resource scarceness and finite energy. The comparison in development of the resources capacity of mobile devices and static computers is presented in a series of five-year gaps from 1997 until 2016 as shown in Table 2.1.

To deal with the low capability issues of devices, CC has turned out to be a ruling model which efficiently overcomes the resource scarceness problems by remote computation and utility services. Hence, by offloading and remote computation technique

Table 2.1 A five-year-gap comparison of advancement between Static Servers and Mobile Devices (1997–2016)

Year	Static Servers			Mobile Devices		
	Processor	Speed	Memory	Processor	Speed	Memory
1997	Pentium I	266 MHz	256MB	Palm Pilot	16 MHz	512 KB
2002	Itanium	1 GHz	512MB	BlackBerry 5810	133 MHz	1 MB
2007	Core 2	3.0 GHz (2 core)	1GB	Cortex A8	600 MHz	256 MB
2012	Xeon X3	3.1 GHz (4 Cores)	2 GB	Samsung Galaxy S3	1.4 GHz (4 cores)	1 GB
2016	Intel Core i7	3.9 GHz (4 cores)	8GB	Samsung Galaxy Note 7	1.6 GHz (8 cores)	4 GB

CC addresses the techniques, CC addresses the inherent issues of mobility using the remote resources, provided by the service providers. The big players in the list of service providers are Google, Amazon, Apple, Facebook, and Yahoo. The Cloud providers offer such infrastructure where both the processing and data storage exist outside of the mobile device termed as *mobile cloud*. Thus, mobile cloud computing (MCC) is a novel model which encompasses CC, MC and Networking.

Figure 2.3 shows a framework of mobile cloud computing (MCC). The model framework composed of mobile computing (MC) and cloud computing (CC) are bridged by Internet. The mobile devices are connected to a network which establishes and controls the connection between the networks and mobile devices through base stations such as BTS, access points, and satellite (Dinh *et al.*, 2013). The user request is then processed and forwarded by central processors to the servers; providing network services. Finally, the requests are transferred to cloud through internet and the cloud controller process the requests and provides the desired services to subscribers.

2.3 Approaches for Augmenting Mobile Resources

There are four main approaches used to augment mobile resources. The approaches are as follows:

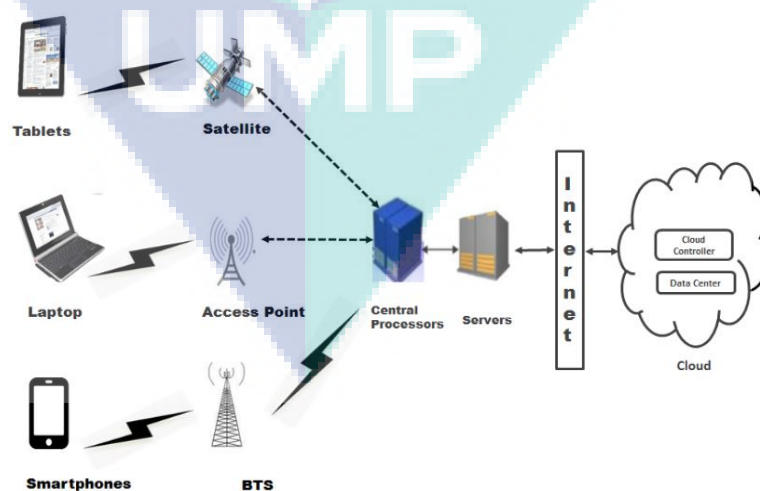


Figure 3.1 A Typical Framework of Mobile Cloud Computing

- i. Generation of New Hardware Resources;
- ii. Slow Execution of Program;
- iii. New Applications for Resource Constrained Devices; and,
- iv. Hardware and Software Management Techniques.

2.3.1 Generation of New Hardware Resources

The root cause of limited battery timing is the very slow development of battery technology. This had been anticipated by Moore (1965, 1975) decades ago who stated that “the development of battery technology hardly doubled in a decade”. In order to tackle limited battery capacity issue, the following two possible approaches should be considered.

2.3.1.1 Need of a New Generation Semiconductor Technology

The current semi-conductor technology has made the transistors smaller. In other words, the transistors consume less power. Nevertheless, due to the smaller size, more transistors are needed to achieve functionalities and produce better performances. Thus, increasing the number of transistors is actually a burden on the power source of a mobile device which ultimately consumes more energy (New Semiconductor Research, 2014).

2.3.1.2 Replenish Resources (Battery) by External Action

Human movements and solar light (e.g., nanotechnology) are some of the possible alternate replacements of the existed mobile’s battery in future. These new technologies may perhaps overcome the issues of limited power of the mobile devices.

2.3.2 Slow Execution of Program

With the increase of the CPU speed, battery consumption would significantly increase, that is, if the processor Clock Speed doubles, the speed of the power consumption would become nearly octuple (Kumar *et al.*, 2010).

2.3.3 New Applications for Resources-constrained Devices

Another approach to augment mobile resources is to rewrite new applications for resource-constrained mobile devices. This approach may seem impractical, costly and pushes towards ad-hoc applications (Dinh *et al.*, 2013).

2.3.4 Hardware and Software Management Techniques

Numerous approaches had been used in the past to curtail the power consumption of SIDs. A few of the very basic approaches are geared towards managing the computing resources of SIDs by optimising the operating systems and software tools in a way that will consume limited power. Some of the hardware and software management techniques are briefly explained in this section. Any of the approaches explained in this section can be adopted to reduce power consumption of mobile devices.

2.3.4.1 Avoid Wasting Energy

Waste of energy can be reduced by avoiding unnecessary processing, better management of resources and setting the components on standby or sleep mode (Balasubramanian *et al.*, 2009; Vallina-Rodriguez *et al.*, 2013).

2.3.4.2 Reduce Resources Requirement

Context-aware mobile applications need to be developed. Such mobile applications could make the device be aware enough of when and what to process and when as well as what not to process in order to reduce the unnecessary use of mobile resources.

2.3.4.3 Fidelity Adaptation

Fidelity adaptation manages the compromise between resources consumption and application quality. Although fidelity adaptation technique deteriorates the quality of results, it allows the execution of applications when there is no other solution to run the application in standard mode.

2.3.4.4 Cyber Foraging/ Computational Offloading

In this technique, the execution of programs is eliminated altogether and the heavy computations are sent to remote servers. If the mobile device utilizes its own local resources solely to perform executions of the complex applications, this will drain its battery faster and in some cases, the execution is even not possible for some kinds of application due to limited processing speed. By computational offloading, the load can be eliminated to augment the device's own resources.

Numerous researchers have attempted and achieved success up to some extent in saving the power consumption of the mobile device's battery. This literature review focuses on the last option, that is, utilization of powerful resource of cloud (Cloudlet) instead of local limited resources of mobile device. In this way, the burden is not solely carried by the mobile device. Rudenko *et al.*, (1998) was the first to introduce the term remote executions which are different from the traditional Client-Server architecture, where a thin client always migrates computational tasks to a server in the same local network. By contrast, in remote executions, the offloading process is accomplished with the computing devices which are outside of the immediate computing environment. Satyanarayanan *et al.*, (2001) presented the concept of remote executions by accessing nearby available machines (i.e., the surrogates) to execute complicated computation on behalf of handheld devices. Although useful, Satyanarayanan *et al.*, (2009) work does not put sufficient focus on power saving issues. Satyanarayanan *et al.*, (2001) termed such type of computing as *Cyber Foraging* or *Surrogate Computing*. In this way, the local execution of the entire complex task is eliminated. Figure 2.4 shows the possible ways of augmenting the mobile resources.

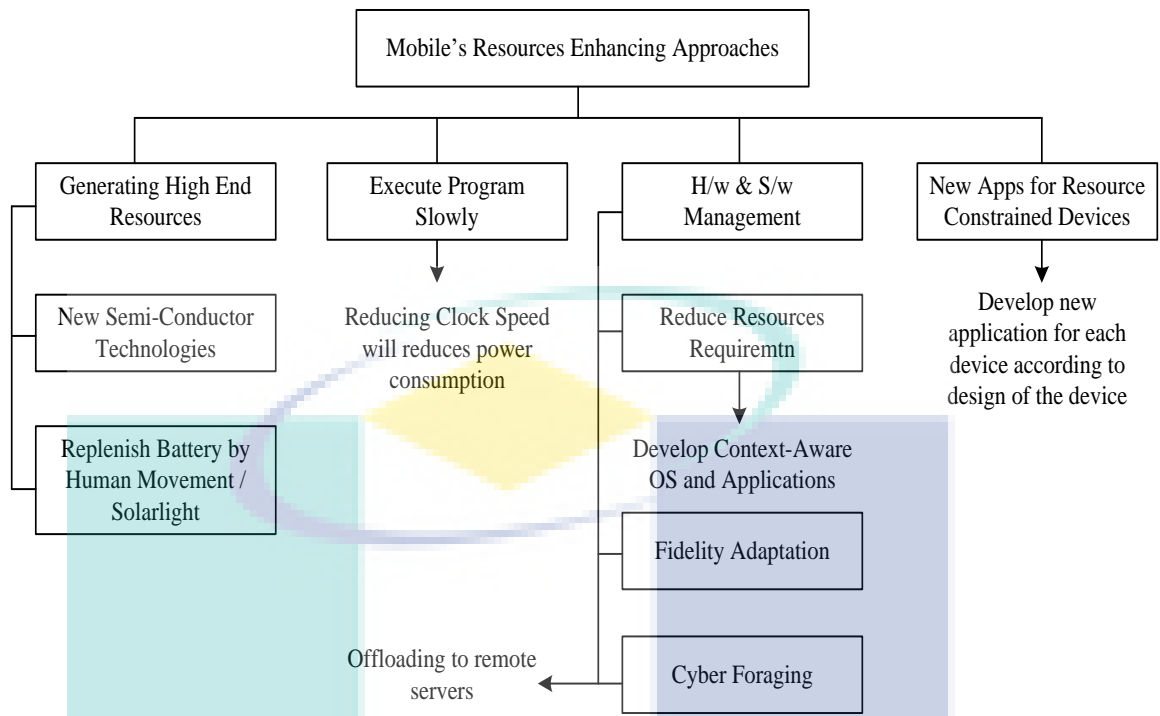


Figure 2.4 Mobile's Resources Enhancing Approaches

2.4 Taxonomy of Mobile Device Battery Augmentation Techniques

Battery is an element that permits the mobility as a luxury feature in the first place. Hence, focus should obviously be on the improvement of energy usage in order to prevent mobile device from becoming stationary due to low bandwidth and resource-hungry applications.

Most of the mobile devices use Lithium-ion batteries (Rodriguez & Crowcroft, 2003). These batteries are comparatively better power sources available in all brands and models of mobile devices. Nevertheless, battery technology has shown that the only substitute left to solve the issue of limited power of mobile devices is by reducing the power consumption at hardware level and designing more power-efficient operating systems and applications. Ongoing research conducted by hardware manufacturers and OS designers has led to some new solutions using augmentation techniques at different levels such as Hardware, Wireless technology, Operating System and Applications (Rodrigues & Crowcroft, 2003). Figure 2.5 shows the taxonomy of smartphone's battery augmentation at different levels.

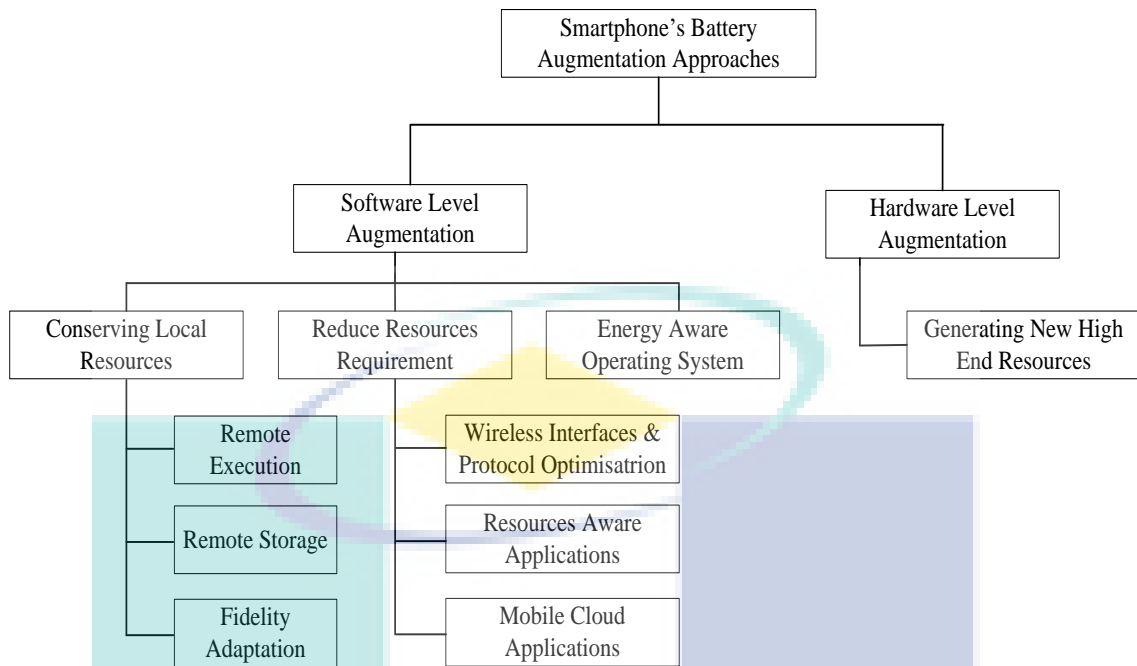


Figure 2.5 Taxonomy of Mobile Device Battery Augmentation Techniques

2.4.1 Hardware Level Augmentation

The first step in hardware level augmentation is to enhance the capabilities of mobile local resources such as high speed multi-core processors, storage and long lasting batteries (Wang *et al.*, 2011). However, the development in battery technology is unfortunately not progressing at the same speed as that of the processors and storage while it is the only un-restorable resource which cannot be renewed without the help of any external resource (A.D, Jan, 2013). Many efforts have been made since 1990s to replenish energy from different sources such as human movement, wireless radiation and solar energy, which are not adequately effective to minimize energy deficiency (Abolfazli *et al.*, 2012). However, this is presently still being researched. Similarly, large screen and data storage also increase the power consumption due to additional weight and size (Perrucci *et al.*, 2011). Large data storage and retrieval have been proven to be power hungry. Hence, memory increase contributes to faster draining of the battery.

2.4.2 Software Level Augmentation

Software level augmentation of smartphone batteries consists of three main categories, namely, energy aware operating system, conservation of local resources and reduction in resources requirement.

2.4.2.1 Energy Aware Operating Systems

Two kinds of programs run in the mobile device, namely, the operating system (OS) and the applications. A question now arises, that is, which one should be responsible for energy management. Some of the researchers have suggested the application level energy management is the best approach as noted by Liu *et al.*, (2005). However, this approach lacks of the main entity responsible for monitoring and supervising the resource's consumptions by other applications (Rodrigues & Crowcroft, 2003).

On the other hand, some researchers prefer the OS to monitor and manage energy resources. Considering OS alone to be responsible for energy management, this solution may lead to a problem of scalability; hence, there are researchers who have suggested a hybrid model (Vallina-Rodriguez & Crowcroft, 2013) in which both the applications and the OS should be aware of the resources utilization and supervision. Examples of such OS are Odyssey and Ecosystem.

Normally, the OS must be made aware of the application's energy demand and the available energy level until the next charging facility. Moreover, new programs, scheduler, models and energy measurement tools should be developed in order to reduce energy consumption and support software level energy measurement. Some of the OS level techniques proposed for power utilization are Hard Disk Management, CPU scheduling, Screen blanking.

2.4.2.2 Conservation of Local Resources

In 1990s, the best way to conserve the local resources of mobile devices was to reduce the workload on the local resources. Later on, the concept of remote execution, remote storage and fidelity adaptation were introduced to conserve local resources.

a. *Remote Execution*

Remote execution is a process involved to transfer the executable codes, control, computational data or any compute intensive application through a network to any local server machine called the surrogate where the execution of computational task takes place. The results are then transferred back to the handheld Client.

The remote execution concept was presented to divide the load of local resources with the remote stationary devices. Rudenko *et al.*, (1998) had first introduced the idea of remote execution to save workload of the local restricted resources and conserve mobile energy. The same idea was introduced by Satyanarayanan *et al.*, (2001) in a broader way and proposed the concept of remote executions. This concept was not only used for conserving the power, storage, and processing efficiency of the device but also to enable the execution of computational intensive mobile applications which might not be possible to be processed locally. The technique of remote execution is known as process offloading or cyber foraging. The term cyber foraging was first introduced by Satyanarayanan for augmenting the computing potentials of resource-constrained devices by exploiting the potential of rich-computing resource devices available in the local environment (Satyanarayanan, 2001). The rich-computing local devices are termed as the surrogates.

The Aura Project at Carnegie Mellon University presented the concept of two techniques, namely, cyber foraging and fidelity adaptation, which seemed promising in terms of substantial power saving (Garlan *et al.*, 2002). In the scenario of cyber foraging, the surrogates can be used by data staging or by computing the surrogates. Data Staging is the technique used with the surrogates to get rid of the long latencies (long RTT or Round Trip Time), If the mobile device itself to request for web data situated far away from handheld device, and then wait for the distant file to receive, the surrogate machine will prefetch the data from distant server in advance, in order to reduce RTT. This increases the user's response time and more importantly, the high latency which causes power consumption.

In this case, the surrogates are used to fetch data for mobile device by data staging. Mobile client send request to surrogates for the distant file. The surrogate fetches the file and Client device retrieves the file from nearby surrogates. This will reduce the RTT as

well as the power consumption. If the data staging technique is coupled with some predictable software, then it will let the surrogate to pre-cache all the files which the client device needs to use next. This will be more effective in terms of RTT and power consumption. By using the technique of computing the surrogate, the client device requests to the nearest surrogates for computation on its behalf. In this scenario, if the Client device detects that its own battery level is not adequate to perform the computation locally, it will send computational intensive file to the surrogates and will obtain the results back on its screen. Research has revealed that remote execution is highly effective in reducing power consumption of mobile devices (Flinn *et al.*, 2002).

b. Remote Storage

The outsourcing of data storage at third location, somewhere in the local network or remote clouds, enhances the storage capacity of handheld devices. A number of remote accessible file storage services are available in the cloud which extend the storage capabilities by providing off-device storage services. Examples of such services are Amazon S3 ("Amazon S3," March 14, 2006), Dropbox (Balan *et al.*, 2004) and Google Docs (Flinn *et al.*, 1999). The online storage services (remote storages) not only provide the facility of unlimited storage space but also ensure the safety and reliability of data storage. Thus, the remote storage, especially the cloud storage services, boosts the limited storage capacity of mobile devices and enables the mobile users to store and access any kinds of data anywhere in the cloud and retrieve the data from anywhere by using a web browser. The writing of data to the local storage and the retrieval of data will consume more power as compared to remote storage of such data. Thus, it is power-efficient to manage big data for remote storage which does not need to be accessed too often. As a result, battery energy can be reserved adequately.

c. Fidelity Adaptation

The cyber foraging can be effective if the surrogate devices are available but what will happen when there is no access to any surrogate device in the surroundings? This raises some concerns such as how the Client devices will minimize the load to reduce battery power. Fidelity adaptation is a technique that will work in such circumstances. Balan *et al.*, (2004) have defined fidelity as “the trade-off between the application’s

quality and power consumption”. For example, if a client device is in a video call and experiences a sudden power drop to a lower level, the video will automatically shut down while the voice call still continues.

The concept of fidelity, with regard to this adaptation technique, involves the release of CPU load and bandwidth. The runtime parameters can be modified so that an application can still be used by the user but in lower quality in order to reserve power, bandwidth and computational resources (Flinn & Satyanarayanan, 1999). For instance, a user watching a full colour video from the server will experience an automatic change in display from full colour to black and white when the bandwidth drops. Many approaches (Flinn & Satyanarayanan, 1999; Lara *et al.*, 2011) have used both cyber foraging and fidelity adaptation to enhance the device’s local resources and achieve better performance.

2.4.2.3 Reduction in the Local Resources Requirement

There are indeed numerous software developers. However, many of these software developers do not have comprehensive knowledge of energy-constrained handy systems such as those embedded in smartphones and personal digital assistants (PDAs). As a result, many mobile applications consumes an unreasonably huge amount of power. To augment the capabilities of mobile resources along with manufacturing the high-end hardware devices, a parallel development of resource-efficient application plays a vital role. This approach focuses on the design phase of software development to form energy-efficient applications.

a. Wireless Interface and Protocol Optimization

Among the other components of the mobile devices which contribute to power consumption, the wireless interface has a greater impact than that of any other component. A total of 10% of the overall power consumption in laptops is caused by the wireless interface while in case of smartphones, this ratio reaches up to 50% of the overall consumption (Kravets *et al.*, 2000). Mobile devices nowadays are equipped with several air interfaces such as GSM, Wi-Fi, 3G and 4G. Hence, it is vital to manage these interfaces either manually or by resources-aware applications to stop the excessive

draining of power. Although all the wireless interfaces provide a flexible choice for communication and saving energy, however, the existing communication protocols such as UDP and TCP do not give any advantages. For example, a communication which is established with 3G connection will last either until the session ends or until the interface becomes unavailable anymore even though the device is close to a better interface in terms of speed, energy saving and communication (Rahman *et al.*, 2013).

New technologies in cellular networks such as 4G LTE have been introduced to reduce the cost per transmission as compared to the previous standards (Frattasi *et al.*, 2006). In all the available interfaces of mobile devices, Wi-Fi is more power-efficient if it stays in continuous data transmission for larger data (Balasubramanian *et al.*, 2009). The only time Wi-Fi drains the battery is the idle time or scanning of Access Point (AP) to connect. Therefore, a common solution has been suggested by researchers (Bertozzi *et al.*, 2003) to explore transport protocols optimization for IEEE 802.11 networks in order to reduce the energy depletion of IEEE 802.11 standards with very little overhead. The study conducted by Bertozzi *et al.*, (2003) was based on the transport protocol which tackled flow control of data to regulate the network traffic. This played an important role in predicting the workload of the network interface. Further evidence has shown that tuning parameters in the protocol would form the activity profile of the network interface and make it more energy-efficient.

b. Resource Aware Applications

In order to protect smartphones from energy depletion, it is vital to understand the need of power consumption of the hardware components and also of the software installed. Many software developers have limited understanding of energy-constrained portable systems such as those embedded in smartphones and PDAs. As a result, quite a number of smartphone applications consume enormous amount of power unnecessarily (Zhang *et al.*, 2013). Thus, better understanding of the power consumption of individual mobile components (e.g., CPU, Memory, Wireless Interface and Screen) would contribute to reduction in significant amount of energy and development of better energy aware systems. For instance, in cases which the available air interfaces are 2G and 3G, a resource-aware application is needed to exploit 2G for voice communication and 3G for FTP services because of their different energy consumption needs (Perrucci *et al.*, 2011).

The memory and computational intensive applications are also considered power hungry. The compute intensive applications engage CPU for a long time to process complicated data thus directly affecting the battery of the mobile device. Both the increase in computational intensive applications and increasing memory size of mobile devices lead us to more power consumption (Perrucci *et al.*, 2011).

c. *Cloud-Mobile Applications*

Cloud mobile applications are identical to Web-based applications. The main similarity in both the applications is that they run on external servers instead of the client device itself. They require a browser on the client device to access them (Claybrook). Moreover, they both are designed for different operating systems and multiple mobile devices unlike native applications, which are designed for specific operating system and single device model only.

Native mobile applications are restricted by capacity of the battery and processing efficiency of mobile devices, which ultimately disrupt the speedy progress of these devices. The concept of cloud computing bridges this gap by offering cloud-mobile applications which have the capabilities of connecting the cloud servers for processing with the remote storage. The new concept of cloud and mobile agreement has generated this new era of rich cloud-mobile applications which are intended to curtail smartphones' resource consumption by utilizing rich cloud resources without changing the quality. Therefore, the development of cloud-mobile applications accelerates code execution by offloading computational intensive data to the cloud server and thus decreasing the overall execution time without using the mobile resources (Kumar & Lu, 2010).

Efforts have been made by numerous researchers for designing perfect cloud–mobile application to leverage the Cloud resources for mobile devices. For example, Lu *et al.*, (2011) have developed an architecture for rendering the mobile screen to the cloud environment. In screen rendering the remote code execution takes place and online data are stored. On the one hand, the cloud–mobile applications architecture helps mobile devices to augment the limited resources of mobile devices. On the other hand, the remote execution of intensive task by the traditional mobile application does not support the development of applications that incorporate the cloud features (Othman *et*

al., 2014). Therefore, the new mobile cloud application development is essential to augment mobile resources by remote processing.

The migration of resource-hungry and interactive portion of the screen for execution in cloud will certainly decrease the power consumption due to minimized computational burden of the local CPU and GPU. A similar effort made by Chun *et al.*, (2010), that is, the CloneCloud service, using the smartphone's internet connection to communicate with a full image (clone) of itself that exists in remote servers in the cloud. In order to execute intensive tasks, the CloneCloud is required to offload data to the server which possesses the clone of the device. Figure 2.6 shows the CloneCloud framework.

2.5 An Analysis of CPU Clock Time, Execution Time and Power Consumption

By developing cloud mobile applications, it is possible to utilize maximum resources of remote servers and minimize the workload of local devices. By offloading tasks, the CPU engagement (CPU execution time) will decrease. According to Perrucci *et al.*, (2011), the power consumption of CPU is directly proportional to the complexity of the instruction (workload of the CPU). To manage complex calculations and make possible execution of complex task locally, CPU clock speed can be increased to get maximum throughput. The processor performance can be assessed simply by calculating the number of operations per given time. That is, the Throughput “T” of CPU can be obtained as:

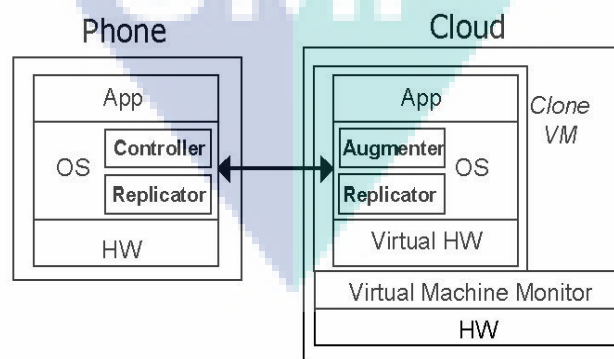


Figure 2.6 Clone-Cloud Framework

$$\text{Throughput (T)} = \text{Operations} / \text{Time (S)}$$

In terms of application, the throughput of processor can be plotted into three main categories as follows:

- i. Maximum Throughputs (T_m), for real time applications (spreadsheet updates, navigations, spell check, scientific calculations and for audio/video calls) as they would be required to response very quickly, in which the throughput should be maximum.
- ii. Normal Throughputs (T_n), for applications running in the background not affected by the delay, as they need to be processed in fractions, in which the processor throughput can be normal or low.
- iii. Zero Throughputs (T_0), in which processor can be idle and no throughput is desired.

The Maximum Throughputs (T_m) can only benefit to the process which are computational intensive and requires to have low latencies. The processes which are running in the background and of high latency cannot benefit from Maximum Throughput (T_m).

The increase of CPU clock speed or Throughput (T_m) reduces the execution time and make possible local execution but it directly hits the battery life, as by Kumar *et al.*, (2011), when a processor's clock speed doubles, the power consumption nearly octuples. If the clock speed is reduced by half, the execution time doubles, but only one quarter of the energy is consumed. Although CPU performance proliferate whenever the execution time decrease (Ramanathan, 2008) i.e.

$$P_{cpu} = \frac{1}{E_T} \quad 2.1$$

where P_{cpu} is the performance of CPU, while E_T is the execution time of a task. Additionally, execution time decreases if Cycle per Instruction (CPI) or Frequency (F) increases,

$$E_t = \frac{1}{f} \quad 2.2$$

From Equation 2.1 and Equation 2.2

$$\frac{1}{f} = \frac{1}{P_{cpu}} \quad \text{or} \quad f = P_{cpu} \quad 2.3$$

Equation 2.3 shows that the increase of CPU frequency will increase the performance. Although increasing CPU frequency improves the CPU performance to some extent, the increasing frequency of the CPU needs more power to operate. Hence, this will affect the battery power. To support extensive range of workloads efficiently, the modern CPUs are capable of adjusting their clock rate dynamically. For example, when the CPU is idle, it adjusts the clock rate to the lower speed and allows the voltage to be lower too. The lower voltage assists in reducing the CPU power consumption. Thus, fewer CPU Cycles (Frequency) means that the CPU requires less power and produces less heat.

However, there are two basic dilemmas. The energy-conscious design of the portable systems is vital without compromising performance (i.e., the CPU). A parallel work needs to minimize the workload of the battery while at the same time improve the CPU speed to handle the most challenging and most demanding applications. Thus, in order to maximize the total computation per battery life, the energy consumption per operation should be minimized. Furthermore, to prolong battery life, operations need to be reduced per battery life. This has been done using numerous techniques adopted in the past. One such technique is to offload the intensive computational activity of mobile device to a remote server. The server will then process the complex task on behalf of mobile device and send the result back to mobile.

2.6 Computational Offloading (Cyber Foraging)

The term computational offloading is also defined by the terms code offload or cyber foraging. The process of transferring some computer processing tasks to remote servers in order to discharge the execution load of task from mobile devices is called computational offloading.

2.6.1 Energy-saving Computational Offloading

Researchers in the past have agreed that computational offloading will always not be power-efficient (Kumar *et al.*, 2012; Satyanarayanan, 2001; Satyanarayanan *et al.*, 2009; Shiraz & Gani, 2014). Thus, the following two questions need to be answered before adopting computational offloading as a solution:

- i. What is the optimum condition for computational offloading?
- ii. What factors need to be addressed before starting computational offloading?

Kumar *et al.* (2012) have addressed this by evaluating the mentioned questions using the mathematical formula as:

$$P_C \times \frac{C}{M} - P_I \times \frac{C}{S} - P_{TR} \times \frac{D}{B} \quad 2.4$$

where:

C – is the number of instructions to be offloaded,

S & M – are the speed instruction/ second of server and mobile device respectively,

P_C – Mobile power consumption (watts),

P_I – Mobile idle power consumption (watts),

P_{TR} – Mobile power consumption during transmission,

D – Data in bytes to be exchanged,

B – Network bandwidth.

If the server speed considered is F times faster than mobile speed, then:

$$S = F \times M \quad 2.5$$

And by substituting Equation 2.4 in Equation 2.5, the formula can be rewritten as follows:

$$\frac{C}{M} \times P_C \left(\frac{P_I}{F} \right) - P_{TR} \times \frac{D}{B} \quad 2.6$$

In this regard, the values M , P_i , P_c , and P_{TR} are constant, and if Equation 2.6 provides a positive number, then offloading will reduce the power consumption of mobile device. The formula will provide a positive number if $\frac{D}{B}$ is sufficiently small (i.e., B is sufficiently large) and F is sufficiently large. In other words, if the bandwidth and server speed are sufficiently large, then offloading will reduce the power consumption. The relationship between B , D and C is important to predict whether or not to offload tasks. For instance, in large computation C , if communication data D is smaller and bandwidth B is large enough, then offloading will be beneficial; otherwise, for small C and low bandwidth B , it is useful to avoid offloading and process data locally. The relationships between B , D and C are illustrated in the Figure 2.7.

Computational offloading is a worthy solution to augment the resources limitations of mobile devices. However, this approach also has several limitations. Firstly, to accomplish the process of offloading some surrogates should be accessible and willing to share their own resources with others (PDAs, Mobile Devices) via wireless networks. Secondly, through cyber foraging, the security of confidential data cannot be guaranteed. Thirdly, cyber foraging is applicable only to the tasks which are transferable and not applicable to some tasks which are not transferable. In addition, offloading of small tasks may not be beneficial due to extra communication overhead or changing of network topology that may affect the offloading process too. Fourthly, computation offloading to multiple surrogates may cause the issue of load balancing.

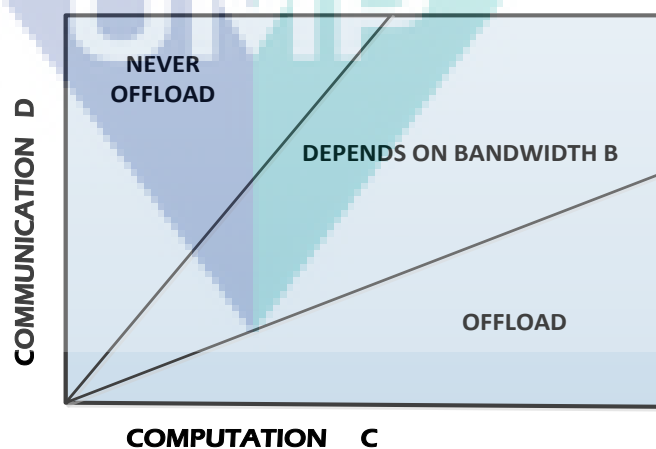


Figure 2.7 Relationship between B , D and C

In addition, to improve the efficiency of remote execution the required data needs to offload only in order to reduce communication overhead. Finally, the dynamic allocation of resources on demand generate the issues of synchronization and resuming (releasing) which ultimately causes latency. A good understanding of all the related issues is crucial to keep in mind before making the cyber foraging practical and useful.

2.6.2 Metrics of Computational Offloading

It is necessary to take into account certain metrics which are influencing the process of computational offloading, such as context specification, mobile and surrogate specifications, network specification as well as application specification. Figure 2.8 illustrates the computational offloading decision which is influenced by metrics that need to be considered. If there are no surrogates available in the surrounding to offload the complex computation, then fidelity adaption, which is the process of trade-off between quality and speed or quality and power consumption, would be considered. User can manually specify the low quality for better speed and this will reduce battery drainage. Furthermore, wireless networks have different type of features and bandwidths. Hence, a mobile device can connect and communicate through any kinds of available network such as Wi-Fi, 3G, and Wi-MAX. Therefore, computational offloading decision is strongly influenced by the different bandwidth of different networks.

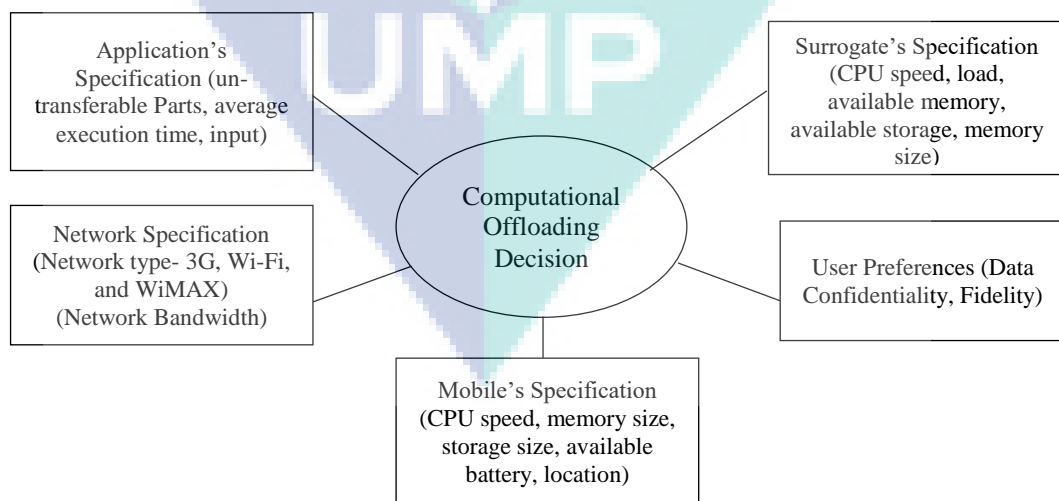


Figure 2.8 Metrics of Computational Offloading (Fernando *et al.*, 2013; Dinh *et al.*, 2013)

Moreover, mobile and surrogate devices have different types of CPU, speed, available memory, and storage capacity. If a mobile device does not have enough speed or storage memory, then offloading is a better option to utilize. For computational offloading, a few pre-identified parameters need to be considered first. Application type is a key metric which needs to be checked prior to offloading decision. If the application is processor-intensive or complex to compute locally, then offloading will be more useful. Meanwhile, other elementary metrics such as user QoS requirement, availability of local resources, SLA and network availability described in a sample flow of mobile application execution are shown in Figure 2.9.

The flow chart depicts four processes of mobile application to be executed. If a mobile device is capable of running the task, it will be executed locally; otherwise, it will be offloaded to the remote servers. If all the available options go false, then application execution request will be killed. The issues related to offloading are the efficiency and dynamism of offloading under changing environments (Dinh *et al.*, 2013). For example, the mobility or movement of the user of the mobile device will affect the bandwidth. This raises some concern about which strategy should be adopted to offload applications.

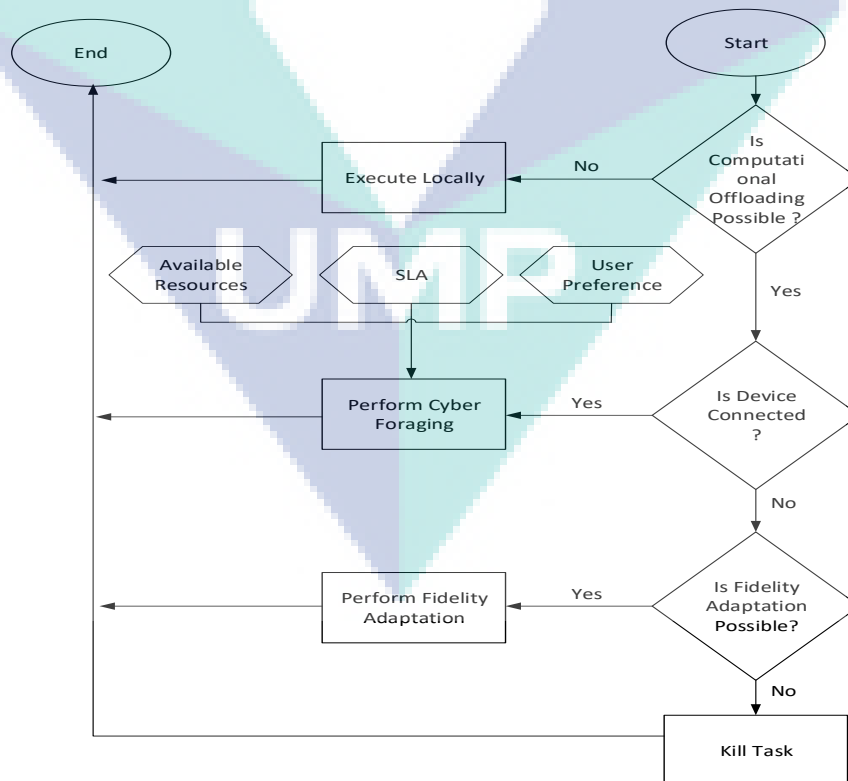


Figure 2.9 Execution Flow of Traditional Computational Offloading (Abolfazli *et al.*, 2012)

In the case of static offloading, the application will be partitioned for offloading at compile time regardless of environmental changes and user context. Redenko *et al.*, (1998) have noted that static offloading is not always an energy-efficient approach. If the size of intensive task is small enough, then offloading will consume more energy than the energy consumed in local processing due to the communication cost. For instance, if the size of intensive task is 500KB, then offloading the same task for remote execution will use about 5% of the battery of the mobile device while local execution of the same size of code will consume approximately 10% of the battery in computation. In this case, offloading can save a significant amount of energy (i.e., 50%). Conversely, if the size of codes is 250KB, then the efficiency reduces up to 30%. Thus, if the size of codes to be executed is small, the offloading will consume more battery than that of the local execution of the same task.

Computational offloading decision for mobile devices can be extremely tricky as it is not easy to decide whether or not to offload and which portions of the application's codes need to be offloaded in order to improve energy efficiency. Moreover, diverse wireless access technologies require different amount of energy and also support dissimilar data transfer rates. These factors need to be considered prior to offloading decision.

Hence, to overcome these issues, the dynamic offloading techniques are used. As suggested by Kumar and Lu (2010), these techniques will decide at runtime whether or not to offload and which portions of the application to be offloaded based on energy consumption. The optimal partitioning of program takes place on the basis of trade-off between computation cost and communication cost.

Additionally, several other solutions have been proposed for the optimal application partitioning. According to Messer *et al.*, (2002), if a device becomes resource-constrained at runtime and accepts that it can beneficially use nearby resources, it then automatically and transparently offloads part of the service to the nearby devices and configures the device to provide the services as a surrogate machine. Messer *et al.*, (2002) have therefore proposed a dynamic shared distributed environment, that is, in case no remote server becomes available, then the load can be shared with other surrogate servers.

2.6.3 Taxonomy of Cyber Foraging/ Computational Offloading

In this section, based on the available information of the existing computational offloading systems, a cyber-foraging or computational offloading taxonomy is presented. The most significant repetitive features such as offload type, surrogate type, offloading scale, solver locations, code availability, offloading granularity, data availability and parameter of decision of computational offloading systems are used to classify and discuss the taxonomy. Figure 2.10 illustrates cyber foraging taxonomy and is briefly discussed in the following subsections.

2.6.3.1 Offloading Types

Offloading can occur either at start time referred to as static offloading or at runtime called as dynamic offloading (Murarasu *et al.*, 2009). During static offloading, a middleware or programmer partitions the program before execution. Thus, at runtime, the system identifies which portions of the program should be offloaded. However, due to the expanded uniformity of network environments and the surrogates, static offloading cannot ensure the best partitioning for all probable situations which could be beneficial. Spectra (Flinn *et al.*, 2001; Flinn *et al.*, 2002) and Chroma (Balan *et al.*, 2003); Balan *et al.*, 2007) are the examples of most important works in which partitioning occurs before program execution.

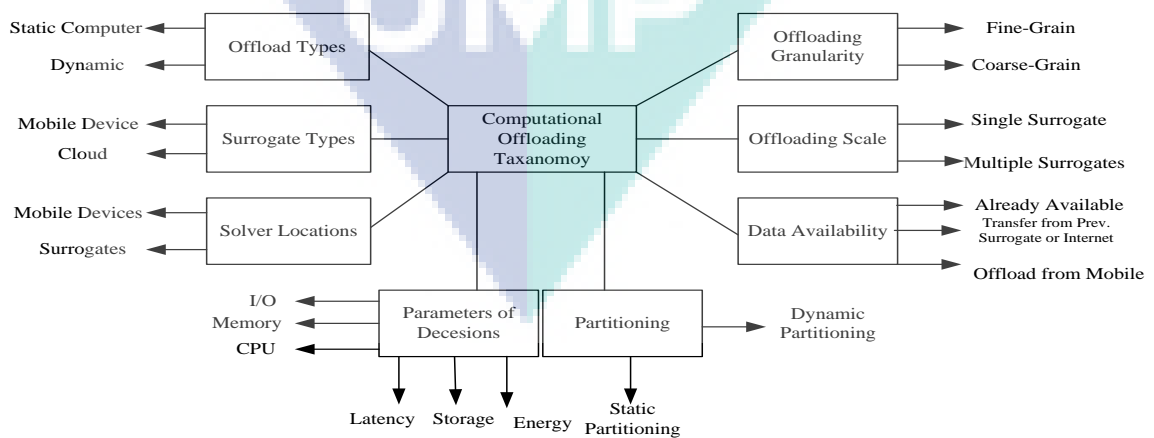


Figure 2.10 Taxonomy of Cyber Foraging (Adapted from Sharifi *et al.*, 2012)

By contrast, dynamic offloading starts to offload tasks when the required resources for offloading are insufficient. Dynamic offloading techniques were used by Gu *et al.*, (2004) in their research. This type of offloading partitions the program according to the availability of resources at runtime. This approach makes offloading decisions based on existing conditions and therefore, is beneficial and more flexible. Such approach however causes more overheads on the system.

2.6.3.2 Surrogate Types

Cyber foraging can be further categorized by the surrogate types. The surrogates can be either static computers or mobile devices. Generally, most of the cyber foraging approaches use static computers as surrogates (Su *et al.*, 2005; Satyanarayanan *et al.*, 2009) while others use mobile surrogates (Begum *et al.*, 2010). Although powerful stationary computers or surrogates are suitable for offloading, in case which no surrogates become available or in circumstances such as changing network topology, user preferences may direct a cyber-foraging system to choose a mobile surrogate for offloading instead.

2.6.3.3 Offloading Granularity

If the surrogate device does not have the required application, then there is a need to offload some of the related parts of the application from the mobile device to the surrogate. The process of offloading parts or the whole of the application is called offloading granularity. In the cyber foraging approach, if some parts of the application are offloaded, this is referred to as fine-grain. In their work, Flinn *et al.*, (2002) used this fine-grain method while other researchers such as Murarasu and Magedanz (2009) and Satyanarayanan *et al.*, (2009) offloaded the whole program which is referred to as coarse-grain. In fine-grain strategy only the parts which are needed can be offloaded and it leads to adequate energy saving. This strategy is suitable for a highly mobile environment, because mobile devices move in the environment and the probability of network disconnection increases due to load and unavailability of wireless signals.

2.6.3.4 Offloading Scale

The selection of surrogate devices to offload CPU intensive parts of applications is called offloading scale. Offloading scale differs in different cyber foraging or offloading approaches. The cyber foraging system either selects a single surrogate from the pool of available surrogates to offload a task and then obtains the result back (Flinn *et al.*, 2002), or in some other cases as noted by Kristensen *et al.*, (2008), the cyber foraging system uses multiple surrogates to offload a task. Offloading scale using multiple surrogates to offload a task is beneficial. This is because it deals with the mobility nature of mobile devices. Moreover, fault tolerance can also be increased by parallel offloading to multiple surrogates which will also facilitate the latency control (Zhang *et al.*, 2010).

2.6.3.5 Data Availability

To perform an execution of a task, some of the related information such as input data need to be available in the execution environment. The assumptions and tactics about data availability can be defined in three cases. The first case refers to a condition in which data are already available on the surrogate and there is no need to transfer anything from the mobile device to the surrogate (Kristensen, 2010b). For example, suppose two tasks A and B are running in mobile device and the A's output is the required B's input. If a surrogate has executed task A, then it has the B's input and it will not need data migration from mobile.

In the second case, information is missing with the surrogate and transfer is needed from a mobile device to surrogate (Balan *et al.*, 2003) whereas in the third case, the necessary information is fetched from another surrogate (Su *et al.*, 2005). This third case strategy can work more efficiently if the required data is fetched from the Internet. Additionally, the forecasting and context-aware information can help to improve this case to provide information prior to execution such as user's location, bandwidth or internet availability. It also foresees the next availability of internet in advance to transfer necessary information before starting to run the next task.

2.6.3.6 Parameter Decision

The main goal of cyber foraging or computational offloading is to cope with the resource constraints of mobile devices. Therefore, several matrices must be kept in mind before considering cyber foraging as a solution to augment a mobile device's local resources. The most essential factors that could be considered for offloading are energy consumption, memory storage, responsiveness and input/ output (I/O).

a. *Energy Consumption*

One of the key constraints of mobile devices is the limited power storage. As the mobile device's energy cannot be replenished by itself, this is the reason why many researches considered energy consumption or battery power as a parameter for taking an offloading decision (Flinn *et al.*, 2002).

b. *Memory Storage*

The applications which are memory intensive usually cannot be run on mobile devices and all such applications entails to offload. Consequently, many of the researchers considered the local memory and storage of mobile device as an effective parameter before offloading (Ou *et al.*, 2006).

c. *Responsiveness*

The execution time of a computational intensive application can be decreased by offloading if the processing power or the CPU speed of the mobile device is significantly lower than that of the static computers. Many researchers considered the response or execution time and latency as the main parameter which could affect the offloading decision.

d. *Input/ Output (I/O)*

Sometimes, input / output (I/O) devices are considered for the improvement of quality. For example, when there is a need to play a movie on a larger screen, use bigger

speaker for playing music or use distant printing. Some previous works focused on augmenting the I/O as an effective parameter in offloading decision.

2.6.3.7 Solver Location

The unit, which is responsible for taking offloading decision, is called a solver. This parameter is considered by many researchers as a location of solver. Normally, every mobile device has its own solver and can play the role of decision maker itself. However, in some works such as Cuervo *et al.*, (2010), the solver was not located in the mobile device. For example, MAUI generates a call graph of application to execute. The call graph may possible obtained the accurate partition to execute while it may sometime miscalculate the partitioning and offloading decision due to insufficient pre-defined parameters. It is therefore significant to have a solver which may take a precise decision for offloading the remote executable tasks.

2.6.3.8 Application Partitioning

As the computational complexity of application processing as well as the resources of computational network increases, logic dictates to distribute a centralized programme into components and execute each component parallel in order to reduce load, share resources and make the processing efficient. The term application partitioning refers to breaking down the application into components in distributed application frameworks while the components preserve the semantic of applications. Current Distributed Applications Frameworks comprise dividing runtime applications in two different ways, either static partitioning or dynamic partitioning.

a. Static Partitioning

The concept of computational offloading has been introduced with static offloading where the application used is partitioned once in compile time or runtime and the static parts of the application are then offloaded to the remote server. In their work, Satyanarayanan *et al.*, (2009) primarily partitioned applications into two parts, namely, the user interface part which stayed in the mobile device, and the resource-intensive or compute-intensive parts delegated to the remote servers. Meanwhile, in their research,

Dou *et al.*, (2010) developed Misco in which the application was statically partitioned into two parts, namely, the *Map* and the *Keys*. The Map function was then applied to the set of input data which produced the intermediary <" Key, values"> pairs. All such pairs were then grouped into a number of partitions. Whole pairs in a single partition were then passed to the reduce function which then produced the final results.

b. Dynamic Partitioning

In the dynamic partitioning approach, the algorithms used to dynamically partition an application continuously monitor the available resources for SIDs. During the processing of the application at runtime, the resources are allocated to each component for processing. The concept of dynamic partitioning is developed in contrast to static partitioning where resources are allocated to components once in compile time or runtime as opposed to the dynamic approach in which resources are allocated to each task in a sophisticated way. Current dynamic computational offloading frameworks are used to exercise the dynamic application partitioning approach. The works of Goyal *et al.*, (2004), Chun *et al.*, (2009) and Zhao *et al.*, (2010) Yang *et al.*, (2013) are examples of the current distributed application computational offloading frameworks which used dynamic application partitioning approach in their studies.

2.7 Related Works

In order to achieve energy efficiency during application processing, the whole focus of researchers would be to execute the intensive tasks as quickly as possible and then allow the platform to go into sleep state. For this purpose, a multithreading concept had been used in the past to allow the applications be executed in multiple parts using multiple cores of the processor concurrently. As soon as the execution finishes, the free cores in the processors go into idle mode and then into sleep mode (Metri *et al.*, 2014). After the era of mobile cloud computing has started, the concept of computational offloading becomes familiar with vigour. The computational load has migrated to the remote servers for processing in order to free the mobile device's processor from the local workload.

2.7.1 Previous Research on Enhancing Mobile Efficiency

For the last few decades, numerous researcher has attempted different techniques to delegate the resource-intensive parts of the applications to remote servers in order to minimize the load of the local resources. In this regard, two approaches are commonly used (Cuervo *et al.*, 2010).

The first approach relies on programmers specifying how to partition a programme, and which parts of a programme need to be remote and how to adjust the programme partitioning scheme with the frequently changing network environment (Balan *et al.*, 2002; Balan *et al.*, 2007). This approach leads to saving adequate energy because it is fine-grained. The application can then be offloaded in sub-parts only if the remote execution is beneficial in terms of energy, processing and storage.

The second approach involves the migration of the entire process (Balan *et al.*, 2002) or OS (Virtual Machines) (Chun *et al.*, 2010) to the cloud instead of to the sub-parts. This approach excludes burden on programmers for instance application does not need to be partitioned and the entire process or system is automatically loaded to the remote servers. A review of some typical research projects is presented next in this section.

This review begins with a discussion of CloneCloud. It was introduced in 2011 by Chun *et al.*, (2010). The Clone is an image of a mobile device residing on a virtual machine in cloud. In contrast to the smartphone, a Clone is in rich hardware, networks and software environment close to energy-efficient resources. This condition is more suitable for the execution of complicated task. The main method used is virtual machine migration which offloads the application's execution blocks from resource-constrained mobile devices to rich resources pool Cloud flawlessly and partly. The CloneCloud system either fully or partly offloads the smartphone based execution to a dispersed environment. The CloneCloud architectural framework is shown in Figure 2.11 (A). Each smartphone's task is divided into five different execution blocks. The blocks are divided on resource-intensive basis. The blocks which are more power hungry are then passed to the cloud for processing. The energy-intensive blocks appear in coloured green in the diagram as shown in Figure 2.11 (B).

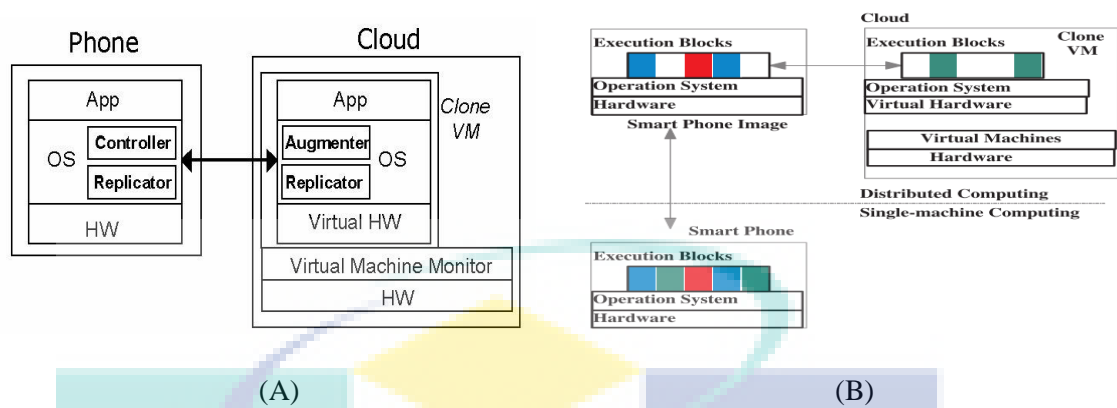


Figure 2.11 CloneCloud–Architectural and System Framework (Chun *et al.*, 2010)

Once the execution of these blocks are completed, the output is then passed from CloneCloud to the smartphone. A face tracking application is taken as a test by Chun *et al.*, (2011) with and without CloneCloud and the result showed that CloneCloud had taken 1 second to process the task while the smartphone had taken 100 seconds to process the same. Another advantage of CloneCloud System is the reservation of battery life, as Smartphones do not need to process complicated tasks.

Nevertheless, this approach has its disadvantages such as the handover delay and bandwidth limitation. Because the speed of data transmission is known to be inconsistent, the CloneCloud System will therefore not be responsive whenever the user moves to a signal blind area. Furthermore, the data that stream from the mobile device to the distant server is also not consistent; hence, data stream needs to be optimized for speedy process of flow and thus reducing RTT (Yang *et al.*, 2013). On the basis of CloneCloud, Zhang *et al.*, (2011) have introduced an elastic application framework to enhance the performance of resource-constrained devices by dynamic execution configuration of application according to the device current status. This framework divides an application into a range of multiple components called weblets. It offers a dynamic adaptation nature of weblet execution configuration and a cost model is provided to adjust the execution pattern; however, this framework needs a mechanism for exchanging of weblets between the mobile devices, as the communication channel of the mobile devices may be changing (e.g., from 3G to GPRS or Wi-Fi). Another challenge is a media channel or high speed bandwidth is needed to ensure that the communication between weblets is reliable.

Although both of the approaches are energy-efficient for mobile devices, the response time for data transmission between cloud and mobile devices is slow, especially when the bandwidth is low. Thus, for light weight applications which can be deployed locally in a smartphone, it cannot be justified to offload all the applications to the cloud. Lu *et al.*, (2011) have made the presentation of Virtualized Screen in the cloud possible. In this approach, the screen rendering moves from the mobile device to the cloud as a service and is brought as an image to the client device for interactive display. They enable thin-client devices to enjoy various compute-intensive and graphically-rich services in the cloud.

Furthermore, screen virtualization does not mean offloading the whole rendering task to the cloud but to make offloading decision on the basis of matrices such as local device resources efficiency, network condition, traffic condition, response time, screen resolution. In this regard, part of the smartphone's screen is virtualized in the cloud which contains a collection of data using display image, audio, video, key board input, and text-contents. The light weight part of an application is deployed locally to process which effectively diminish the power consumption. As such, Lu *et al.*, (2011) have suggested that the framework for screen rendering should be partially done in the cloud and partially in the mobile device. Consequently, they sought to resolve network bandwidth obstacles and curtail energy consumption. The challenge in remote screen rendering is that the real-time and high-fidelity processing of the remote execution of the screen is affected by low bandwidth. The low bandwidth and the offloading to a distant cloud server for execution impede the optimal performance of mobile applications (Ahmed *et al.*, 2015).

In order to solve the issue of bandwidth delay between the mobile device and the cloud, Satyanarayanan *et al.*, (2009) have presented the concept of Cloudlet which is a *Micro Cloud* configured in the middle of the mobile device and the Cloud. The author argued that even though Cloud Computing is the finest solution for resource-constraint devices, the long WAN latency impedes its performance. The rapid changing of the computing environment causes changes in the bandwidth access between the mobile device and the cloud. This leads to different kinds of delay, especially when mass data need to be transferred and processed. The occurrence of such delays will then be experienced by the user. Unfortunately, the bandwidth delay is totally unavoidable because of firewall filtering or data checking which are inevitable for security. To

overcome the problem, virtual machine (VM) technology is used to provide instantaneous customized services. Figure 2.12 shows a Cloudlet, which is a resource-rich computer or cluster of computers, installed in a coffee shop to provide rapid customized services to the client devices using VM technology through a high bandwidth to mobile users.

Compared to the distant Cloud, Cloudlet exists in a single hop distance, which provides the fastest processing and transmission bandwidth to the connected devices. In case where no Cloudlet exists in the surrounding, the mobile devices will then access the resources of a distant cloud or in the worst case scenario, the mobile devices will use their own local resources to handle the execution of applications. The main challenge in this approach is the compatibility issue related to applications running in Smartphones which are rapidly improving. This is because the VM based Cloudlet might not possess such a wide range of compatible applications.

A slightly similar approach has been introduced by Canepa *et al.*, (2010), namely, the *ad-hoc mobile cloud framework*, which is a virtual cloud computing platform. Canepa *et al.*, (2010) have discussed communities which are built of mobile devices where the mobile devices become capable of executing shared tasks.

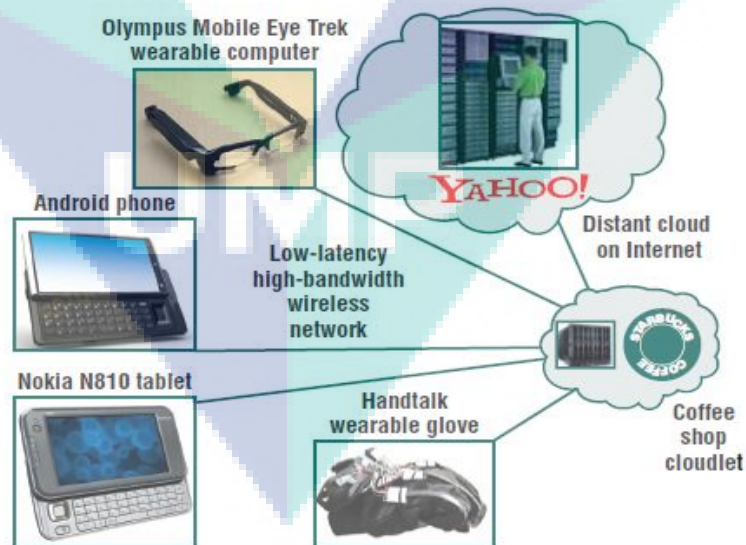


Figure 2.12 Cloudlet Architectural Framework (Satyanarayanan *et al.*, 2009)

The framework proposed by Canepa *et al.*, (2010) allows a small portion of the task to be executed locally while the rest is delegated to the nearest mobile device available in the same vicinity already running the same task. The concept of ad-hoc mobile cloud framework presented by Canepa *et al.*, (2010) consists of five basic components such as application manager, resource manager, context manager, P2P component, and offloading manager. Each of these five basic components will be described briefly.

First and foremost, the application manager, is responsible for the starting and the stopping of an application at loading time. It also modifies the application to take in features according to the current context needed for offloading such as RPC support and proxy creating.

Secondly, the resource manager is in-charge of the application profiling and monitoring of resource in a local device. For each application to execute, a profile is created to keep record of all of the remote devices which are needed to build a virtual cloud. The application profile is then checked by the application manager every time an application is executed in order to determine whether or not an instance of the required virtual provider needed to be created.

The third basic component is the context manager. It is responsible for the synchronisation of contextual information getting from context widgets and makes them available for other process. Context manager consists of three sub-components: 1) Context widget, which is responsible for handling communication with the sources of context information; 2) Context manager, which receives new context from the available information; and, 3) Social manager, which keeps record of several types of relationship between users.

The fourth and the fifth basic components are the P2P component and the offloading manager. P2P component is responsible for informing the context manager of joining a new device in the vicinity or leaving away status of an old device. Meanwhile, the offloading manager handles the offloading task to the neighbouring device for execution. It also accepts tasks from the other remote devices and process the tasks accordingly. This approach would certainly save energy; however, the pervasive nature

of nodes needs to have an adopting access mechanism from the neighbouring device. Hence, a mechanism for dealing with the energy consumed in extra computation for making decisions is needed.

The biggest challenge to mobile devices is distributed computing. In such computing, new class of applications are needed to react to the rapidly occurring changes. Schilit (1994) introduced the term context-awareness in distributed computing (i.e., ubiquitous computing). Applications should be aware of the environment they are running in and adapt to the changes according to the context. The mobile devices can manage their resources in a better way when devices are aware of the contextual computing in the pervasive environment. For example, GPS is used in Smartphone devices to detect locations, but it drains the battery of the mobile devices more than other components. The context aware approach will keep the GPS usage in schedule to trigger whenever it is needed or otherwise the GPS will be turned off.

Zhuang *et al.*, (2010) had first developed a framework for location sensing based on the contextual information which is energy-efficient as compared to GPS. Kim *et al.*, (2011) then developed Wi-Fi Sense system to sense the environment using low power sensors and previous recorded data in order to predict the best available network interface for communication, and to turn on Wi-Fi interface on demand fashion to save battery life.

Meanwhile, Herrmann *et al.*, (2012) have proposed a system, namely, the Dynamic Power Management (DPM), to avoid the unnecessary sensing of distributed sensing application. This system uses the context knowledge to adapt to the behaviour of applications. According to the current user's context, the system starts, suspends and changes the sampling rate of application used for collecting sample in a sensor network.

In addition, context-aware battery management architecture for mobile devices (CABMAN) has been proposed (Ravi *et al.*, 2008). On the basis of user's current context, if the system detects a charging opportunity, it will then warn the user that the device's battery may be running out of power. The system works on the proposed algorithms for processing user's location and call-logs for making some of the predictions.

By using the embedded sensors of the mobile device, Moghimi *et al.*, (2012) have presented a middleware context-aware power management system. Fuzzy inference is used in this system for extracting the high level context from the low level context, which provides near to accurate results of the user context. The system proposed by Moghimi *et al.*, (2012) consists of the sensors, the context detection block (CDB) and the power manager as shown in Figure 2.13.

The system receives raw sensing data from the embedded sensors. CDB works as an inference unit, which extracts the high level context from the low level sensing context. This eliminates the possibility of an application retrieving the same context for the second time. The power manager is placed in between the CDB and the applications. It receives the sensing variables registered by the respective applications and tune them by some defined rules to deliver context-aware energy-efficient performance. This results in 10-50% lower power consumption of the system. The challenges of this system are to expand the context variables and adopt a dynamic way of distinguishing the high level context from the low level context.

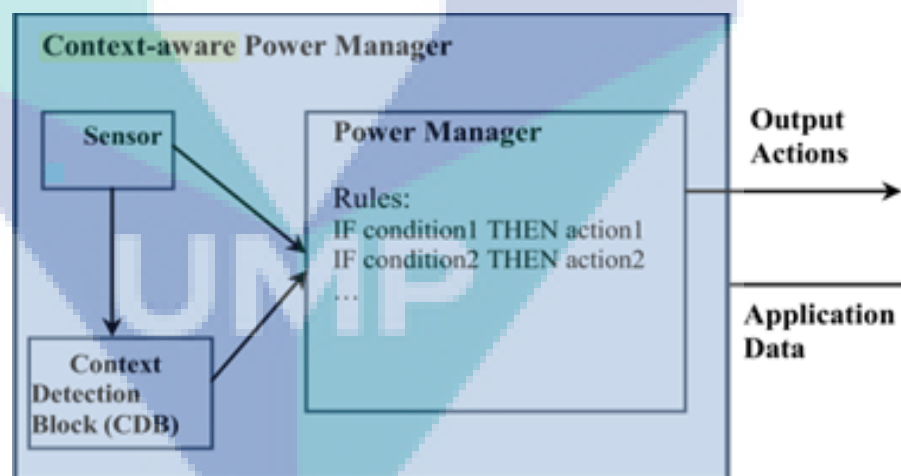


Figure 2.13 Context-Aware Power Manager

To make the user interaction limited and build a smart environment, Sathan *et al.*, (2009) have proposed a context-aware lightweight energy-efficient framework (CALEEF). CALEEF consists of seven components as shown in Figure 2.14. Using this approach, the smartphone needs to be intelligent enough to decide when to access or to execute the application on the basis of high level contextual information. For example, if the user is in a meeting room, the context-aware mobile device will sense the environment and reject all unimportant calls.

First Component: Context Acquisition Context acquisition acts as a mediator between an application and its operating environment. At data acquisition layer, specific widgets are developed in order to capture different kinds of information. This layer releases the applications from the issues relating to context sensing by tying the sensor with a single interface. This way it makes independent application design for the method of context sensing. The context widget will continuously update the context encoder with context information. The context information is then sent to the context service provider for the storage and dissemination of the context to the consumers.

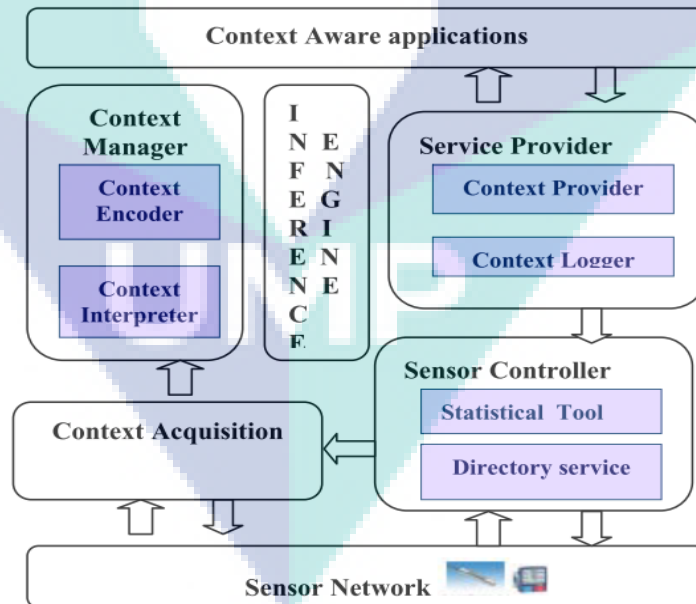


Figure 2.14 CALEEF Architectural Framework (Sathan *et al.*, 2009)

Second Component: Context Manager Context manager is responsible for the conversion of context data received from sensors to the context information that will be provided to application for further action. This component is further divided into two sub-components, namely, the context interpreter and the context encoder. The context interpreter does context processing by logical reasoning; as a result, the high level context is derived from the low level context. It also resolves context conflicts. The context encoder, as the name suggests, encodes the context information using Ontology Web Language (OWL) and then passes it to the context logger for record.

Third Component: Interface Engine The interface engine performs reasoning on stowed facts. Using the past and current context information, it defines how an application should change its behaviour accordingly.

Fourth Component: Context Logger The context-aware applications may change their behaviour using the past context along with the current context. For this reason, the previous context is encoded and stored in the context history that may be queried by the applications later whenever needed. The context logger is made up of the context knowledge base and the context history. The context knowledge base provides a set of API's for the components of other services to query, modify, add, delete the context information.

Fifth Component: Context Provider The context provider is responsible for keeping record of the context consumers. This component will always trigger the record it keeps whenever new context information is obtained.

Sixth Component: Directory Services The directory services register the sensors' information of the surrounding and keep record of the sensors' attributes such as refresh rate, spatial information, and correctness. By using this mechanism, CALEEF selects the sensor that is most suitable for receiving context data.

Seventh Component: Context Consumers Context consumers consumes different kinds of context information and adapt to their behaviour accordingly. It is done by either listening to the context provider for new context information or querying it to receive updates. The main feature of CALEEF is context reasoning. High level implicit context can be derived

from low level explicit context. Application confidently uses the high level context information to change their behaviour.

The challenging issue in CALEEF framework is, in case there is a failure in any sensor or component, the system needs to restart and restore itself to the last working state. Some researchers have utilized cloud resources using the contextual approach to minimize local resources operation. For example, Xiao *et al.*, (2011) proposed a framework CasCap (Cloud-assisted Context-aware Power Management). In their work, the cloud resources for processing, storage, and networking were utilized to provide an efficient and low cost power management of mobile devices. CasCap consists of three main components, namely, mobile devices, internet services and clones. Mobile devices in CasCap have the following five components: resource manager, context manager, scheduler, policy manager and communicator.

The first component, namely, resource manager runs in the background and is responsible for monitoring the device's resource consumption. It also collects the contextual data from sensors such as the GPS and the accelerometer. The second component is the context manager. It generates the contextual information on the basis of the data collected by the resource manager from the sensors and uploads the context information to the cloud. A crowd-context monitors service in the cloud which receives the contextual information from context manager of the mobile devices and other network elements and then queries on them to get meaningful context information.

The third component is the scheduler. This component is responsible for keeping track of the changes in the context and then adapts to the mobile devices according to the changes. Meanwhile, the fourth component is the policy manager which stores all the policies that are made to govern the process of the CasCap framework. These policies are specific rules and actions that should be taken by the device itself or by the internet Cloud whenever a specific change occurs in the context information. Finally, the communicator is the component which is responsible for providing the wireless communication facility between the mobile device and the cloud for sharing the policies and the context information and for using internet services.

The findings of a study conducted by Xio *et al.*, (2011) are significant in terms of the main features of CasCap such as the monitoring of crowd-sources context, functional offloading to the cloud and the adaptation as services. This research first time considered the third party services to be part of the development and the deployment of power management services. The system still needs to resolve the challenges in the migration of radio stream from one proxy to another whenever the user moves. A filter process is also needed for context information stored in the cloud. This is because after some time, the stored context information might become invalid.

All of the discussed approaches have resulted in different solutions at different levels to minimize the power consumption of handheld devices. The two most popular solutions are, either by conserving battery life through resource management or by offloading (i.e., migrating load to cloud servers). Singular solution by simply adopting job migration to the cloud or only through resource management alone cannot provide an adequate power-saving solution as both approaches have their own limitations. The different approaches reviewed are summarised in Table 2.2.

The logo for UIMP (University of Malaya Power Management) is a large, downward-pointing arrow shape. It is composed of four quadrants: top-left is light blue, top-right is light purple, bottom-left is light purple, and bottom-right is light blue. The letters 'UIMP' are written in white, bold, sans-serif font across the center of the arrow.

UIMP

Table 2.2 Analysis of the Previous Research Work on Resources Augmentation of Mobile Devices

Focus / Purpose	Framework	Method	Outcomes / Results	Limitations / Future Work
Reduce Mobile Resources Labour	CloneCloud Chun <i>et al.</i> (2010)	Offloading	21.2x speedup smartphone device application processing Protect battery life by remote application execution	Handover delay and bandwidth limitation
Dynamic application execution	Weblet Zhang <i>et al.</i> (2011)	Elastic application configuration	Performance enhanced by dynamic adaptation nature of complicated tasks. Quick and dynamic access of application reduces the local resources operation and save battery life	Mechanism need for exchanging weblets between devices, with changing communication channels (3G to GPRS or Wi-Fi)
Reduce Mobile Resources labour	Interactive Screen Remote system Lu <i>et al.</i> (2011)	Screen Virtualization in Cloud	Thin-client devices to enjoy various compute-intensive and graphically rich services in cloud. Reduces local resources operations & conserve battery life	For real time and high fidelity processing the remote execution of screen might affected by low bandwidth
Solve the issue of bandwidth delay b/w mobile device and cloud	Cloudlet Balan <i>et al.</i> (2001)	Offloading (Cyber Foraging)	Cloudlet exist in a single hop distance, provide the fastest processing and transmission bandwidth to the connected devices Provides the rapid customized services to the client devices by using VM technology through a high bandwidth.	Applications compatibility issues
Reduce Mobile Resources labour	Virtual Cloud Computing Platform Hureta-Canepa <i>et al.</i> (2010)	Offloading (Remote Execution)	The pervasiveness of mobile devices, creating a cloud among the devices in the vicinity, allowing them to execute jobs between the devices.	The pervasive nature of nodes needs to have an adopting access mechanism from neighbour. Also a mechanism for dealing the energy consumed in extra computation for making decisions.
Resource Management	PARCTAB System Ali <i>et al.</i> (2015)	Context-aware Computing	Developed unique set of context-aware application which enhances the operation of applications by communication and context information. PARCTAB depends on small cell wireless communication, thus combines portability with information about context.	PARCTAB system has very limited use when disconnected from a network.
Resource Management	Location Sensing Framework Zhuang <i>et al.</i> (2010)	Context-aware Computing	Reduce GPS usage up to 95 % while increase battery life up to 75 %	As compare to GPS, the proposed system cannot provide accurate location sensing in some cases.

Table 3.2 Continued

Resource Management	WiFisense System Kim <i>et al.</i> (2011)	Context-aware Computing	Increases Wi-Fi usage for various scenarios. Save energy consumption for scanning by up to 79 % while reduces false triggering by up to 4.3 %	The accelerometer is unable to provide the accurate movement information without any location base sensor.
Resource Management	Context-aware DPM Herrmann <i>et al.</i> (2012)	Context-aware Computing	The tested technique on a real system shows that it can extend smartphone battery life by 5x.	The context detection using sensor is an extra cost that the system has to pay to gain the context knowledge.
Resource Management	CABMAN Ravi <i>et al.</i> (2008b)	Context-aware Computing	Predict next charging opportunity on the basis of developed prediction algorithm. Accurate battery life prediction based on a discharge speedup factor. Save battery life for crucial applications for instance Telephony	For those users who spent a very high entropy routine, the prediction algorithms may not work very well. As many users charge phones in their cars while driving, in such condition it is not always possible to consider location prediction for charging availability.
Resource Management	Context Aware Power Manager Moghimi <i>et al.</i> (2012)	Context-aware Computing	Fuzzy inference used to provide high level context. The results show reduction in energy 13-50% for periodic applications, and for streaming applications 18- 36%.	Need to expand the context variables and adopt a dynamic way of determining high level context from low level context.
Resource Management	CALEEF Sathan <i>et al.</i> (2009)	Context-aware Computing	Reduces the cost and complications of developing context-aware applications by a shared context model of distributed software components. It also get context from a widespread range of sources rather than sensors only that are rooted in the local environment. It enables knowledge sharing among applications entities.	The need of autonomic service-oriented Computing ideas for developing context-aware service frameworks. In case of sensor or any component failure need the system to restart and restore itself to the last working state.
Resource Management + Offloading	CasCap Xiao <i>et al.</i> (2011)	Context-aware Computing	CasCap comprise of crowd-sourced context monitoring, function offloading, and adaptation as service. For the third party service providers the frame work provide a fresh way to develop and deploy power management services.	The system still needs to resolve the challenges in migration of radio stream from one proxy to another whenever the user moves. Need a filter process for context information stored in the cloud as after some time the stored context information might become invalid.

Chun *et al.*, (2010) have used the concept of cloning mobile device in distant cloud to process computational intensive tasks migrated away from mobile device; however, the mobility features and low or interrupting bandwidth issues sometimes cause delay in the services. Moreover, synchronization with the clone device each time increases Round Trip Time (RTT).

Meanwhile, Lu *et al.*, (2011) have presented a concept of the screen rendering instead of migrating the whole task to the distant cloud. Lu *et al.*, (2011) have introduced Virtualized Screen in the cloud to overcome the bandwidth delay issues. In this approach, the screen rendering moves from the mobile device to the cloud as a service and is brought as an image to the client device for interactive display. In this approach, part of the smartphone's screen is virtualized in the cloud which contains collection of data using display image, audio, video, key board input, and text-contents.

Similarly, Balan *et al.*, (2002) had earlier presented the concept of Cloudlet, which is a *Micro Cloud* configured in the middle of mobile device and the Cloud. They argued that even though Cloud Computing would be the finest solution to overcome limitation of resource constraint devices, the long WAN latency may impede its performance. The rapidly changing computing environment would alter the bandwidth access between the mobile device and the cloud thus leading to different kinds of delay, especially when mass data would need to be transferred and processed. The presence of such delays would be detected by users. This approach has shown that a Cloudlet, which is a resource-rich computer or cluster of computers installed, provides the rapid customized services to the client devices by using Virtual Management (VM) technology through a high bandwidth. In comparison to the distant cloud, the Cloudlet, which is situated in the nearest distance, provides the fastest processing and transmission bandwidth to the connected devices. In case there is no Cloudlet that exists in the surrounding, the mobile devices will then access the resources of the distant cloud or in the worst case scenario, the mobile devices will use their own local resources to handle the execution of applications. The main challenge in this approach is the compatibility issue related to applications running in Smartphones which are rapidly improving. This is because the VM base Cloudlet might not possess such an immense range of compatible applications.

Likewise, the approaches such as those adopted by Balan *et al.*, (2002), Chun *et al.*, (2010), and Lu *et al.*, (2011) encompasses the issues of low bandwidth and excessive offloading, while in particular circumstances the mobile device can locally process the task easily. For this reason, to make the user interaction limited and build a smart technique, Sathan *et al.*, (2009) have proposed a context-aware lightweight energy-efficient framework (CALEEF). The limitation of this approach is the deployment and then the failure of the sensor which causes disconnection of the whole service. In case of sudden failure of the system, an automatic mechanism needs to restate the system to the previous state. Past researchers have used the concept of context-awareness to make a precise decision at the time of offloading remote executable parts of different applications (Ravi *et al.*, 2008a, Kim *et al.*, 2011, Xiao *et al.*, 2011, Herrmann *et al.*, 2012, Moghimi *et al.*, 2012). The major issues in context-aware approaches are the use of extra sensors and filtration of high level context information from low level sensed context input.

2.8 Review of Computational Offloading Frameworks

By nature of the computational offloading, its frameworks are mainly divided into several main categories. These frameworks will be explained in this section.

2.8.1 Whole Application Migration Frameworks

In this approach of computational offloading frameworks, the entire application is offloaded to remote servers for processing. The Application Migration offloading Frameworks are used to exclude the partitioning and granularity overhead at mobile device; however, offloading the entire application is sometimes communication- or bandwidth- intensive due to the limited available bandwidth. Furthermore, delegating components of the application to resourceful servers which are, by contrast, lightweight and easy to process locally, causes delay and consequently drains the battery's power. A few past research works that were based on the whole application migrations are a study conducted by Chun *et al.*, (2004) and also another study conducted by Chun and Maniatis (2010).

2.8.2 Virtual Machine (VM) Level Migration Frameworks

Virtual machine (VM) level migration frameworks fall into the second category of computational offloading frameworks where a VM instance of each SID is required to be created on the server which serves as a shared infrastructure. The advantage of VM migration frameworks is that a single surrogate can run a number of VM instances with a complete isolation and security; therefore, these frameworks are less vulnerable to security concerns. The adverse side of these frameworks is the deployment of template based virtualized approach which is resource-intensive and highly time-consuming for VM deployment. This approach is used by many researchers such as Goyal *et al.*, (2004), Satyanarayanan *et al.*, (2009), Chun *et al.*, (2010) and Wang *et al.*, (2011).

2.8.3 Method Level Migration Frameworks

In method level migration, the computational intensive methods of a running application are marked as heavy methods and lightweight methods. Different terms have been used in research for both heavy and lightweight. Some research works have referred to heavy methods as intensive methods or computational intensive methods (Ewens *et al.*, 2001). These methods involve heavy computations required to offload for execution.

By contrast, lightweight methods are methods which do not involve heavy computations and can be processed locally. The term lightweight is used by many researchers in their studies such as Shiraz *et al.*, (2013), Shiraz *et al.*, (2014) and Shuja *et al.*, (2015). To partition the application into lightweight and heavy methods, the heavy methods are symbolized as computational-intensive using specific keywords. The whole burden of partitioning the application in local and remote methods is placed on the developer.

The concept of method level migration was introduced after the migration at class level. After the concept of class level migration which had been used by Gu *et al.*, (2004), Yang and Liotta (2006) developed a concept in which the whole class was identified as intensive and migrated to the remote servers. This concept would be secure and easy to implement. However, the only disadvantage would be a class may contain many methods

including methods which would be easy to process locally and therefore would be illogical to offload such methods to the non-computational-intensive remote servers.

By contrast, at method level, the computational-intensive methods are offloaded only during which the rest can be executed locally. As a result, it eliminates the process of lightweight methods having to wait for execution while remote execution is in process. Meanwhile, Rim *et al.*, (2006) and Cuervo *et al.*, (2010) have used the method level migration. In their research, Rim *et al.*, (2006) employed the technique to reduce the code size in the mobile device by transformation method and used Distributed Execution Transformer (*DiET*) to generate slim codes for heavy methods. The mobile device would download the modified bytecodes and execute the application computation at the server.

The traditional offloading approach refers to the existing and recent research works. In the literature, the term traditional offloading has been used in many research works. For instance, Wu *et al.*, (2013) and Shiraz *et al.*, (2014) have used the same term. In the present research, the term traditional offloading has been used with the intention of simulating the traditional offloading prototype which would be compared based on efficiency with the proposed model. Thus, traditional offloading frameworks from 2004 until 2015 are critically examined in this literature review in relation to power efficiency of mobile devices. Table 2.3 summarises the traditional computational offloading frameworks on the basis of computational offloading nature, type of offloading, granularity, application partitioning and offloading scales.

Table 2.3 Comparative Review of Computational Offloading Frameworks

Framework	Application Partitioning	Migration Level	Offloading Type	Remote Server Type	Offloading Scale	Offloading Granularity	Focus
Offloading Inference Engine (Gu <i>et al.</i> , 2004)	Dynamic	Class level	Static	Surrogate	Single	Fine-grained	Memory Management
Roam System (Chu <i>et al.</i> , 2004)	Dynamic	Application Level	Dynamic	Cloud Server	n/a	Coarse-grained	System for Heterogeneous devices
DiET (Rim <i>et al.</i> , 2006)	Static	Method Level	Static Computer	Cloud Server	Multiple	Fine-grained	Save Battery
Offloading Toolkit (Ou <i>et al.</i> , 2006)	Static	Class Level	Static	Surrogate	Single	Fine-grained	Reduce Complexity of Partitioning
IDP (Xian <i>et al.</i> , 2007)	Static	Task Level	Static	Surrogate	Single	N/a	Save Battery
mPlatForm (Gorackzko <i>et al.</i> , 2008)	Dynamic	Task Level	n/a	n/a	Multiple	Fine-grained	Energy Saving
n/a (Huerta <i>et al.</i> , 2008)	Dynamic	Tasks level	n/a	n/a	Single	Fine-grained	Reduce Execution time
WishBones (Newton <i>et al.</i> , 2009)	Static	n/a	Dynamic	n/a	Multiple	Fine-grained	High-rate Data processing
CloudLet (Satyanarayanan <i>et al.</i> , 2009)	n/a	VM level	Dynamic	Surrogate	single	Coarse-Grained	Reduce Complexity
Mobile Service Cell (Liu <i>et al.</i> , 2009)	n/a	n/a	Dynamic	Surrogate	Multiple	n/a	Reduce Complexity
Scavenger (Kristensen, 2010)	Static	n/a	Dynamic	Surrogates	Multiple	Fine-grained	Energy saving & Augmenting CPU

Table 2.3 Continued

Framework	Application Partitioning	Migration Level	Offloading Type	Remote Server Type	Offloading Scale	Offloading Granularity	Focus
Cuckoo (Kemp <i>et al.</i> , 2010)	Dynamic	Method Level	Dynamic	Cloud Server	Single	Fine-grained	Reduce Energy Consumption
MAUI (Cuervo <i>et al.</i> , 2010)	Dynamic	Method Level	Static	Surrogate	Single	Fine-grained	Energy Saving
N/a (Chun <i>et al.</i> , 2010)	Dynamic	Application Level	Dynamic	Cloud Server	n/a	Coarse-Grained	App. Partition Problem
CloneCloud (Chun <i>et al.</i> , 2011)	Dynamic	Thread Level	Dynamic	Cloud Server	Single	Coarse-grained	Saving Energy
SociableSense (Rachuri <i>et al.</i> , 2011)	Static	Task Level	Static	Cloud Server	Single	Fine-Grained	Social Behavior
ThinkAir (Kosta <i>et al.</i> , 2012)	Static	Method Level	Dynamic	Cloud Server	Multiple	n/a	Energy and execution time reduction
DCOF (Shiraz <i>et al.</i> , 2014)	Dynamic	Method Level	Dynamic	Cloud Server	n/a	Fine-grained	Energy and offloading cost reduction
EECOF (Shiraz <i>et al.</i> , 2015)	Dynamic	Task Level	Dynamic	Cloud Server	Multiple	n/a	Reduce complexity and energy
Code Offloading (Flores <i>et al.</i> , 2015)	Dynamic	Code Level	Dynamic	n/a	n/a	Fine-grained	Energy Saving
MCC Offloading (Shuja <i>et al.</i> , 2016)	Static	Process State Migration Level	Dynamic	Cloud Server	n/a	n/a	Reduce Overhead

The study conducted by Goyal and Carter (2004) is based on dynamic offloading to the static computers (i.e., the surrogates) in close proximity, utilized the computing resources nearby. It is a simple client server environment where a mobile device requests to utilize the server's resources for processing and the surrogate computers provide the resources on demand. Goyal's framework supports VM migration for remote processing and falls in the category of virtualization approaches. The framework is good for privacy and security due to local availability. It also involves low latency due to limited hop proximity. Conversely, the deployment template of virtual machine each time with each offload is resource-intensive and time-consuming (Wang *et al.*, 2011).

To overcome the issues of virtual machine deployment, VMbase cloudlet framework had been proposed by Satyanarayanan *et al.*, (2009). In this framework, instead of deploying the template of virtual machine, the image of running applications would be migrated to the Cloudlet, which is a local server or cluster of servers at single hop proximity. In the proposed work, the mobile device acted as a thin client, utilizing the server's resources through user interface. The actual processing of application takes place at the remote server. For customization of services, extra hardware resources are involved in implementing the framework. The cloning of mobile application each time with a fluctuated bandwidth is resource-intensive as well as time-consuming. It also leads to issues of security and privacy.

A more recent approach of Chun *et al.*, (2010) proposed the migration of clone virtual machine image for cloud-augmented execution. The approach adopted by Chun *et al.*, (2010) differed from those adopted by Goyal & Carter (2004) and Satyanarayanan *et al.*, (2009), that is, proposing three different types of offloading algorithms for different types of applications. Chun *et al.*, (2010) works resembles to Cloudlet in terms of migrating the virtual machine image. A simple synchronizing approach is used to reduce the application's dynamic transmission overhead. Clone cloud approach implements a simple partitioning method of executing applications in two main parts. The user interface which is less computational-intensive processed locally while the heavy tasks are offloaded for remote execution. The critical part of the Clone cloud approach is migrating the execution environment from the mobile device to the remote server, which implicates the issues of access control, privacy and security. It also involves in the complications management of mobile resources and of VM deployment.

Furthermore, in the system virtualization approach, sometimes the size of the VM image to be transferred over Wi-Fi/ Cellular networks is within the range of gigabytes (Shuja *et al.*, 2016). In addition, some of the approaches used the concept of migrating the entire application for processing to the remote servers such as those used in studies conducted by Liu *et al.*, (2009), Lai *et al.*, (2010), Hung *et al.*, (2012) and Liu *et al.*, (2012).

On the one hand, the approach of migrating the entire application is useful as it eliminates the partitioning and managing local resource's overhead at the local device. It does not need to take any smart decision. Instead, it simply has to offload the whole application and thus reduces the complexity of the mobile device. However, it is bandwidth-intensive and time-consuming because in the fluctuating bandwidth, offloading unnecessary parts of an application is illogical. Each application has some activities which can easily be processed at the local device while offloading of such activities causes communication overhead.

The offloading frameworks proposed for computational offloading based on system and application virtualization simply causes unnecessary overhead during offloading attempt to the remote servers (Shuja *et al.*, 2016). Thus, by evaluation and also complex procedure adopted in virtualizing the running states of machine to the remote cloud, it is considered resource-intensive as well as a time-consuming process.

Many other researchers have proposed the method level computational offloading approach and claimed to have eliminated the complexity of partitioning and of unnecessary offloading as well as management overhead. The method level computational offloading concept was once used by Rim *et al.*, (2006) to reduce the code size at the mobile device by transformation method and used Distributed Execution Transformer (*DiET*) to generate slim codes for heavy methods. The mobile device then downloaded the modified bytecodes and executed the application computation with the server. This whole transformation of codes would be heavy to process. It would also generate slim codes each time with each offloading method and thus rendering the method as time-consuming as well as resource-intensive.

Meanwhile, Cuervo *et al.*, (2010) had earlier presented MAUI, which is an energy-aware offloading system enabling fine-grained strategy during offloading to

remote infrastructure. In MAUI, offloading of application code to the remote server takes place at the method level. The ultimate goal of MAUI is to reduce energy consumption of the mobile device. This framework dynamically partitions the application into local and remote methods which are annotated by the developer using special annotation symbols. Consequently, the MAUI profiler determines the remote methods which are then offloaded to remote server. In terms of energy consumption, it has been established that offloading is not always beneficial.

Thus, with each time offloading, the MAUI Profiler and MAUI solver are invoked. If the remote server is available, then the optimization framework decides whether or not the intensive components should be offloaded. Once the process of offloading method ends, MAUI profiler gathers information as a context which is then used to better predict any future computational offloading calls.

Furthermore, the MAUI solver works to find the remote location where the offloading methods are executed. The MAUI profiler works as an input provider for MAUI solver. The authors have established that as the Round Trip Time (RTT) increases, the energy cost linearly increases almost for all the networks, including the same network type. It has also been concluded that high bandwidth (BW) technology and low RTT latency make computational offloading more preferable to attempt over Wi-Fi instead to offload over 3G which carries very low and inconsistent bandwidth.

In addition, it has been concluded that offloading codes to a cloud server residing at the distant cloud with Wi-Fi drains more energy than the same offloading to a server nearby at single hop distance. The concept of local server by the name of surrogates or cloudlet which was first used by Satyanarayanan *et al.*, (2009) has motivated the present researcher to reach a decision to use a nearby server rather than a remote cloud distant server for offloading the resource-intensive components of mobile applications. The goal of using any offloading techniques would be to minimize the local resources consumption and offloading overhead while using the MAUI profiler in this work would need extra resources such as CPU and energy itself to evaluate the individual method calls.

Similarly, Kosta *et al.*, (2012) have proposed ThinkAir, which is a concept of offloading at method level with integration of multiple VM images delegation to the cloud

server. The focus of executing parallel processing of intensive method codes at multiple virtual machine at the same time is to achieve energy efficiency of mobile devices. Kosta *et al.*, (2012) have concluded that using a memory hungry tool for image combination shows that during processing, each application can send request to many virtual machines of high computational power simultaneously in order to complete its own complex delegated computations. This approach is beneficial in offloading and complex computation of delegated components.

Nevertheless, the involvement of many virtual machine images altogether is a compute-intensive and resource-intensive process itself for the local device and results in local mobile device overhead. In addition, the method level computational offloading approach has been used in many studies by other researchers such as Dynamic Compilation and Method Execution by Chen *et al.*, (2004), Cuckoo by Kemp *et al.*, (2010), and Distributed Computational Offloading Framework (DCOF) by Shiraz *et al.*, (2014). However, general quantitative analysis in all the computational offloading approaches is still missing. Furthermore, user requirements such as delay-tolerance threshold have yet to be considered. Although the static partitioning and method level computational offloading minimize the overhead, still there must be a proper resources management tool to predict offloading cost precisely and manage the whole process of offloading efficiently.

2.9 Analytical Analysis of Method Level Computational Offloading Frameworks

This section presents an analytical analysis of the three recently developed method level computational offloading frameworks. With respect to the offloading techniques, the following aspects are critically reviewed and then analysed: partitioning, remote execution environment and reduction in communication data size. Chen *et al.*, (2004) have used the concept of Java-based wireless communication where the mobile devices are leveraged with cross platform compatibility. The mobile device can work as a personal computer and can communicate to any kind of platform or member of the network family.

Table 2.4 Summary of Method Level Computational Offloading Frameworks

Framework	Partition	Service Call	Comm. Medium	Remote Server	Predefine Parameter Mechanism	Data Size Reduction	Contribution
Dynamic Compilation and Method Execution (Chen <i>et al.</i> , 2004)	Dynamic	SOAP	3G/4G	Cloud Server	No	Data compression (ejava.util.zip)	Save energy
Cuckoo (Kemp <i>et al.</i> , 2010)	Dynamic	SOAP (RPC)	Wi-Fi	Cloud Server	No	n/a	Save energy
DCOF (Shiraz, Gani, <i>et al.</i> , 2014)	Static	SOAP (RPC)	Wi-Fi	Cloud Server	No	Deployment of SaaS model and remote services	Reduce data size and save energy

The focus of Chen *et al.*, (2004) research was to improve client-server collaboration by offloading computational task to the remote server for execution. In their research, they used dynamic partition of application at runtime, which is a resource-intensive process (Shiraz *et al.*, 2014). It needs to have an additional component act as an inference engine. This component decides at runtime based on the previous execution pattern or contextual information gathered by a sensor, to partition application. Any additional hardware (sensor) will consume more power. In addition, the deployment of inference engine needs additional computational at mobile device which is a resources intensive approach. As a result, more power is consumed.

Furthermore, it has been established from the previous research works that calling the remote server through SOAP-based offloading techniques and using mobile networks (3G and 4G) would consume more power and would be resources-intensive. SOAP supports XML as a data carrier which is crowded wordy and increases mark-up overhead based on the analysis carried out in this research. It needs longer time to read as more data presented and therefore more time is needed to parse (Nurseitov *et al.*, 2009).

Moreover, 3G is used as a communication medium which is a low bandwidth network. In fact, some features of the existing works can securely offload the tasks for remote executions. However, this may also incur additional computations which will make the framework resource-intensive for mobile computing.

There is no any mechanism defined for considering the predefined conditions such as which network to consider as a communication medium before making an offloading decision. If a distant cloud server is configured as a remote execution environment, then it will increase RTT (Abolfazli *et al.*, 2014). To reduce the size, an additional data reduction technique is defined. In order to reduce the communication data size during communication between the client device and the cloud sever, a data compression approach is used. This is done to reduce the communication data size in order to optimize communication energy process.

As the data compression and decompression also incur some amount of energy, there are compression tiers introduced both on the client and the server device. The data are compressed into GZIP format through *java.util.zip* package before being sent through wireless medium. The receiving data are decompressed at the client device before proceeding to the same to application. Consequently, the energy to be consumed in compression and decompression overlaps with the operations of sending and receiving to minimize the consumption.

The present research has fallen into the category of the first few approaches employed to save energy through computational offloading. From these approaches, two main conclusions can be made. Firstly, a low bandwidth network such as 3G is energy-draining. Secondly, the service call made through SOAP, which is an old technique of calling remote services, takes longer time to parse and read. In addition, it also causes extra baggage of communication during offloading, which turns into heavy process.

In this regard, Kemp *et al.*, (2010) have presented a better concept in terms of reducing RTT using a good bandwidth medium; however, Kemp *et al.*, (2010) have used a distant cloud server for remote execution of intensive tasks same as adopted by Chen *et al.*, (2004). Furthermore, Kemp *et al.*, (2010) have dynamically partitioned the application at method level which has led to RTT and resources-draining issues.

Moreover, the mobile network mediums (i.e., 3G and 4G) are replaced by a Wi-Fi. There is no any mechanism defined for data compression or data reduction at the client device before offloading. As the dynamic partitioning of application is a resources-intensive procedure (Shiraz *et al.*, 2014), therefore DCOF has used the concept of static

partitioning to reduce the energy consumption during runtime partitioning of application. Additionally, SaaS model of cloud computing facilitates to provide software as a service, which is then approaches by the client device to utilize.

Meanwhile, Shiraz *et al.*, (2014) have deployed the SaaS model of cloud computing at the cloud server to reduce the overhead during offloading. This overhead is the key reason which sometimes increases RTT using a limited bandwidth medium. The SaaS model and remote services are configured at the cloud server which reduces the burden of client devices to offload portions of the application alongside of intensive computational data. This is due to the fact that if the remote services do not possess the complete operational codes of the application, then SaaS model of the cloud computing will assist in configuring the execution before executing the offloaded tasks.

Distributed Computational Offloading Framework (DCOF) is resources-intensive due to the SOAP call. Shiraz *et al.*, (2014) had configured a distant cloud server as a remote execution environment similar to that of the previous two research works. Nevertheless, there is no mechanism defined for including predefined parameters to consider prior to the offloading decision.

The development in computational offloading solutions started in the year 2004, during which the whole application was offloaded for remote executions. In that year, Chun *et al.*, (2004) presented a Roam System in which the whole application was delegated to the remote server for execution. It was soon realised that, instead of offloading the whole application, only the intensive parts should be offloaded because this could reduce the size of communication data.

The concept of application partitioning such as *DiET* was developed by Rim *et al.*, (2006). Further, *DiET* was used to execute the task at a single remote server which was then modified to a concept of using multiple remote servers. This concept was developed for faster execution by concurrent processing of the task at multiple servers. Goraczko *et al.*, (2008) applied the concept of multiple servers in their *mPlatForm* work.

As the distance of the remote server execution effect RTT, a novel of concept of Cloudlet was developed by Satyanarayanan *et al.*, (2009) to reduce hop distance which

would then reduce the RTT. Satyanarayanan *et al.*, (2009) presented the concept of VM and brought the cloud closer to the execution environment. The VM was used to bring automation in the partitioning and increase compatibility of the offloaded task executions. Execution of the whole application was dynamically partitioned and offloaded with a VM which then increased the additional computation in mobile device.

The new concept was later developed in the year 2010 to reduce the additional computation. Kemp *et al.*, (2010) applied the same concept of VM with partitioning application at method level. The method level partitioning concept was used by many other researchers in their works, such as MAUI, ThinkAir, and DCOF. This level of partitioning at method level is the partitioning of application at the smallest unit (level). It minimizes the chances of offloading the unnecessary tasks (components) and hence reduces computation “C” as well as size of communication data.

More recently, the concept of code level offloading developed which is going counter the intensive operation procedure of the VM level. Shiraz *et al.*, (2015) applied the code level offloading concept in the research ECOF. Code level offloading somehow exclude the VM deployment from offloading which further reduce the operational and computational overhead prior offloading.

However, as for the present research was concerned, the three existing method level frameworks were studied and critically analysed either in terms of either the intensity of local computing or remote execution. All the selected frameworks partitioning application at method level were used to reduce the size of data and delegate the computational-intensive tasks for executions only.

Based on the comparison of the three recent research works, it can be concluded that computational offloading would be effective only if the three basic parameters B, C and D are considered. Kemp *et al.*, (2010) considered method level computational offloading; nevertheless, their research lacked the mechanism for reducing the size of communication data. Later, Shiraz *et al.*, (2014) adopted the method level computational offloading using the static approach to partition application at method level. However, the cloud server was considered as a remote execution environment, whereas the long run RTT during offloading to the cloud server would drain the power. The mechanism for the

selection of predefined parameters was also lacking in their research. Therefore, the traditional computational offloading frameworks (VM level), Class Level, Task Level and also the method level offloading frameworks would be resources-intensive if the basic three parameters in the design the framework are not considered.

It can be concluded from the literature review that the previous computational offloading frameworks have not been fully successful in making computational offloading an energy-saving solution. The techniques adopted either by whole application level migration or by virtual machine level migration are computational-intensive. They involve utilizing maximum resources which ultimately makes offloading a resources-intensive procedure. The method level computational offloading techniques adopted is partially an effective approach by offloading the smallest unit (method) for remote execution. However, the dynamic partitioning and offloading to distant cloud increase RTT and consume more resources of mobile devices. Therefore, the previous research which worked on method level offloading have not been fully battery-saving. In addition, to make the process of computational offloading beneficial, it is necessary to consider all the limited resources, limited bandwidth and compact size of mobile as well as the battery before designing the frameworks. Furthermore, the same mechanism for considering predefined parameters has been missing in the previous research works. To deploy the computational offloading application, all the necessities need to be fulfilled in order to make the process lightweight for the mobile cloud computing environment.

2.10 Summary

This chapter has reviewed and discussed the concept of mobile computing, cloud computing, and mobile cloud computing. It has explained the limitations especially the battery power limitations of mobile devices supported by studies conducted worldwide. It has further discussed the best possible approaches for augmenting the limited resources of mobile devices. It has analyzed the taxonomy of battery augmentation techniques and further explained the different techniques which can be implemented by researchers and users to curtail battery consumption.

It is also argued that the current computational offloading frameworks for Mobile Cloud Computing are the equivalent extensions of traditional computational offloading

frameworks for ubiquitous computing (i.e., mobile computing). Further, details of the computational offloading have been explained and discussed with all possible matrices and taxonomies which may influence the process of computational offloading. The review continues with an analysis of the previous research works based on all of the parameters and matrices as well as their efficiency in terms of their results. It has been concluded that the current approaches of computational offloading are heavyweight and deficient of addressing the limitations of mobile resources.

In the final part of the review, the focus has been made on discussing the computational offloading frameworks based on method level computational offloading. In this regard, three most recent method level offloading frameworks have been selected and thoroughly discussed. These frameworks have been critically analyzed based on the parameters (partitioning technique, remote execution environment, service call, data reduction and mechanism for selection of predefined parameters). It is argued that any of the three mentioned method level computational offloading frameworks are resource-intensive. As a result, power consumption of mobile devices has not been successfully reduced significantly.



UMP

CHAPTER 3

METHODOLOGY

3.1 Overview

This chapter presents the research methodology adopted to achieve the goal of the research which is “A model for power efficiency of mobile devices through lightweight method level computational offloading”. The chapter is divided into four main sections. Section 3.2 includes the research approach adopted. Section 3.3 highlights the research phases carried out to reach the target. Section 3.3 is further divided into sub sections: Sub-section 3.3.1 consists of the planning phase. Sub-section 3.3.2 includes the analysis (Problem Analysis), design (The Model, Operational Logic of Model Components, Application Execution Logic and Proposed Algorithm) and implementation phases. Sub-section 3.3.3 describes the evaluation phase of the research. Sub-section 3.3.4 provides the comparative analysis while the last section 3.4 summarises the chapter.

3.2 Research Approach

Before proposing any solution, it is important to understand the power consumption first in terms of “where and how the power consumes” in any modern mobile device. A thorough investigation of power consumption of different mobile applications and system components of the target devices will be organized by conducting experiments to determine which mobile application and system component drains power the most. Once the power draining mobile applications and system components are identified, then the basic causes will be taken into account as predefined parameters for developing the model.

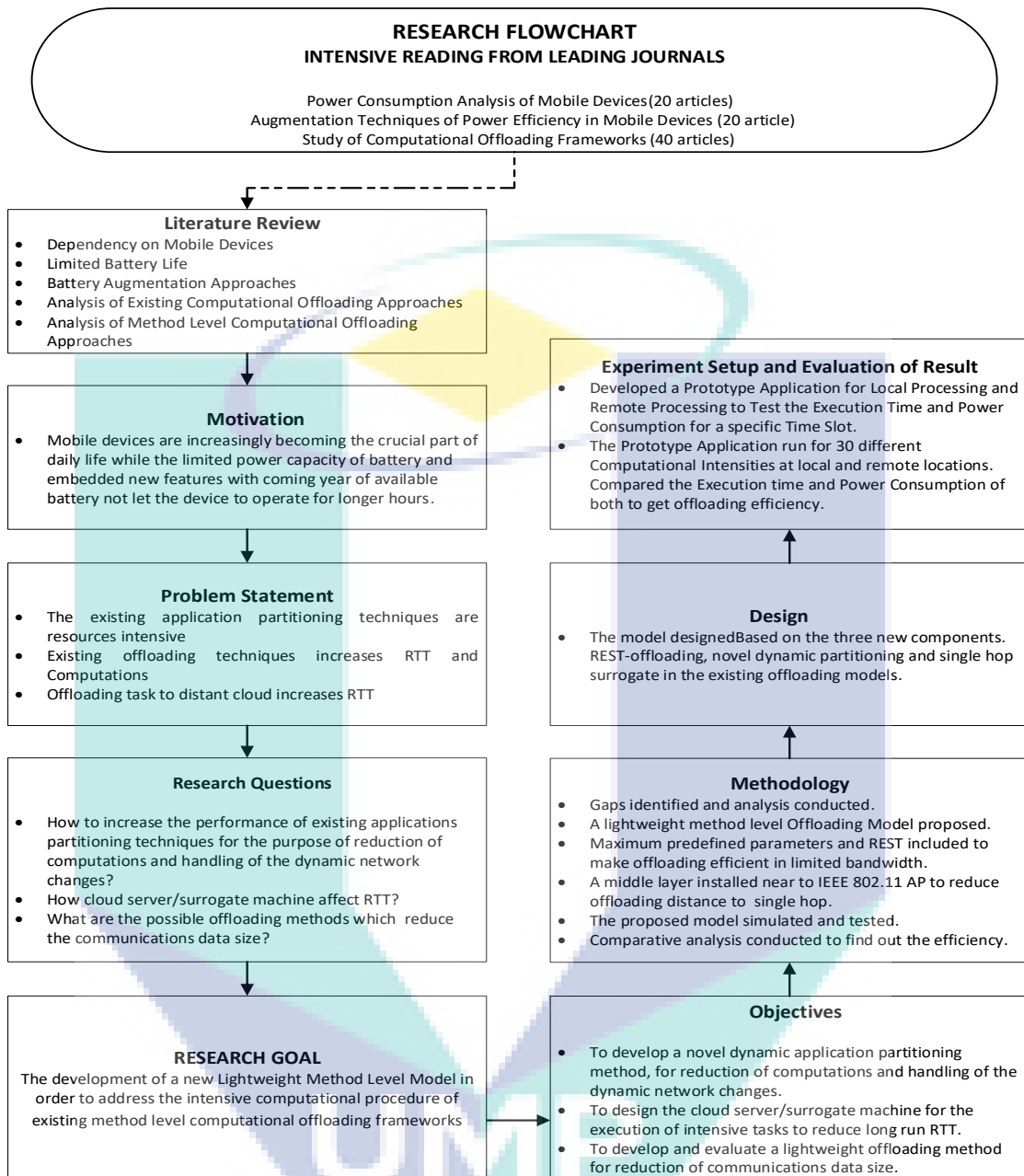


Figure 3.1 Research Flowchart

Additionally, the traditional computational offloading frameworks/models emphasis on computational intensive mobile applications using different techniques at different granularity levels, thus implicates a mechanism which is resource intensive. The word traditional been inherited from the previous research works where it refers to the existing computational offloading methods (Shiraz *et al.*, 2014). In traditional offloading the application profiling and partitioning at runtime utilizes maximum resources of SIDs,

as a result, the Turnaround Time (TT) and Energy Consumption (EC) costs of applications increases.

The focus of this research is to reduce both, in order to achieve maximum throughput in minimum time along with curtailing the extra battery drain. The proposed method level computational offloading model is inspired by the concept once used in code transformation method (Rim *et al.*, 2006). Rim (2006) partitions application at method level and offloads the heavy parts for remote execution; while a code transformation technique used at mobile device for reducing the code size by *DiET* (Distributed Execution Transformer) to generate slim codes for heavy methods. The *DiET* generator of slim code is an additional computational load on mobile device with each offload.

This research intends to use REST (Representational State Transfer) which is an architectural style of World Wide Web and communicates over HTTP protocol, instead of slim code concept or sending whole executable codes direct to a server. REST was developed by Roy Thomas Fielding (Fielding, 2000). REST was initially developed to restructure the Web Applications while REST not been used as a carrier protocol in mobile computing for offloading purposes. Subsequently, this research eliminates the necessity of generating slim codes for reducing the size of data as well as, replaces XML with JSON as a data carrier file which is comparatively more lightweight and fast parsing than XML. Normally in the past researches, SOAP (Simple Object Access Protocol) has been used to establish client server connection for transmission of data between Web Applications. The reason of using REST instead of SOAP is the portability, simplicity and lightweight nature. Hence, REST is more lightweight against *DiET*, SOAP and RPC (Giorgio *et al.*, 2010).

Furthermore, to reduce the delegation of computational intensive tasks to cloud server which resides at multi-hop distance; a middle layer solution is proposed which was first presented by Satyanarayanan *et al.*, (2009) and used by Magurawalage *et al.*, (2014). Offloading any computational intensive task direct to a distant cloud always result in a long run RTT which is resource intensive. In order to reduce the distance, Satyanarayanan *et al.*, (2009) used the concept of Cloudlet, which is a small cloud in the nearest computing

environment. Normally, the Cloudlet concept presented by Satyanarayan *et al.*, (2009) were to install in the nearest café, library, coffee shop or any public place. This research used a similar concept with further addition of a single hop where any personal computer (PC) connected to IEEE 802.11 access point acts as a cloudlet layer in between the client device and the cloud infrastructure. The cloudlet layer serves as a confined service in a faster communication approach of client device to offload. Again, on top of the middle layer, an algorithm was proposed in order to define maximum parameters before taking any offloading decision, such as to check network type and network bandwidth, thus, eliminates the issues of limited bandwidth and size of data to be offloaded. It will also consider an offloading decision, to either offload to the local cloud (cloudlet) or to execute the task using mobile device's resources.

The proposed algorithm encompasses the user preferences, applications requirement and maximum predefined parameters for a precise offloading decision. Application requirements contain the nature of application such as real-time, fidelity adaptive and intensive parts in terms of computation and communication. User predefined parameters comprise of reliability and secure offloading while predefined parameters include current battery level, type of network available, execution time and most importantly, the available network bandwidth.

The proposed model will be developed and then will be evaluated in the simulation environment using SDK (Software Development Kit), Java and REST/SOAP APIs along with Glassfish Server. A prototype application named REST-Offload, Android-Local and Traditional-Offload will be simulated for testing the intensity of applications in terms of computation both on local and offloaded time. The Local Execution component of the prototype application will be designed to execute the whole application locally. Traditional offloading component will be designed using the SOAP offloading techniques, the cloud server for remote execution and the application will be partitioned dynamically based on the existing techniques.

Furthermore, while offloading, each offload will be tested and the intensity of application will be analyzed in terms of resources utilization (battery consumption) and the Execution Time of application at mobile device and at remote server. The analysis

will be considering all three scenarios (Local Execution, REST-Offload and Traditional-Offload). Accordingly, the power consumption of mobile device will be tested while executing the tasks locally as well as when offloading task to remote servers for execution. With 30 different computational intensities starting from 160x160-450x450, the Execution Time and Energy Consumption will be tested on all three scenarios. Each intensity will be tested 20 times for validation purposes. For further measurement of precisions, the value of sample mean for each experiment will be calculated which is signified with 99% confidence interval for the sample space of five values. The lightweight nature of the proposed computational model will be tested and validated by the comparison of REST-Offload results of computational intensive tasks execution against the benchmarks.

3.3 Research Phases

By onion approach, peeling the layers one after another, the research is carried out in few fundamental phases, which are presented in Figure 3.2. Starting from the planning phase, the research proceeds to analysis phase processes which are reviewing articles, identifying gaps, designing research questions and defining objectives. The analysis design and implementation phase consist of analysis of problem, designing the model and proposing algorithm for pre-defined parameters. It also includes the implementation of model and selection of dataset. The last phase consists of the evaluation and testing of the model.

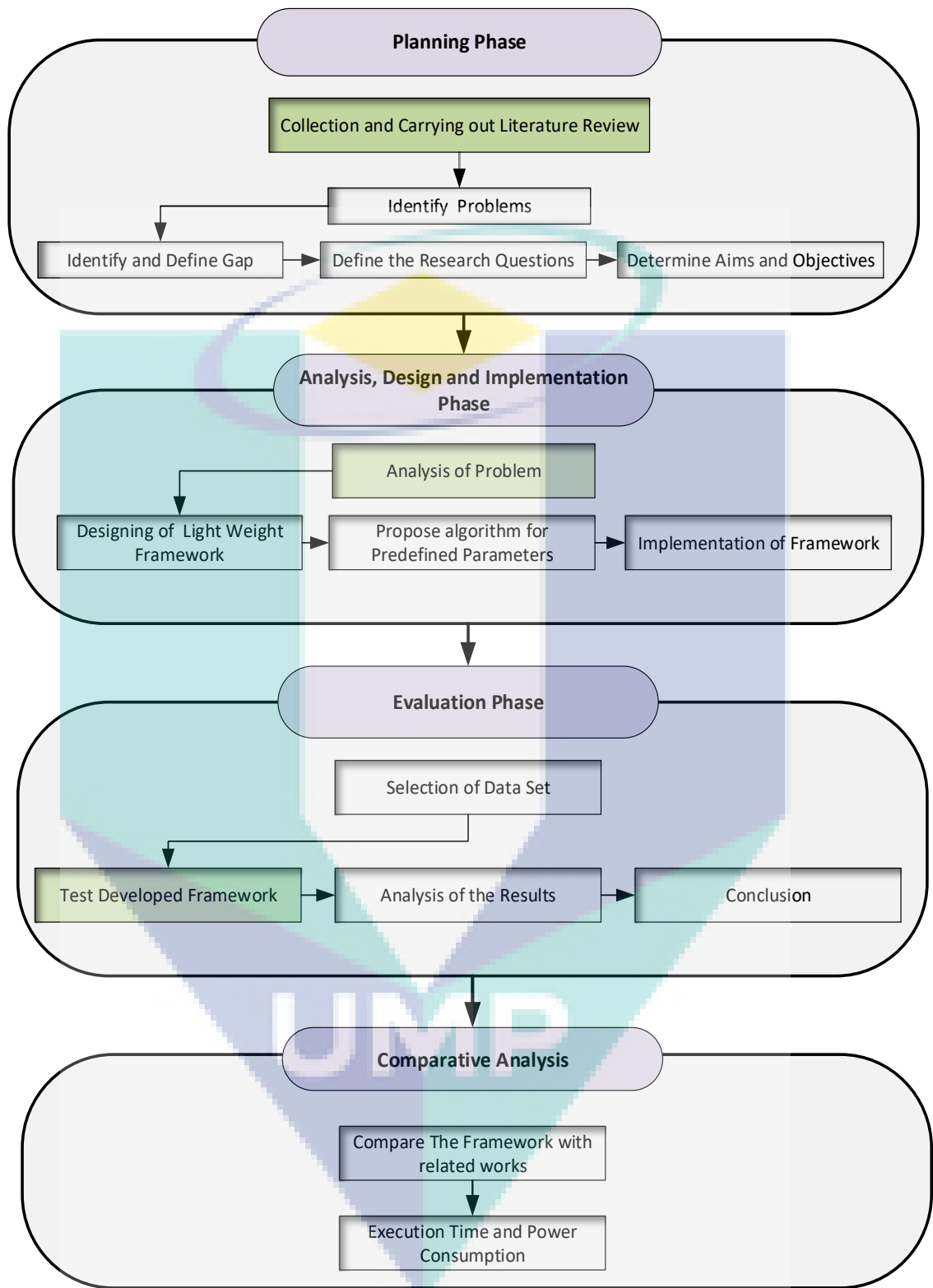


Figure 3.2 Operational Model

3.3.1 Planning Phase

The planning phase is the starting phase of the research which consists of many sub-phases including the literature review. In the literature review, the main problems are identified first before precisely identifying the research gap. A total of 80 articles closely related to the title of the research, published between 2009 and 2016 collected from different standard journals indexed in ISI and Scopus are critically reviewed. The total articles are divided into two groups. One group of articles were discussing the problem in general and consists of about 60 articles which were reviewed for understanding the background of problem. After understanding the overall problem of insufficient capacity of mobile battery power, the researcher investigated other problems and their possible solutions. In addition to that, the researcher sought to identify other research approaches that had been used in addressing the problem in the past. The second group of 20 articles investigated the specific gaps identified in the first group of articles reviewed. In this second group of articles, the different research frameworks and models based on computational offloading developed in the near past were studied critically and then compared and analysed in order to identify the research gap.

At the completion of review stage, it is observed that the previously proposed models on computational offloading are unnecessarily complex to configure and heavy to operate while utilizing maximum resources of the mobile device; which ultimately causes additional consumption of power. Furthermore, traditional computational offloading models that focus on computational intensive mobile applications using different techniques at different granularity levels, implicates a mechanism which is resource intensive. The application profiling and partitioning at runtime utilizes maximum resources of SIDs, as a result, the turnaround time and energy consumption cost (ECC) of applications increases. The focus of this research is to reduce both (execution time and energy consumption), in order to achieve maximum throughput with minimum time along with curtailing the extra battery drain. Moreover, the existing models have not precisely considered the three basic parameters (available bandwidth B , size of data D and the computation required C) which are important for any computational offloading model to produce efficient results.

Based on the shortcomings of existing computational offloading models, the research questions which were not yet completely answered by the previous research works were identified. Further to that, to answer the research questions, the goal of the research and objectives of the research were determined. The goal of the research is to develop a model in order to minimize the additional computation at mobile device by offloading the computational intensive tasks. However, the proposed model must avoid the additional computation of application partitioning which has not been addressed in the existing computational offloading frameworks/models. Furthermore, the size of data exchange between mobile device and the remote execution counterpart must be reduced in order to minimize RTT. If RTT is reduced, it will reduce the power consumption (Kumar *et al.*, 2011).

3.3.2 Analysis, Design and Implementation Phase

In this phase of the research, the problem was analyzed first. Based on the observation of problem analysis, the model was designed and then implemented through simulation in order to test its efficiency against traditional computational offloading frameworks/models.

3.3.2.1 Analysis of the Problem

The analysis phase consists of analysis of problem which determines the design adopted in this study. The traditional offloading solutions (VM Level, Whole Application Level and Method Level) which involved in dynamic runtime migration of the resource intensive components of mobile applications create resources management overhead (Hung *et al.*, 2012). Enormous computational offloading frameworks and models used dynamic application partitioning and profiling techniques (Giurgiu *et al.*, 2009; Cuervo *et al.*, 2010; Chun *et al.*, 2011; Shiraz *et al.*, 2014). All the traditional computational offloadings are focused on how to partition runtime application, how and where to offload the intensive parts. However, the offloading models have failed to consider the additional cost of runtime migration of offloading parts. It has been analyzed that the traditional offloading models with offloading intensive parts through dynamic partitioning to distant cloud are resources intensive; as a result, the execution time and energy consumption

increases. The resource intensive problem of traditional offloading analyzed in terms of Execution Time (ET) and Energy Consumption (EC) as:

I. Analysis of the Energy Consumption (EC) Cost

The energy consumed using traditional computational offloading techniques during runtime computation offloading is denoted by Total Energy consumption (E_T) in Joules (J). The total energy consumption (E_T) is equal to the energy consumed in saving the data states of running instance of application to offload, energy consumed in runtime component offloading, energy consumed for uploading the remote executable methods to remote server, energy consumed in idle time waiting for results to come and energy consumed in returning the result data (download result) files to mobile device. Therefore, the total energy consumption for each component of mobile application offloaded at runtime to remote server is given by the following equation:

$$E_T = E_S + E_M + E_{UP} + E_I + E_{DW} \quad 3.1$$

Where,

E_S - Energy consumed in saving running app states: represents energy consumed in saving the running instances of the mobile application.

E_M - Energy consumed in component Migration (E_M): represents the energy consumed during offloading intensive component of mobile application.

E_{UP} - Energy consumed in Uploading preferences (E_{UP}) represents energy consumed in uploading the data file (which is known as preferences file) to remote server node at runtime.

E_{DW} - Energy consumed in Downloading result file (E_{DW}): represents energy consumed in downloading the resultant data file to mobile device.

As there are several components of each application to offload that may be intensive, then, let E denotes the finite set of total energy consumption for runtime offloading all the intensive components of mobile application to remote server. Then E

can be calculated as: Lets E_{Ca} represents the cost of energy consumption for runtime offloading a single component of the mobile application. Whereas $a=1, 2, \dots, n$.

$$\therefore E = \{E_{C1}, E_{C2}, \dots, E_{Cn}\}$$

As E_{Ca} denotes the energy consumption of a single component which is a positive real number, hence, by using the set builder notation E_{Ca} is represented as:

$$E = \{E_{Ca} : E_{Ca} \in \mathbb{R} \wedge E_{Ca} > 0\} \quad \text{Where } a=1, 2, \dots, n$$

The energy consumption of a single component belongs to the set of real numbers and is greater than 0. Next, the total energy consumption of a runtime offloading application is equal to the sum of energy consumption cost of all the instances $a=1, 2, \dots, n$. Let β denotes the total energy consumption of the runtime application offloading of all the instances E_{Ca} , where $a=1, 2, \dots, n$.

Therefore, β is represented as follow:

$$\beta = (E_{C1} + E_{C2} + \dots + E_{Cn}) \implies \forall E_{Ca} \in E \wedge |E| \geq 1 \quad \text{where } a=1, 2, \dots, n$$

By using the summation notation, the total energy consumption cost β of runtime computational offloading of mobile application is represented in Equation 3.2 as bellow:

$$\beta = \sum_{a=1}^n E_{Ca} \implies \forall E_{Ca} \in E \wedge |E| \geq 1 \quad 3.2$$

Equation 3.2 describes that for all the E_{Ca} 's which denotes the energy consumption of each component at runtime offloading, belongs to the set of total energy consumption E and cardinality of set E must be greater than or equal to 1. E is the set of energy consumption cost of the components of the mobile application which are offloaded at runtime. The precondition validates that E is a non-empty set. One component (E_{C1}) of Equation 3.2 was considered to be evaluated and analyzed for energy consumption.

Table 3.1 Energy Consumption Cost EC1 of Offloading Matrix Multiplication Service in Traditional Computational Offloading

Matrix Size	Energy Consumption Cost (J)				Standard Deviation (SD)	Confidence Interval
	CPU (J)	LCD (J)	Wi-Fi (J)	Total consumption (J)		
160x160	3.9	1.4	2.5	7.72	0.420714	7.8(+/-)0.966
450x450	78.4	7.9	15.2	102.08	8.517159	101.5(+/-)1.388

This component carries 30 different computational intensities (160x160-450x450) with the increase of 10 in each higher intensity. The Energy Consumption Cost (EC_1) of runtime computational offloading of the Matrix Multiplication service was calculated using Equation 3.1. The Energy consumption results of traditional computational offloading techniques are displayed in Table 3.1. The lowest and highest intensities are given as sample while the results of intensities in between are presented in Table A.1 in Appendix A.

The size of a matrix shows the dimensions of matrices which are going to be multiplied. The Energy consumption cost in Joule (J) indicates the mean energy consumption of a sample space of 20 values in each experiment. In order to know variations in the values of sample space, the standard deviation (SD) is calculated. The confidence interval column shows the range of the sample mean calculated of the whole sample, 20 values in each experiment with 99% confidence. Figure 3.3 displays the energy consumption of matrix multiplication during runtime offloading in traditional offloading techniques.

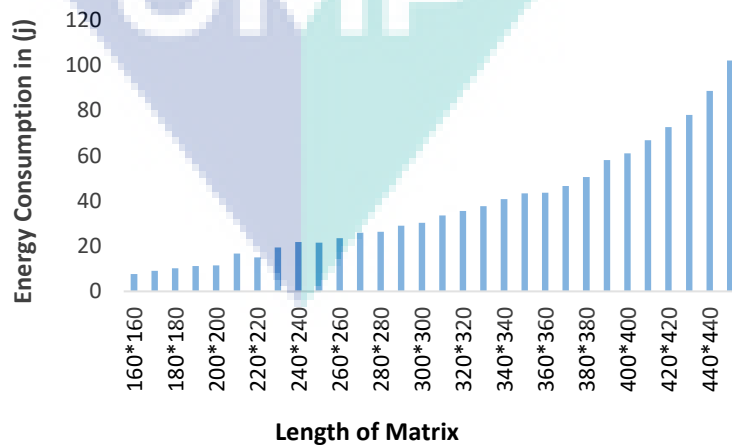


Figure 3.3 EC cost of matrix multiplication in traditional offloading

It shows that as the size of matrices increase, the complexity increases and as the complexity increase the energy consumption cost also increase. Based on all the collected results as displayed in Table A.1 Appendix A, the energy consumption cost of matrices 160x160 offloaded at runtime is 7.72 J which is approximately equal to energy consumption of matrices 180x180 and 200x200. It shows that the runtime offloading component is not much complex to process and hence consumes an average 10J. As the complexity increases with the size of matrices such as 260x260 and so on until 440x440, the energy consumption cost reaches up to 80 J. The total percentage of increase in energy consumption between 160x160 and 440x440 is about 1300%. The analysis of results shows that using traditional offloading that works on runtime computational offloading with virtual machine migration or whole application migration are energy intensive because they need much energy to manage resources and to handle calculations.

II. Analysis of Execution Time (ET) Cost

The time consumed by traditional computational offloading frameworks/models during runtime computation offloading is evaluated by Time Cost (T_C) in Joules (ms). The total time cost (T_C) is equal to the time consumed in saving (T_S) the data states of running instance of application to offload, time required in runtime component offloading (T_O), time required for uploading the remote executable methods to remote server (T_U), time required in idle time waiting for results to come and time for returning the result (download) data files (T_D) to mobile device. Therefore, the total time consumption for each component of mobile application offloaded at runtime to remote server is given by Equation 3.3 as:

$$TC = TS + TO + TU + TD \quad 3.3$$

As there are many components denoted as intensive and needed to offload, let T denotes the finite set of total time consumption for a runtime offloading all the intensive components of mobile application to a remote server. Then T can be calculated as:

Let T_{Ca} represents the cost of time for runtime offloading a single component of the mobile application, where $a=1, 2, \dots, n$

$$\therefore T = \{T_{C1}, T_{C2}, \dots, T_{Cn}\}$$

As T_{CA} denotes the time consumption of a single component which is a positive Real number. Hence, by using the set builder notation T_{CA} is represented as:

$$T = \{T_{Ca}: T_{Ca} \in \mathbb{R} \wedge T_{Ca} > 0\} \quad \text{Whereas, } a=1, 2, \dots, n$$

The time consumption of a single component belongs to the set of real numbers and is greater than 0. Next, the total time consumption of a runtime offloading application is equal to the sum of time consumption cost of all the instances $a=1, 2, \dots, n$. Let Y denotes the total time consumption of the runtime application offloading of all the instances T_{Ca} , whereas $a=1, 2, \dots, n$. Therefore, Y is represented as:

$$Y = (T_{C1} + T_{C2} + \dots + T_{Cn}) \Rightarrow \forall T_{Ca} \in T \wedge |T| \geq 1 \quad \text{whereas } a=1, 2, \dots, n.$$

By using the summation notation, the total time consumption cost of runtime computational offloading of mobile application is represented in Equation 3.4 as follow:

$$Y = \sum_{a=1}^n T_{Ca} \Rightarrow \forall T_{Ca} \in T \wedge |T| \geq 1 \quad 3.4$$

Equation 3.4 describes that for all the T_{Ca} 's which denotes the time consumption of each component at runtime offloading, it belongs to the set of total time consumption T and cardinality of set T must be greater than or equal to 1. T is the set of the time consumption cost of the components of the mobile application which are offloaded at runtime. The precondition validates that T is non empty set. One component of Equation 3.4 (T_{C1}) considers and the time consumption cost of the component (offloading matrix multiplication component) of the prototype application at runtime offloading is evaluated for 30 different computational intensities (160x160 to 450x450).

The prototype application developed consists of one component; matrices multiplication services. This service gets two random matrices as an input from the user then offloads the matrices and performs the multiplication operation at server which is a computational intensive operation. The prototype application is developed using SOAP as a carrier protocol while application partitioned at method level and then migrated to

Table.3.2 Time Consumption Cost TC1 of Offloading Matrix Multiplication Service in Traditional Computational Offloading

Matrix Size	Execution Time Milliseconds (ms)	Standard Deviation (SD)	Confidence Interval
160x160	9608	535.7891	9608 (+/-)1236.39
450x450	189523.8	2089.393	189523.8 (+/-)4821.52

remote cloud servers for processing. By runtime computational offloading with the traditional approaches, multiplying matrices are evaluated for 30 different computational offloading intensities 160x160 to 450x450 with the increase of size 10 with each intensity. The total Execution Time (T_{ET}) and total Energy consumption (E_T) are evaluated by offloading the service components of mobile application at runtime. Table.3.2 shows the Execution Time of the component Matrix Multiplication during runtime offloading. The result displays highest and lowest intensities as samples while the complete results of all 30 intensities are given in Table B.1 Appendix B. The attribute of matrix dimensions shows the size of matrices to be multiplied and the time consumption cost in milliseconds (ms) indicates the mean time consumption of a sample space of five values in each experiment. To know the variation in the values of sample space, standard deviation (SD) is calculated. The confidence interval column shows the range of the sample mean calculated of the sample five values in each experiment with 99 % confidence. Figure 3.4 displays the time consumption of matrix multiplication during runtime offloading in traditional offloading techniques.

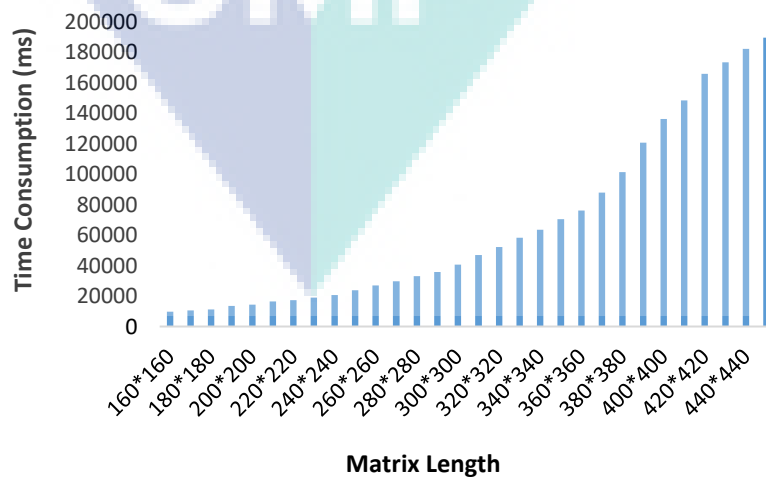


Figure 3.4 ET cost of matrix multiplication in traditional offloading

It shows that, as the size of matrices increases, the complexity increases and as the complexity increases, the time consumption cost also increases. While offloading, the execution time cost increases with the increase of uploading data size (matrix size). It also increases due to saving the running states of application each time. The time consumption cost of matrices 160x160 offloading at runtime is 9608ms which is approximately equal to time consumption of matrices 180x180 and 200x200. It shows that the runtime offloading component are not much complex to process and hence consume an average 10000 ms. As the complexity increases with the size of matrices such as 260x260 and so on until 440x440, the time consumption cost reaches up to 189,523 ms.

The analysis of entire results shows that by using traditional offloading techniques based on virtual machine migration, the whole application migration and dynamic method level migration to cloud servers are time consuming. It involves much time to manage resources and handle calculations. The distant cloud execution increases RTT which causes extra battery drain. Further to that, the analysis of problem phase surfaces the causes of why traditional computational offloading is resource intensive and it helps in designing the proposed offloading model of this research.

3.3.2.2 Designing of the Lightweight Model

After the complete analysis of problem which is “runtime migration of computational intensive tasks to distant cloud (cloudlet) are resources intensive”, the observation is tabulated in analysis phase. Based on the observation of analysis phase, a model for power efficiency of mobile devices through a new lightweight method level computational offloading is designed. The real experimentation devices have been selected to implement the model. This selection was made to observe the real battery consumptions. There is a possibility to script the mobile battery in the existing CloudSim tool, however, modifying CloudSim for the battery consumptions might change the parameters and may affect the accuracy of the results. Therefore, the use of CloudSim has been avoided. The test setup is adopted from the literature where most of the previous research works conducted the experiments in the similar way using real device. The setup consist of Mobile device, Remote Computing Device, Network Device and Simulations as shown in Table 3.3. The rest of the details on designing the proposed new lightweight method level computational offloading model is provided in Section 4.3, Chapter 4.

Table 3.3 Test Setups of Previous Research Works

Research Works	Mobile Component	Computing Component	Network Component
Cuevro <i>et al.</i> , (2010)	Mobile Device	Maui Server	Wi-Fi & 3G
ThinkAir <i>et al.</i> , (2012)	Mobile Device	Cloud Server	Wi-Fi
Shiraz <i>et al.</i> , (2015)	Mobile Device (Smartphone)	Cloud Server	Wi-Fi (radio type 802.11 g)
Shuja <i>et al.</i> , (2016)	Mobile Device (Smartphone)	Cloud Server	Wi-Fi

3.3.2.3 Implementation Phase

Once the completion of designing phase completed, the model is then developed and evaluated in the simulations environment using SDK (Software Development Kit), Java and REST/SOAP APIs along with Glassfish Server. Three components of the prototype application have been developed. SOAP-Offload is developed to analyze the results of traditional computational offloading in terms of execution time (ET) and Energy Consumption (EC) based on method level offloading to remote cloud servers. The Android-Local is developed to collect results of the computations performed pure locally at mobile device while REST-Offload is developed based on the proposed model to identify the ET and EC of intensive components at surrogate. Matrix multiplication in mathematics is considered one of the computational intensive (computational complex) calculations. Matrix multiplication was selected, in order to test the three developed porotype components for ET and EC, to analyze the performance (ET and EC) of each scenario.

3.3.3 Evaluation Phase

The significance of the proposed model is evaluated by simulating in real mobile cloud computing environment. A prototype application Android-Local, Android-Soap, and REST-Offload is developed for android mobile devices, in order to test different computational intensities in all three scenarios. The execution performance of application

is conducted in reference with execution time of application (response time) and power consumption during execution. A process of evaluation was organized in order to test the model. The dataset selected are of 30 different computational intensities of matrix multiplication starting from 160x160 to 450x450. Each intensity was tested five times for validation purposes. For further measurement of precisions, the value of sample mean for each experiment was calculated which is signified with 99 % confidence interval for the sample space of five values. The lightweight nature of the proposed computational model was tested and validated by the comparison of REST-Offload results of computational intensive tasks execution using traditional offloading frameworks/models and of local execution at mobile device. Furthermore, to measure the efficiency of the model, the percentage efficiency equation has been used. This equation calculates the percentage efficiency in execution time and power consumptions against the local and traditional computational offloading. The equation used is:

$$Y = P\% * X \quad 3.5$$

Where, Y is the first variable referred to the execution time and energy consumptions while X referred to the parameters of local and traditional offloading. All the empirical results were collected by conducting experiments with (DUTs) of two different vendors Samsung Galaxy A5 and ASUS Zenfone5. By the same comparison, the significance of the proposed solution is proved in this research. The experimental setup has been elaborated here, along with the tool used in these experiments, techniques for collection of data and statistical techniques used for operating the collected data for meaningful results.

3.3.4 Comparative Analysis

In order to justify the results of the proposed lightweight method level computational offloading model, the recent research work, closely related to the proposed model was selected. The commonalities of all the selected models are stated based on the design of the model. Further, all the differences of the models are tabulated and discussed. At the end, the results of the proposed method level offloading model then compared to the benchmarking data collected from the previous research works, individually with the results of each selected model in terms of ET and EC. Likewise, the percentage efficiency

of each model was compared and contrasted against each other. At the end, efficiency of the proposed model drawn against all the three approaches in terms of ET and EC.

3.4 Summary

This chapter was designed to state the methodology adopted for achieving the goal of the research. The goal of the research is to design and develop a framework for power efficiency of mobile devices through lightweight method level computational offloading. Starting from the planning phase of the research, the research carried out into analysis into analysis phases.

Based on the outcome from the analysis phase the model was designed. The designed model is then implemented by developing a porotype application in real mobile cloud computing environment. In the evaluation phase the data set was selected for testing and evaluation purpose. In the last phase of the research (Comparative Analysis) the results of REST-Offload are comparatively analysed with the three most recent method level computational offloading frameworks/models. Based on the analysis results the efficiency in terms of EC and ET obtained.

The logo for UWP (Universiti Utara Malaysia) is a large, stylized 'U' shape. The top part of the 'U' is a light blue horizontal bar. The two vertical sides of the 'U' are light blue. The bottom part of the 'U' is a light blue inverted triangle. The letters 'UWP' are written in white, bold, sans-serif font across the bottom part of the 'U'.

CHAPTER 4

DESIGN AND IMPLEMENTATION

4.1 Overview

This chapter presents the proposed lightweight method level computational offloading model in order to address the heavy procedural issues of traditional computational offloading approaches. The chapter is divided into six sections. Section 4.1 gives an introduction to the chapter. This is followed by Section 4.2 which discusses the relationship between computational offloading and execution time. Then, Section 4.3 describes the proposed lightweight method level computational offloading model while Section 4.4 explains the operational logic of the proposed model and presents the proposed computational offloading algorithm. Meanwhile, Section 4.5 provides details of implementation, evaluation of the proposed model, and also includes data gathering procedures of the experiments conducted. Finally, Section 4.6 concludes the chapter.

4.2 Computational Offloading and Execution Time

The mobility of mobile devices and the changing network bandwidth have led the traditional computational offloading frameworks or models to employ the heavyweight procedure for processing of the intensive components of mobile applications. Due to the limited feature and fundamental limitations in wireless networks, lightweight computational offloading models would be needed in order to achieve the best possible results. The traditional computational offloading basically works in three parts, namely, initialization of offloading, offloading of intensive task to remote servers and execution of the task at remote server. During the first phase, the network information, availability of servers and contextual information would be gathered by using sensors. Then, during

the second phase, VMinstance would be created in the local mobile device where the application partitioning and migration of the VM instance would take place. Altogether, this whole process of three phases would be a heavyweight computational intensive process itself and could not produce potential results. Thus, a lightweight method level computational offloading model was proposed to address the limited computational performance of mobile device and limited bandwidth of transmission. In this chapter, the architecture of the proposed solution is modeled and the operating procedures are explained. Compared to the traditional offloading frameworks or models, the proposed lightweight computational offloading model would result in an efficient gain in the performance of mobile devices. Compared to local execution, the execution time of the applications would significantly be reduced. Therefore, this would directly affect battery consumption and its life.

The turnaround time (TT) of an application to offload is very important to be considered before making the offloading decision. It has been claimed that a longer TT would affect the battery (Kumar, 2011). In addition to that, the applications such as speech recognition, natural language translation and image manipulation would need to have real time processing. Only the spontaneous processing for all the real time systems would render the system to be a useful tool. Numerous past studies have claimed that computational offloading would lead to saving considerable amount of the battery life while reducing the TT. In principle, while processing offloading to remote servers, the mobile device resources would need to be free of computational load. However, it should be kept in mind that mobile device would need to use sufficient amount of energy during activities such as establishing a connection with remote server, sending a request, waiting until a task would be completed and obtaining the result back and then closing the connection.

The whole process starting from establishing a connection to remote servers until closing the connection after completion would take a significant amount of time (Kumar *et al.*, 2011). The longer the TT, the poorer the performance of CPU would be. In addition, the fluctuating lower bandwidth would cause lengthier TT while transmitting huge amount of data. The relationships between TT, communication bandwidth, amount of data to be exchanged, mobile speed, server speed, and energy consumption are illustrated in Figure 4.1. An increase in TT would increase battery consumption; thus, if the total

energy consumption during the whole process is less than the amount needed by the mobile device to consume on its own execution, then offloading would be effective.

4.3 The Model

Traditional computational offloading models have adopted different approaches for computational intensive mobile applications using different techniques at different granularity levels. This would implicate a resource intensive mechanism. A middle layer solution was presented for the first time by Satyanarayanan *et al.* (2009) and the same solution was also used by Magurawalage *et al.*, (2014). Offloading any computational intensive task direct to a distant cloud would eventually result in resource intensive due to long run RTT.

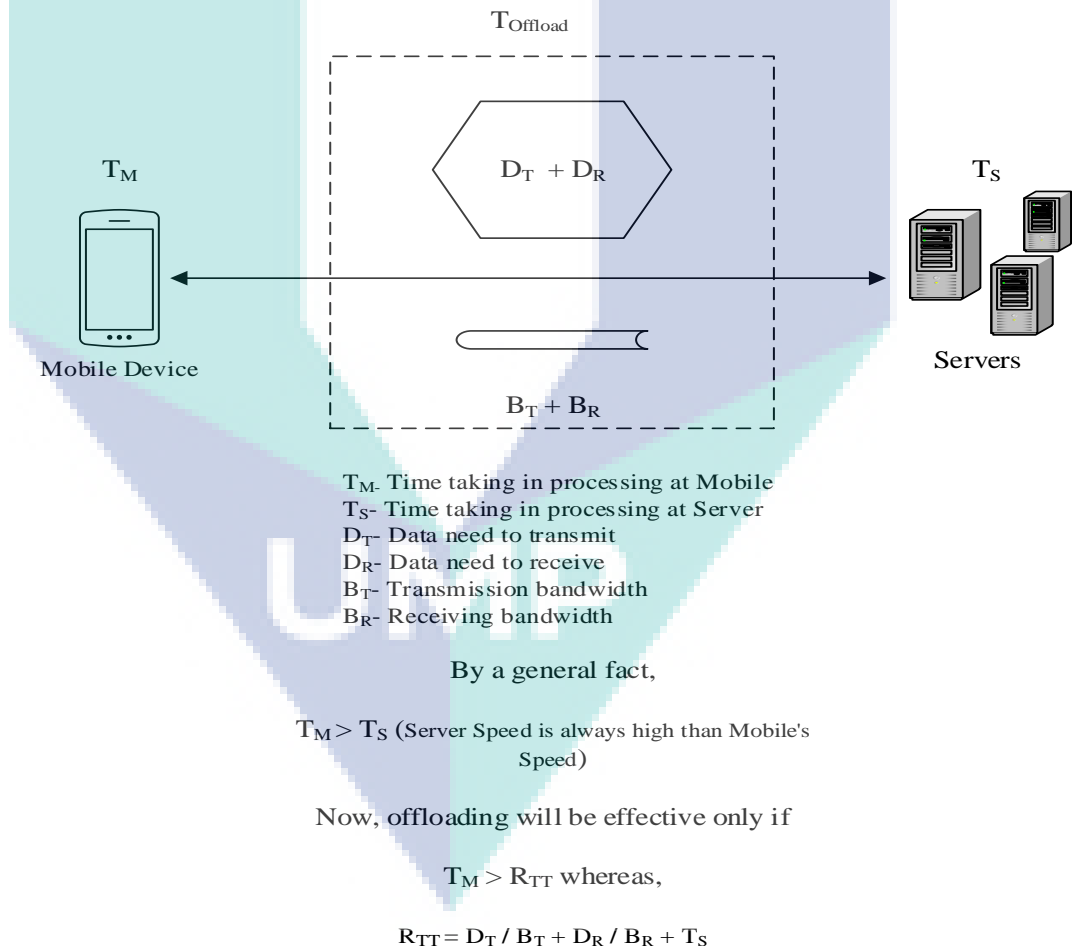


Figure 4.1 Computational Offloading and Execution Time/Turnaround Time

In order to reduce the distance to a single hop, a cloudlet layer would be used and installed next to IEEE 802.11 access point, in between the client devices and its cloud infrastructure. This cloudlet layer would serve as a confined service which is in the fast approach to client device for offloading the task.

The single hop surrogate is the modified concept of existing cloudlet. The hop distance would affect RTT. This had been analyzed by Aiguo *et al.* (1998) who stated in their findings as shown in Table 4.1 that if hope count increased, the packets would have to go through many routers. At each router, the packets would have to consume a certain amount of time to be routed for the next router and this would be repeated continuously until reaching the destination. Thus, at each router, packet delay would occur and this would increase the overall delay as the hop count increased. Figure 4.2 illustrates the conceptual diagram of a mobile device that is connected to remote the servers through single hop, limited multi-hop and unlimited multi-hop.

The multi-hop which consists of unlimited hops is the initial concept where the distant cloud server has to serve as a remote computer. The cloudlet concept has brought the cloud closer to the computing environment and reduced RTT (Satyanarayanan *et al.*, 2009). The last one which is a single hop, is the modified concept of this research which is presented to bring the computing to a single hop and reduce RTT further.

Table 4.1 The Hop-Count and Average Delay

Region	Average Hop Count	Average Delay
WEST	10.7	33.6ms
Mountain	13.7	60.2ms
Central-East	14.4	94.8ms
East	15.5	99.3ms

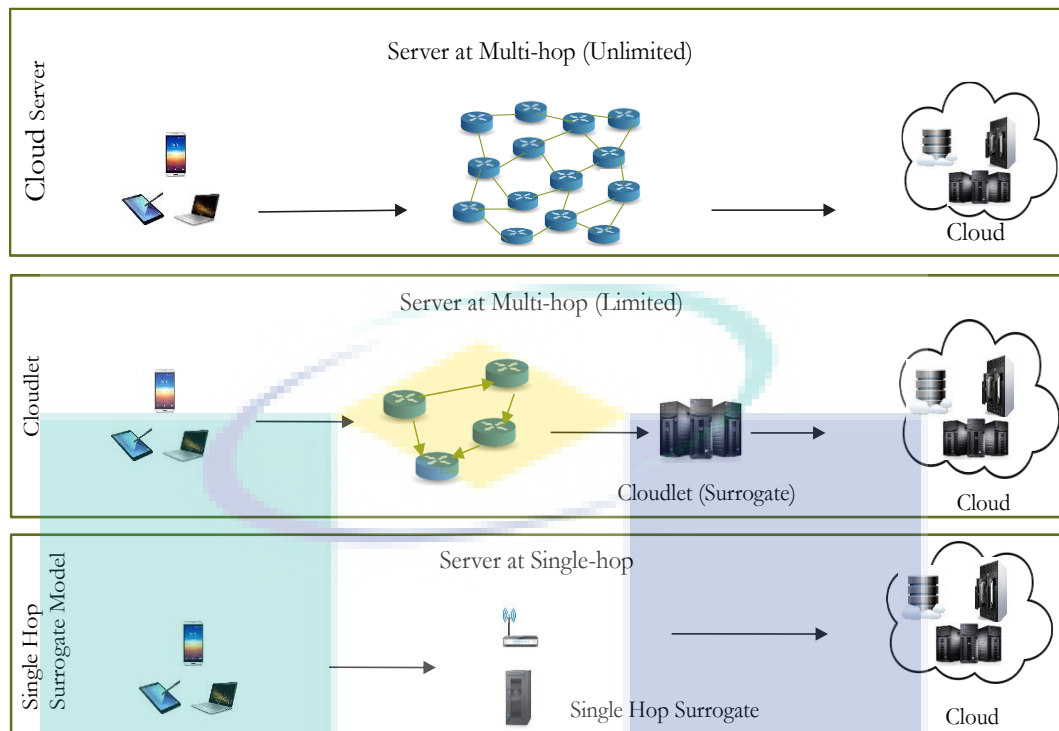


Figure 4.2 Mobile device connection to remote servers through single hop, limited multi-hop and unlimited multi-hop

Furthermore, in traditional offloading models, offloading takes place at VM level, application level, task level and method level. All the three levels, namely, the VM level, the application level and the task level would involve extra computation in the local device and would therefore be resource intensive. In the proposed solution, offloading would take place at the method level where computational intensive methods would be identified before taking offloading decision. This concept was once used by Rim *et al.*, (2006) to reduce the code size in the mobile device by transformation method and developed Distributed Execution Transformer (*DiET*) to generate slim codes for heavy methods. The mobile device would download the modified bytecodes and execute the application computation with the server.

This whole transformation of codes involves heavy processing and generates slim codes each time with each offloading method; thus, this would be time consuming as well as resource intensive. In the proposed lightweight method level model, REST (Representational State Transfer) which is an architectural style of World Wide Web communicating over HTTP protocol was used instead of the slim code concept or sending whole executable codes directly to the sever. Normally in the past solutions, Simple

Object Access Protocol (SOAP) had been used to establish client service connection and transmit data. Instead of SOAP, REST was used because of its portability and lightweight as well as simple nature.

On top of the middle layer, an algorithm was proposed for the purpose of taking decision, and offloading either to the local cloud (cloudlet) or executing the task using the mobile device's resources. The proposed algorithm encompasses user's preferences, application's requirement and maximum predefined parameters in order to take useful offloading decision precisely. Application's requirement includes the nature of the application such as real-time, fidelity adaptive and intensive parts in terms of computation and communication. User predefined parameters comprise of reliability and secure offloading while predefined parameters include current battery level, type of network available, execution time and most importantly, available network bandwidth.

Figure 4.3 shows the architectural diagram of the proposed light weight method level computational offloading model. Every computational offloading approach in the present study consists of the following three main components: the mobile component, the server component and the communication medium. Each main component has many sub-components. The novel aspect of the proposed model is the combination of the concepts from the two previous offloading frameworks or models. The first concept revolves around the cloudlet where a middle layer is deployed between a mobile device and the cloud infrastructure in order to reduce the distance of delegating tasks to a server at single hop distance referred to as a *cloudlet layer* by Satyanarayanan *et al.*, (2009).

There are a several reasons why the cloudlet is used instead of cloud. The main reason is the cost. If the mobile client has internet access and uses it, this would incur certain amount of cost and if the client is connected to a local server with an available Wi-Fi network, this would eliminate such cost.

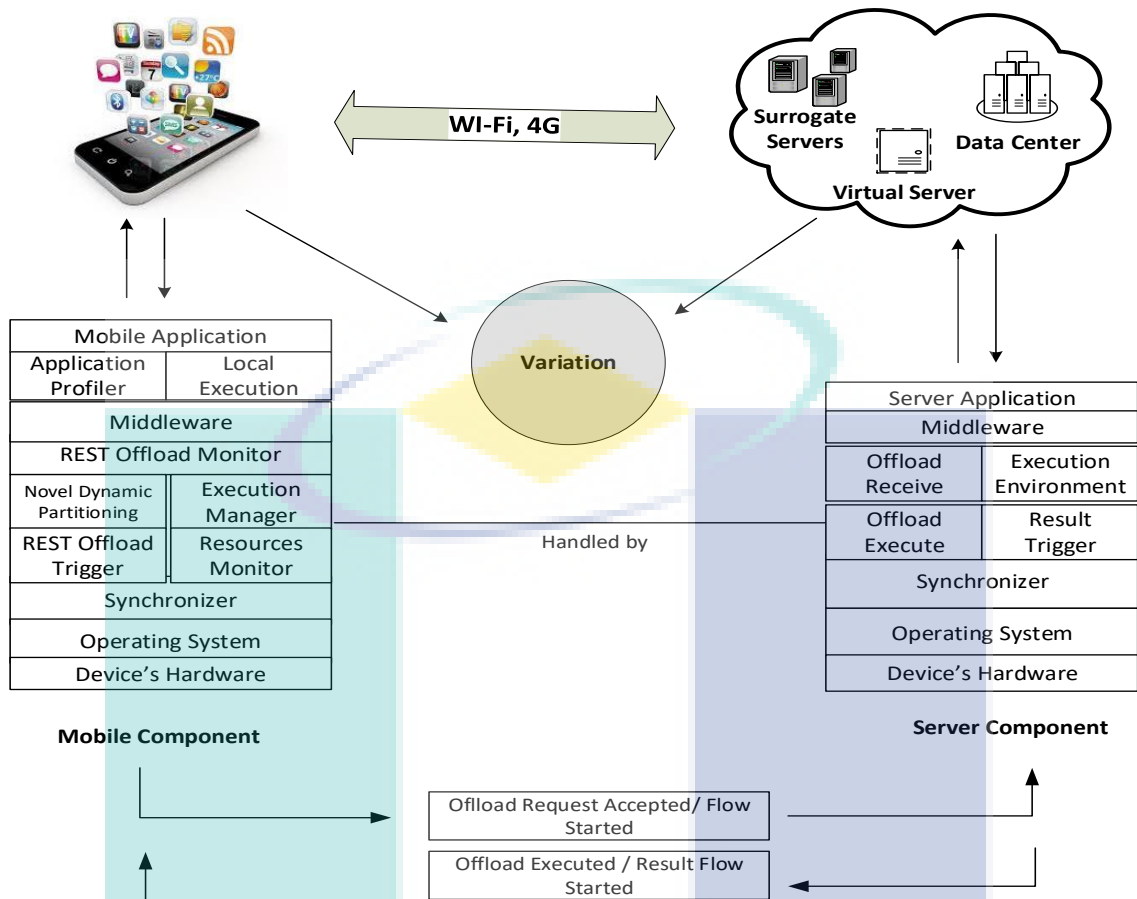


Figure 4.3 Model of REST-Offload

Latency is the second fundamental reason why the cloudlet is used. Despite connection of the network bandwidth to the internet has been rapidly increasing over the past few decades, the issue of latency has not been addressed equally rapidly and convincingly. In this regard, latency causes inappropriate cloud computing in certain cases such as in real-time processing, considering that response time is very crucial.

Bandwidth is the third most important reason. Although mobile internet speed is increasing with each coming year, it is still very slow compared to the speed of locally connected WLAN networks. This reduces the effectiveness of cloud computing in special cases such as in data intensive situations like speech recognition, natural language translation and image manipulation. Hence, the model of the proposed solution consists of these three main components: 1) Mobile Component; 2) Server Component; and, 3) Communication.

4.3.1 Mobile Component

Mobile component of the model is deployed to the mobile device. This component of the model is capable of processing application. The application processing would first start in the mobile device if the complexity of application is less than the value of predefined conditions (Execution Time, Battery Level and Available Network). Mobile components usually consist of the following sub-components.

4.3.1.1 Application Profiler

Application profiler is the essential component of any offloading system which drives the automatic evaluation of resources during offloading. Computational offloading models employ various kinds of application profiler. This is because the kind of application profiler to be used depends on its function, for instance, dynamically evaluating the availability of resources such as RAM, battery level, network availability and types of networks. Therefore, for evaluating different kinds of information, different profilers were used. The application profiling components in the proposed model was to evaluate maximum resources before taking an offloading decision. These application profiling components determine the feasibility of application partitioning and tasks offloading. Based on the information collected by the application profiler, a decision would be made either to execute task locally, offload the task or terminate the task. In REST–Offload model, the application profiler operates along with the execution manager to dynamically switch between local execution and remote execution based on the evaluated information regarding the battery, the network and the execution time.

4.3.1.2 Local Execution

Mobile application component of the proposed model consists of local execution and offloading. Local execution component encompasses the complete executable codes and could process the application completely. Local execution of the application is always the first priority to try during processing. This is due to the variation in the communication bandwidth and the uncertainty of available surrogate servers which would sometimes fail to provide efficiency in terms of power and performance. Additionally, the offloading of application parts, which are easy to process locally, would put transmission overhead.

This would ultimately cause loss of power and loss of performance. The offloading of executable codes would be beneficial only if the energy needed to consume during offloading is less than the energy consumption of the mobile device to process tasks using its local processing resources.

4.3.1.3 Application Partitioning

The first step before offloading is application partitioning. It divides the application into non-offloadable (local) and offloadable components. *Non-offloadable* means that the components are to be held in the mobile device for local execution while *offloadable* means that the components migrate to server for execution.

Application can be partitioned either dynamically or statically. In the dynamic partition of the application, one additional component, that is, the inference engine, needs to be installed for inferencing the offloading decision based on the contextual information collected by the application, the mobile device, the remote server and the network load (Shiraz *et al.*, 2013; Akherfi *et al.*, 2016). The inference engine would also need to read loads of the mobile device's resources (i.e., memory, and CPU) and available battery level. All this information could be collected either from the sensors or the previous execution pattern. The inference engine then evaluates the information and takes a precise decision. Compared to static partitioning, dynamic partition takes more precise decision at runtime to avoid offloading of unnecessary component. However, due to the involvement of inferencing component sensors and the continuous changes of execution pattern due to dynamic changes, dynamic partition utilizes more resources of the mobile device thus rendering it a resources intensive approach (Shiraz *et al.*, 2014).

In order to counter the intensiveness of dynamic partitioning of the application, the static partitioning approach is used. Nevertheless, both the approaches have their pros and cons. In case of the static partitioning, the additional computation is eliminated in the mobile device by eliminating the inference engine and the sensors. The programmers would annotate intensive components through a special Application Programming Interface (API) as an offloadable component (Akherfi *et al.*, 2016). In the static partitioning approach, the decision of the partitioning is made at compile time (design time).

4.3.1.4 Novel Dynamic Partitioning in Source Code

In order to avoid the complexity in terms of resources utilization and execution time of previously used partitioning techniques, the partition of application at source code was proposed in REST–Offload. Partitioning of application can be either be static or dynamic. Both types of technique would have some intrinsic limitations. Static partitioning takes place at compile time through manual annotation of intensive methods. This would reduce the computations “C”; however, it would not be able to cope with the changes occurring in the execution environment due to the mobility of the users moving from one location to another.

By contrast, the dynamic partitioning automates the partitioning of the applications. This type of technique would be able to cope every dynamic change of the execution environment; however, the pattern of execution would change each time with every new change in the network. Thus, the computation in the mobile device would increase and hence, would be resource intensive.

The proposed novel dynamic technique carries some static features, along with its ability to cope with the dynamic changes in the execution environment. To cope with such dynamic changes, an algorithm was deployed for the selection of execution parameters prior to offloading. Hence, by combining the positive features of both the static and the dynamic approaches, the novel approach would be able to overcome the lacks of both static and dynamic approaches. Therefore, the novel approach would be more effective in reducing the RTT and energy consumption. The novel dynamic technique proposed based on the analysis conducted for both the existing application partitioning techniques.

Moreover, the heavy methods are statically annotated as the remote methods at compile time by using the symbol @Web Method. This component is the sub-component of the Offload Monitors and is responsible for dividing the application into offloadable and local executable tasks. The component also distinguishes the offloadable methods from the local methods at compile time. Thus, the complexity of the dynamic partitioning and the static partitioning would be reduced as special APIs are used. All the local

executable tasks are then processed locally while all the offloadable tasks are transferred to the server for further processing.

4.3.1.5 Execution Manager

This component consists of the complete execution environment. It controls and monitors the whole environment of the application process. It scrutinizes and utilizes the whole system thus making the execution of the application possible and smooth. If the user was in the proximity of the surrogate servers, it would monitor and provide connectivity as an option to the mobile user. Execution manager also monitors to exhaust the option of local processing first. If the application processing at local device is complex enough, then the predefined condition would be read and the resources for offloading would be monitored. If during remote execution the network was interrupted or services were no more available, then execution would be switched from remote execution to local execution. Finally, execution manager would stay active for collecting the processed result from the remote server and terminate the process.

4.3.1.6 Offload Trigger and Result Trigger

The Offload Trigger component is responsible for triggering the offload process once the partitioning of application is completed. Normally, the SOAP offload trigger had been used in past studies. SOAP offload supports the XML file to trigger with offload for carrying data from the client device to the remote servers. With the analysis of SOAP testing, as a carrier protocol, it has been observed that SOAP is heavy to execute and complex to parse. Although SOAP- and XML-based offloading is considered as more secure compared to the RPC-, RIC- and REST-based offloading, communication data would be increased. Hence, this would increase computations in the mobile device.

Therefore, to reduce the size of data during communication at both sides between the mobile device and the remote server, a new lightweight offloading technique referred to as the REST-offloading technique was proposed, based on the analysis carried out involving many different offloading protocols. This new technique combined REST, JSON and WSDL to perform offloading. Normally, XML and JSON had been used as solutions in many different problems in past studies. In the present study, a technique

based on the combination of REST, JSON and WSDL was proposed to reduce complexity, computations and size of communication data. The intention of using REST and JSON in this study was influenced by the low computing ability of the mobile device. This technique would reduce the size of communication data and eliminate the computations as well as RTT. Furthermore, REST is simple to write because of HTTP and some CRUD (Create, Read, Update, and Delete) operations. Nevertheless, REST is also less secure as compared to SOAP. This is because REST has inherited the security from the underlying transport while SOAP defines its own WS-Security (Web services Security).

4.3.1.7 REST–Offload Trigger and REST Result Trigger

A REST–Offload Trigger component was added in the REST–Offload model. This component is the sub-component of Offload Monitor and is responsible for transferring the control of execution of the intensive tasks identified by the partitioning part to the remote servers. The REST–offload receives the component at the server side and is responsible for receiving the incoming intensive tasks which would further transfer the tasks to Offload Execute Component. After executing the task, the REST Result Trigger component was activated and the result was sent back to the client device. Deploying REST–Offload Trigger and REST Result receiver would not only execute the offloading processes but also provide a lightweight medium of carrying data from the device to the remote server. REST eliminates the heavy XML exchanging files between the client device and the remote server (i.e., the surrogate at one hop distance) which would ultimately reduce the size of communication data.

4.3.1.8 Resources and Offload Monitor

This component controls and monitors the offloading process from the start of establishing a connection until the termination of the connection. It checks the availability of the network and a surrogate willing to share its resources. Then, it establishes a connection and allow the REST–Offload Trigger to transfer the computational intensive task to the remote servers.

4.3.1.9 Middleware

Remote execution of mobile application in REST–Offload model needs a middleware, that is, the sources bridging the client device and the remote services. It provides access to remote services in order to execute intensive tasks delegated by the mobile device. Middleware would keep hiding the complications between the client device and its remote counterpart. REST–Offload makes a transparent remote execution available in the environment through middleware and allows the client device approaching the services using REST as a carrier protocol. Web Services Description Language (WSDL) was used as a middleware to advertise the remote services. WSDL supports XML format, that is, a hardware and software independent tool used to store and transport data between network devices.

4.3.1.10 Synchronizer

The synchronizer component keeps the client side and the server side parts of the application intact. It also keeps the sending and receiving of offloadable data over the bandwidth synchronous to avoid missing and interrupting any sequence of data. The basic role of the synchronizer component is to act as a coordinator between the client device and the server. It also monitors the communication overhead and ensure the smooth transmission and reception of communication bandwidth.

4.3.2 Server Component

Server component of the model consists of a physical server (surrogate) configured with glassfish server. The server component of the model was deployed on configured glassfish server to entertain all the computational intensive tasks received from the mobile device. The server component also consists of sub-components similarly identical to the sub-components of the mobile component. Hence, the description will not be repeated in this section. The REST–Offload model was configured with a surrogate server.

4.3.2.1 Surrogate Server

The geographical distance between the mobile device user and the cloud service provider plays a vital role to measure the latency rate. Previously, cloud servers had been used for the execution of the computational intensive tasks. In order to reduce long run RTT, cloudlet was used to bring the distant cloud to a single hop distance. In the proposed REST–Offload model, a surrogate server was configured to be connected at a single hop to the access point. This would reduce the distance, which in turn would reduce RTT as well as lead to the PC or the laptop in the working environment becoming a powerful remote execution machine for mobile applications.

4.3.3 Communication

The third component of proposed model is the communication medium. The role of communication is as important as the role of high speed CPU required to be at the remote execution environment. If the bandwidth of the available communication medium was not good, the uploading and downloading process would consume more time and it would ultimately affect the battery (Kumar *et al.*, 2011). Therefore, it is essential to consider the type of communication medium in order to ensure that the uploading and downloading process would be fast during offloading.

4.4 Operational Logic

This section consists of operational logic of the REST–Offload model, operational logic of the execution of applications and the proposed algorithm deployed for the consideration of pre-defined parameters.

4.4.1 Operational Logic of REST–Offload Model

Figure 4.4 shows the operation of different components of the REST–Offload Model. Mobile application was deployed for execution on the mobile device. The application is capable of executing a task locally as well as at the remote server node. During execution, the application profiler would continuously monitor resources such as

battery level, execution time of a task and network bandwidth of the mobile device and would then profile the resources. After completion of the profiling resources, the first attempt of any mobile application would be to execute the task locally in order to eliminate the remote execution time and avoid resource consumption in remote processing.

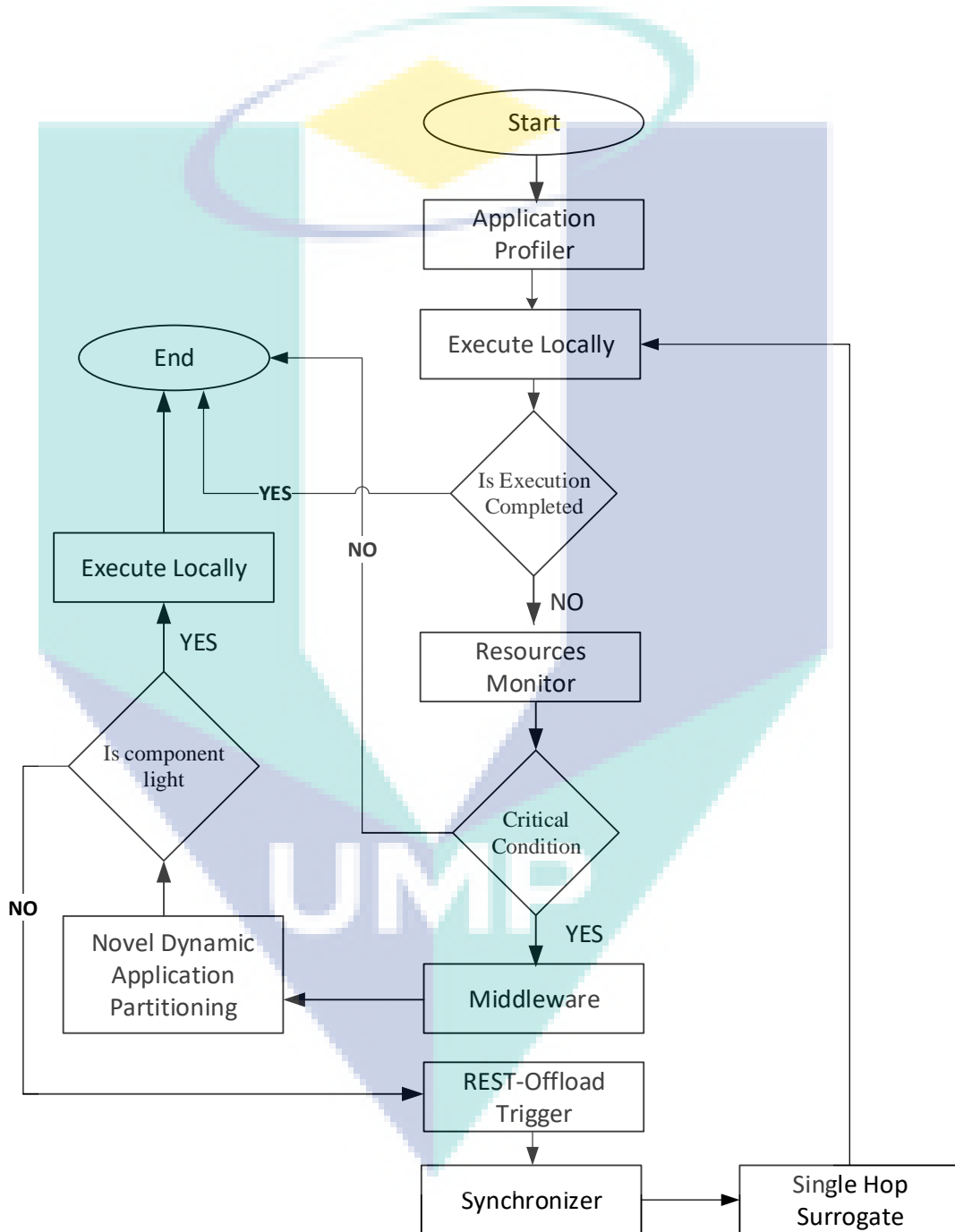


Figure 4.4 Operational Logic of REST-Offload Model

If the application was already complex enough, or if it was not be possible to execute at local device, then the predefined conditions for remote execution would be checked. If the conditions have been met, then resources monitoring for remote processing would start to search for the availability of a remote server (i.e., a surrogate) and then establish a connection to the remote server. If the conditions have not been met, then local processing would be checked again. If the local execution was not possible, then execution would be turned back to the first step of mobile application and subsequently stop further processing. If the condition has been satisfactory based on the defined values, then resources monitoring would take place and the middleware component would be activated to check the available services. Similarly, it would allow the application component to employ static partition application into lightweight parts and computational intensive parts where both will start execution concurrently.

If a component of a mobile application is lightweight, then component would be proceeded for local processing. After completion of the local processing the result would be sent back to the mobile application. Otherwise, if a component is not lightweight, then it would be passed to the REST–Offload Trigger component, which would then activate the synchronizer and offload the components (i.e., methods) to the surrogate for execution. At the completion of the remote execution of the application, the results are triggered back to the mobile execution manager components. This would consequently lead to the combination of both the result of the local execution and that of the remote execution. Finally, the combined results would be displayed.

4.4.2 Application Execution Flow in REST–Offload Model

The flowchart in Figure 4.5 shows the details of the interaction of the components in the proposed lightweight method level model in leveraging the services of the remote server node. The Client side application of the model is executed on the mobile device while the server side application is deployed to the remote server node. Whenever the Client side application needs the services of the surrogate server, it would activate the offload trigger and delegate the computational intensive task denoted as heavy method to the server.

In the online scenario, the client application utilizes the services provided by the remote server while in the offline scenario, the client application is capable of using its own local resources to process the task. The distinct procedure in the proposed solution was adopted to avoid unnecessary load in the runtime prediction and extra management of resources by using VM migration or whole application delegation. In this regard, the application partitioning is reduced to a simple two-step process. A novel dynamic approach used here, which will statically partition the application at method level and denote heavy method as offloadable method and delegate it to the counterpart at runtime. Clearly, in the proposed algorithm, all the predefined parameters would be firmly checked before complex tasks are offloaded to the remote server.

However, remote task execution while being connected to the 3G network was excluded. This is because 3G, which is a low bandwidth network, would fluctuate according to the mobility of the mobile application user from one location to another. Using an unreliable network for establishing connection to the remote server would yield inefficient results. The battery level should also be checked before starting the offloading task. A battery operating at critical level during offloading task to the remote server could turn the device off and would subsequently discard the offloading task. The most important parameter would be to predict the complexity of the task or the method to offload. A predictive algorithm would load on the limited available resources of the mobile device.

The logo for UIMP (University of Management and Applied Sciences) is a large, stylized letter 'V' shape. The left side of the 'V' is light blue, and the right side is light green. The letters 'UIMP' are written in white, bold, sans-serif font across the center of the 'V'.

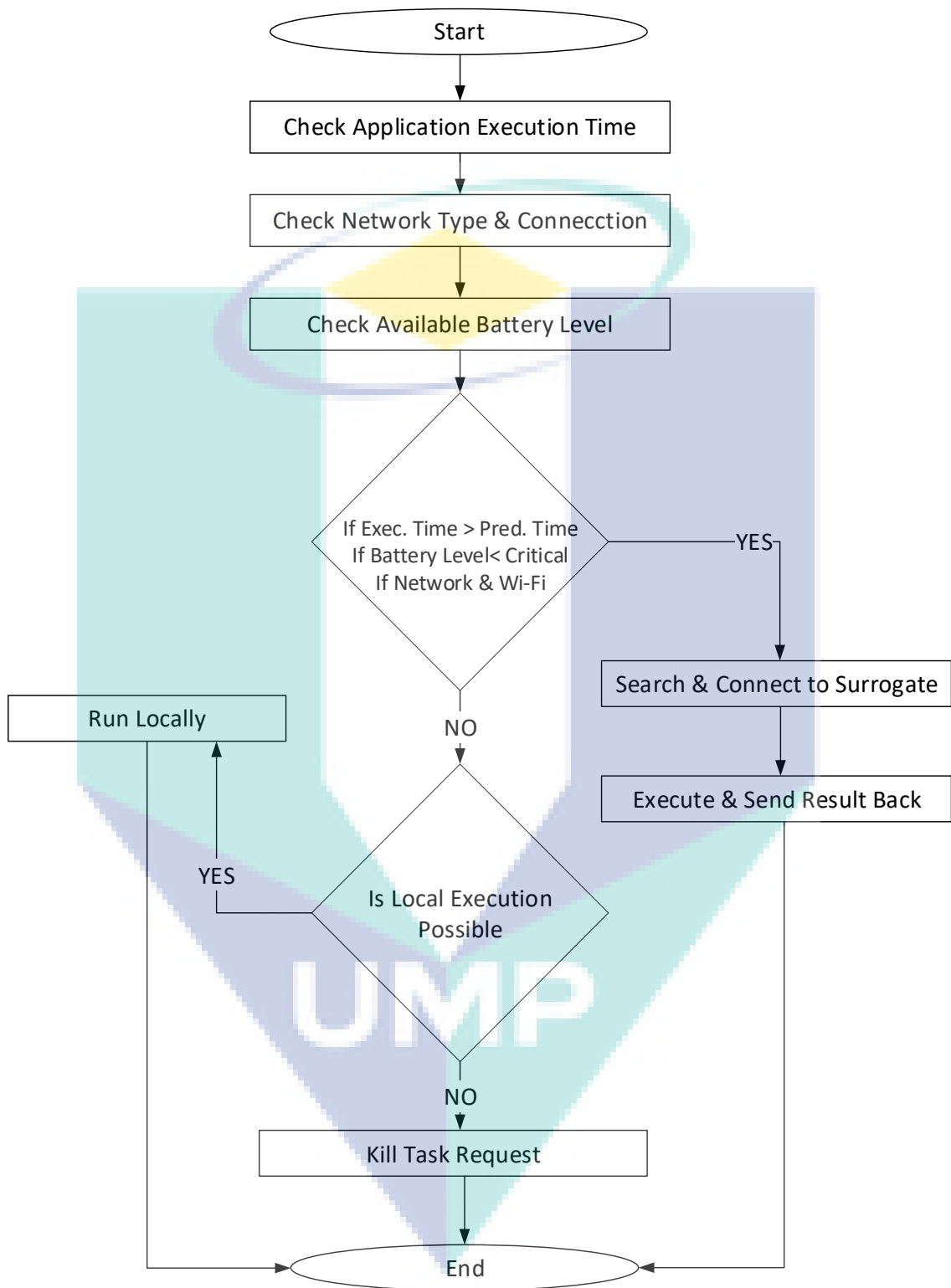


Figure 4.5 Execution Flow of the Mobile Application in Proposed Model

Additionally, an algorithm was proposed to encompass all the predefined parameters identified after a thorough investigation of power consumption of a modern

mobile device. The thorough investigation of the power consumption was conducted for both the application and the system components in order to identify the power draining factors. These factors are important to include which possibly alters the offloading results. For instance, checking the network type and network bandwidth would thus eliminate the issues of limited bandwidth and size of data to be offloaded. The proposed algorithm would also consider an offloading decision, that is, whether to offload the task to the local cloud (cloudlet) or to execute the task using the mobile device's resources. In short, the proposed algorithm would encompass the user preferences, applications requirement and maximum predefined parameters for a precise offloading decision. Application requirements would include the nature of the application such as real-time, fidelity adaptive and intensive parts in terms of computation and communication. User predefined parameters would comprise reliability and secure offloading while predefined parameters would include the existing battery level, the type of network available, the execution time and most importantly, the available network bandwidth.

The proposed lightweight method level model was first designed with the deployment of supplementary components to address the intensive partitioning issues of the traditional offloading methods and to counter the heavy parsing issues of previously developed frameworks and models. Further, the proposed method level computational offloading model has been inspired by the concept which had once been used in code transformation method by Rim *et al.*, (2006) who had partitioned the application at method level and offloaded the heavy parts for remote execution. This had reduced the size of communication data and led to a reduction of RTT. However, the code transformation technique used by Rim *et al.*, (2006) in mobile devices for reducing the code size via Distributed Execution Transformer (*DiET*) was a heavy process which caused computational overhead.

The *DiET* generator of slim code is a supplementary computational load on the mobile device with each offload. Hence, in designing the proposed model, instead of using the slim code concept or sending whole executable codes directly to the server, the present study aimed to deploy Representational State Transfer (REST), which is an architectural style of World Wide Web that communicates over HTTP protocol. REST was first developed by Roy Thomas Fielding (Fielding, 2000) and was initially developed to re-structure the Web Applications as none of the past researchers had used REST as a

carrier protocol in mobile computing for offloading purpose. Subsequently, the present research aimed to eliminate the need to generate the slim code for reducing the size of communication data by using REST as a carrier protocol because of its advantage of being lightweight compared to *DiET*, SOAP and RPC (Giogrio *et al.*, 2010).

In addition, using REST protocols instead of SOAP for communication carriers would eliminate XML file transfer. This would support the JSON format, thus eliminating the extra baggage of data to delegate. As a result, the transmission data “D” would be reduced, which would then reduce the Turnaround Time. This would ultimately save battery. Normally in past research, Simple Object Access Protocol (SOAP) had been used to establish client service connection for transmission of data between Web Applications. The reasons for using REST instead of SOAP are due to the portability, simplicity and also the lightweight nature of REST.

Furthermore, to reduce the delegation of computational intensive tasks to the cloud server residing at multi-hop distance, a middle layer solution had been proposed first by Satyanarayanan *et al.*, (2009) and later used by Magurawalage *et al.*, (2014). Offloading any computational intensive task directly to a distant cloud would always result in a long run RTT which is resource intensive. In order to reduce the distance, Satyanarayanan *et al.*, (2009) had used the concept of Cloudlet, which is a small cloud in the nearest computing environment. Hence, the present research used a similar concept. A further addition of a single hop was made in the present study where a PC would be connected to IEEE 802.11 access point as a cloudlet layer in between the client device and the cloud infrastructure. This cloudlet layer would serve as a confined service in a fast approach for the client device to offload.

4.4.3 Proposed Algorithm for the Selection of Pre-defined Parameters

It is an established fact that, offloading in any circumstances, is always not energy-efficient and involves burden of calculations over resource constrained devices. Chapter 2 of this thesis has discussed details of when and how to offload. The focus of the proposed model is to avoid any further load in terms of computations over mobile device. A predefined time slot used by deployed an algorithm for completing execution of a task at local scenario is shown in Figure 4.6. While processing any task locally, if the

predefined time slot was knocked out, the control would be passed to the remote server with a message to compute the specific heavy method at the server node.

This algorithm would guide the Offload Monitor to verify all the predefined conditions and ensure that the right decision of offloading would be made. The flow of execution of the offloading task is given in the flowchart in Figure 4.6. The basic steps of the proposed algorithms are given in algorithm 1 as follows:

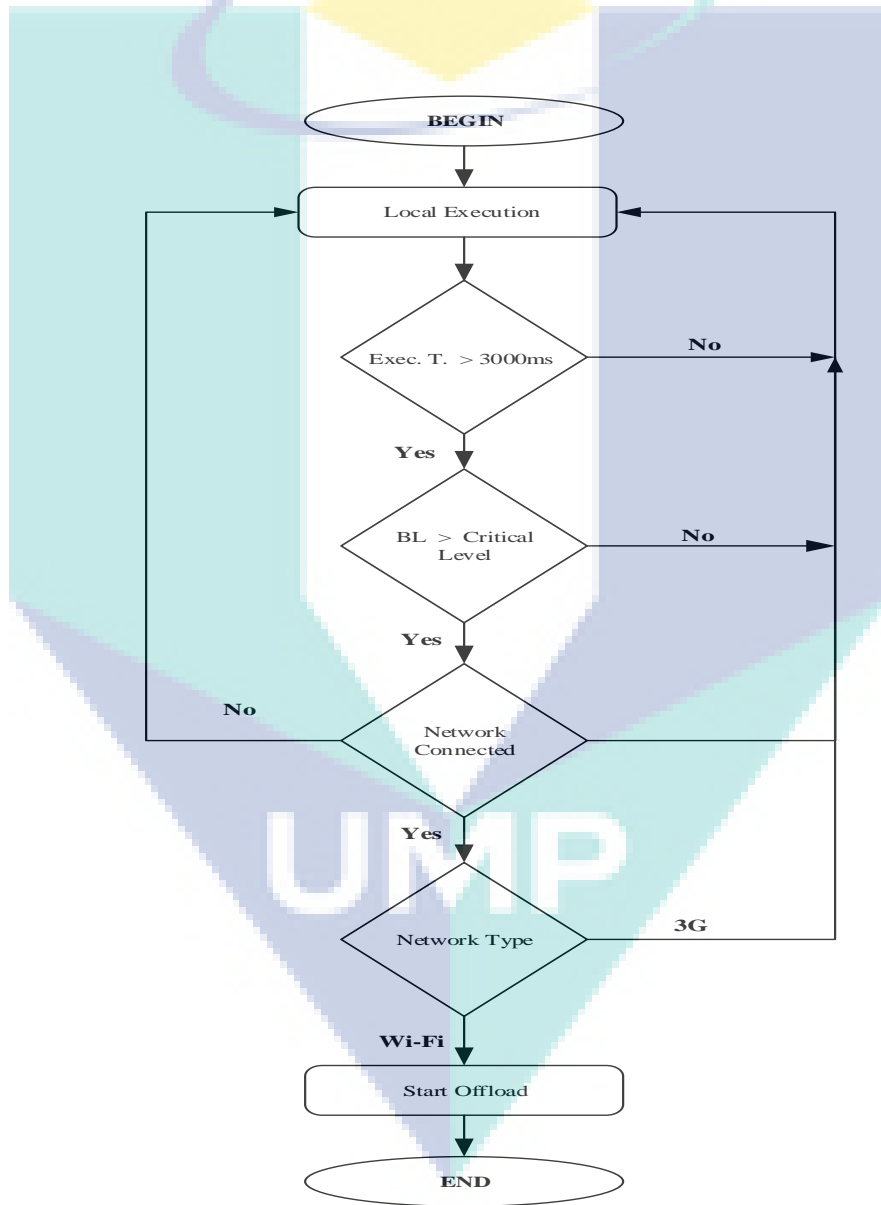


Figure 4.6 Flow of the Selection of Predefined Parameters

Step 1. Attempt local execution first. If the execution time exceeds the predefined threshold value, then go to *Step 2*.

Step 2. Check all the predefined parameters such as battery level, network connection, network type and execution time.

Algorithm 1: Computational Offloading Service

```

1:  procedure OFFLOADINGSERVICE ( $E_T, B_L, N_T, N_S$ )
    $E_T$ -Execution Time,  $B_L$ -Battery Level,  $N_T$ - Network Type,  $N_S$ - Network Status
2:  Read:  $E_T, B_L, N_T, N_S$ ;
3:  Run: Local-execute();
4:  if  $E_T >$  Threshold Value then;           // Predefined Parameter 2 seconds
5:    trigger offload monitor;
6:  end if
7:  if  $B_L <$  critical Value then             // Predefined parameter of critical battery
8:    write: " battery critical";
9:    local-execute();
10:   end;
11:  else if
12:     $N_S ==$  connected then
13:    check network type:
14:    if  $N_T ==$  Wi-Fi or 4G then           // 3G excluded because of limited bandwidth
15:      Trigger-offload();
16:    else
17:      Local-execute();
18:    end if;
19:  else
20:    Kill-request ();
21:  end if;

```

Step 3. If all the predefined parameters meet, then trigger-offload is activated to search the remote server.

- Establish a connection to the remote server and start offload,
- Return result back to handheld device.

Step 4. If any of the above defined conditions is false, then resume local execution and end.

Step 5. If the local execution is not possible, then kill the request and end.

The response time of an application varies from application to application. For all the real-time systems such as GPS, natural language translators and online games, the

response should always be crispy. In the proposed algorithm, an execution deadline has been defined with a threshold value which would be entertained for the local execution to attempt first. If the deadline threshold value has been met and execution has not yet been completed, then Execution Monitor would be activated and the offloading process would take place. The threshold value in the proposed algorithm would be 3000 ms (3s) which is considered as the lowest response time (Ferreira *et al.*, 2011; Izaki, 2000). The threshold value would vary for different applications and could be selected solely based on the complexity of application and available resources. Hence, 3000 ms was selected as the threshold value for testing purpose through the developed prototype application.

The prototype application was developed to test a single component of an intensive task; therefore, the threshold value was kept minimum. Moreover, the real-time systems would need to have a spontaneous response whereas the response time for applications such as editors could be up to 2000–3000 ms. Furthermore, the critical level of battery would also vary for different manufacturers mobile devices as well as researchers. For example, the critical battery level for the *iPhone* is about 6%. For *iPhone*, once the mobile device drains the whole battery, it would not turn on until the recharge percentage of battery level reaches up to 6%. Similarly, for all *Samsung* devices, the critical battery level starts from 5%. In the present research, 3% was considered the critical level because the percentage would vary between 1 and 6% for different mobile devices. Offloading in such a critical level would most likely result in the mobile device turned off thus discarding the whole efforts during offloading. In both cases, if the predefined parameters have not been satisfactorily met, offloading tasks to the remote servers should be avoided and local execution should be attempted. In any case in which the local execution would not be possible, the request should be turned down.

4.5 Evaluation of the Proposed Model

The significance of the proposed model was evaluated by simulating it in real mobile cloud computing environment. A prototype application for Android mobile devices was developed in order to test different computational intensities. The execution performance of application was conducted with reference to the execution time of the application (i.e., response time) and power consumption during the execution. All the empirical results were obtained from simulations using 30 different intensities with a

sample space of 20. The experiments conducted with *Device under Tests* (DUTs) from two different vendors, namely, the *Samsung Galaxy A5* and the *ASUS Zenfone 5*. To validate the resultant values, a sample mean was calculated for the sample data and for further measurements of error estimation, the standard deviations were used. With 99% confidence interval, all the empirical results were then compared to the benchmarking data collected from the local execution of mobile device as well as the findings of previous studies.

Moreover, the findings of a study conducted by Shiraz *et al.*, (2014) have also been considered as a second benchmark for the proposed lightweight method level offloading. Both comparisons have indicated the significance of the proposed solution in the present research. The following sections will elaborate the experimental setup, the tool used in these experiments, the techniques for data collection and statistical techniques employed in the analysis of the collected data in order to obtain meaningful results.

4.5.1 Experimental Setup

The experimental emulation environment in the present study was set up as illustrated in Figure 4.7. The setup composed of the remote surrogate machine and mobile client devices. The surrogate machine used 32-bit *Microsoft Windows 10 Professional* operating system (OS) with Intel® Core™ i7-3770 Processor @ 3.40 GHz speed and 4.0 GB RAM capacity. A D-Link wireless Wi-Fi modem/ Access Point with a physical layer data rates of 54 Mbps was used to connect the remote server machine with the mobile devices.

To test the developed prototype applications, the experiment setup was developed in Wi-Fi wireless network of radio type 802.11g, the surrogate machine and two different brands of Client mobile devices, namely, the *Samsung Galaxy A5* and the *ASUS Zenfone 5*. The *Samsung Galaxy A5* was operating using the Android 5.0.2 (Lollipop) OS with Quad Core Cotex-A53 Processor @ 1.2 GHz and 16 GB memory on 2300 mAh battery. Meanwhile, the *ASUS Zensfone 5* was running on the same Android v5.0.2 (Lollipop) OS with 1.2 GHz Processor and 2 GB RAM but Dual Core instead of Quad Core and only 8 GB instead of 16 GB memory as well as slightly less 2110 mAh battery capacity. Mobile

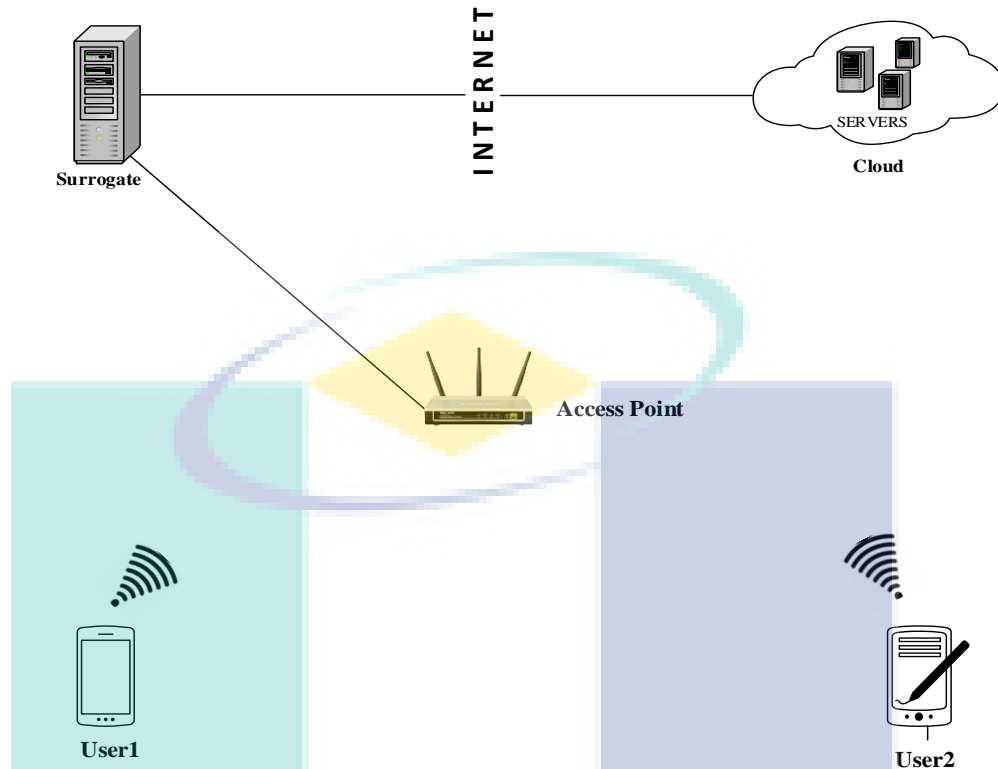


Figure 4.7 The Environment of Experimental Offloading Scenario

device accessed the wireless network via Wi-Fi wireless network connection of radio type 802.11g, with the available physical layer data rates of 54 Mbps.

Two components of the proposed model were developed, namely, one for the Client side that was used in the mobile device, and the other for the surrogate servers. The surrogate machine was configured for the provisioning of services utilizing Software as a Service (SaaS) and Infrastructure as a Service (IaaS) models of cloud computing. The Server side application was deployed by configuring a Glassfish server at the surrogate using Web Services Application tools in Eclipse. Android Developers (Java-based Android Software development tool kit) was deployed for the development of the Client side application in IntelliJ IDE 15 Community environment. Android Debug Bridge (ADB) plugin was used to develop the prototype Android application.

The traditional computational offloading models had used KSOAP libraries in prototype development which is an XML-based messaging protocol. SOAP is conceptually more difficult and more heavyweight as compared to REST. The aim of the

current work was to develop a lightweight Android application model which could be easily managed and should reduce the heavy processing steps during migration of data. SOAP-based applications would be harder to develop and configure as well as heavy in size. SOAP is the older form of providing communication environment to client devices which communicate with its remote server counterparts.

By contrast, REST is an architectural style used to design a system which solves the common issues occurring during communication which was developed by Roy Thomas Fielding in his PhD research (Fielding, 2000). The aim of implementing REST in the present study was to ensure fast performance, reliability, lightweight and extensibility. REST was used in the development of the prototype application to access the preconfigured services deployed to the configured Glassfish server. To monitor the execution time/ turnaround time, a stopwatch Time Left was used.

Meanwhile, for energy consumption of different computational intensities, Power Tutor, which is a power estimating tool, was used in the first attempt. Power tutor is a built-in power estimating tool available at Android *Play Store*. Power Tutor acquires the battery discharge curve for each discrete component of the mobile device using built-in battery voltage sensor. It determines the energy consumption state of each component and application. Additionally, it also performs a regression to obtain the power mode. However, as it was designed initially for a specific model (brand) of a mobile device, this power estimating tool would not always be profiling the accurate energy consumption of every mobile device. Therefore, Monsoon Power Monitoring Tool, which is a hardware power estimating device, was used. This device provides a strong power estimating solution for any kinds of mobile device powered by Lithium-ion or Li-ion batteries at 4.8 volts or lower.

Figure 4.8 is a screenshot of the graph of power consumption measured by the Monsoon Power Monitor application tool. Most of the Android developers have failed to consider the limited battery timing of the mobile devices during the designing process of the application. Consequently, mobile applications have consumed enormous amount of power and would drain the limited battery capacity in just a few hours. To measure the power consumption of each individual application, numerous software estimating tools have been used but all the tools would run on the devices and collect power data and

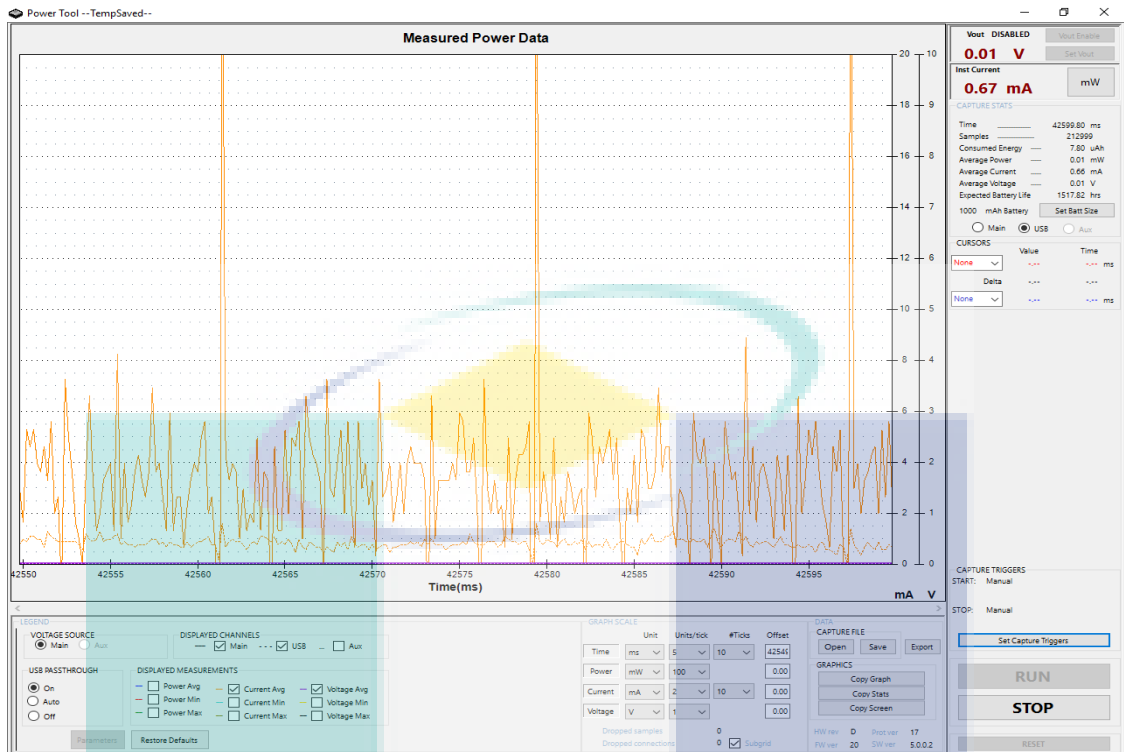
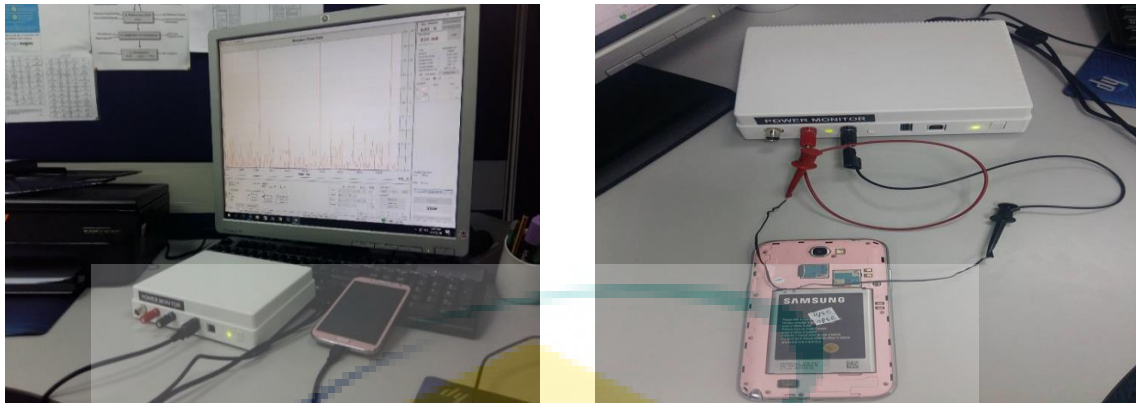


Figure 4.8 A Screenshot of the Monsoon Power Monitor Application Tool

then store it locally. This would ultimately use CPU cycles and result in inflating power reading. The most popular devices such as the *Samsung Galaxy* and *Nokia* smartphones could not report the power consumption of an application accurately even with the help of software estimating tools. An off-target hardware tool would be needed to measure the power consumption of such devices.

Therefore, the Monsoon Power Monitor application was used to help Android developers create mobile applications with better battery life. The Monsoon Power Monitor application provides the popular off-target power consumption estimation. It also is capable of measuring the current, voltage and power and then connected to a special Monsoon Power Application (Computer Software) which gives control over power data and collect and display the data in the form of a graph.

Figure 4.9 gives the experimental scenario of Monsoon Power Monitor for power consumption used in the laboratory environment. The power monitor device was connected with a PC using the backside USB port. The mobile device could either be connected to the Auxiliary port, Main port or USB out port on the front side. In the



(A)

(B)

Figure 4.9 Use of the Monsoon Power Monitor in the Experiments for Power Consumption Readings

experiments, two mobile devices, namely, the *Samsung Galaxy A5* and the *ASUS Zenfone 5* were used to estimate power consumption. Both mobile devices were connected to the USB port on the front side because the battery was not removable. Therefore, the only way to connect was via the USB port as shown Figure 4.9 (A). In the case of the *ASUS Zenfone 5*, the main port was used as the battery had been removed and a connection was provided to the main port of the Power Monitor as shown in Figure 4.9 (B). Each time when the prototype application ran locally or remotely while mobile device was connected to the power monitor device, the readings of consumption were recorded.

4.5.2 Prototype Applications

The proposed model was implemented by developing a prototype Android application. The prototype application composed of two components, namely, the service provider called REST–Offload Service and the service consumer called Android-Local. The service provider component was installed and configured at the remote server while the service consumer component was deployed to the Client device (i.e., the mobile device). The prototype application was designed for computational intensity of generating two random matrices of type integer. Both the matrices were multiplied and then the resultant matrix was obtained. As matrix multiplication is a computational intensive task, hence it was selected to be implemented in the experiments. The matrix multiplication logic of the prototype application was tested for 30 different computational intensities by varying the matrices length between 160 x 160 and 450 x 450.

4.5.3 Data Collection and Data Processing

Three different types of testing scenarios were used in the experiments for data collection. In the first scenario, data were collected by testing the prototype application in the local mobile devices both on the *Samsung Galaxy A5* and the *ASUS Zenfone 5*. Meanwhile, in the second scenario, the traditional computational offloading logic was implemented and the computational intensive tasks were offloaded to the remote servers on both mobile devices. Finally, in the last scenario, the logic of the proposed lightweight computational offloading was implemented and the execution of task offloaded to the surrogate servers was tested. In all of the three scenarios, the aim was to analyse the prototype application for execution time or turnaround time and energy consumption of both mobile devices while executing the task.

The experiments considered a sample space of 20 values and for each scenario, the experiments were evaluated 20 times of each computational intensity. In this regard, the sample size (i.e., $n = 20$) and a sample mean (i.e., \bar{x} of $n = 20$) were calculated for each computational intensity. Standard deviation (SD), which showed the variation in the execution time as well as in energy consumption while running each intensity, was calculated. The Central Limit Theorem states that as the sample size increases, the sampling distribution of the sampling means approaches to a normal distribution. Therefore, the sample, $n = 20$, was considered and about 99% of the sample means fell within 2.58 of the standard deviation of the population mean. Hence, the confidence interval calculated showed the range of the sample mean of 20 values in each experiment with 99% confidence. The prototype application developed for mobile devices composed of a computational intensive component, that is, the matrices multiplication service. This was evaluated on the basis of two parameters execution time in milliseconds (ms) and energy consumption in Joules (J). The empirical data were gathered for each computational intensity. The intensities that ranged from 160 x 160 to 450 x 450 which increased with the increase size of 10 with each intensity, were calculated. The whole data were then tabulated. For all the three scenarios, the total Execution Time (T_{ET}) and total Energy consumption (E_T) were evaluated.

Meanwhile, the confidence interval was calculated for each sample intensity. Confidence interval is the range of values of sample statistic which is likely to contain

the value of an unknown population parameter. Data sampling may have sampling error that is, the sample mean calculated could differ from its population mean. Therefore, to signify the correctness of the calculated sample value, the interval estimate of each sample was calculated. A certain percentage of the sample values may contain the parameter of an unknown population. The percentage of such confidence interval having the population value is called the confidence level. Confidence interval is a range which estimates the true population value for a statistic. There was a margin of error, E , which existed and this showed the probability of not occurring mean value in the whole the population value. If E denotes the error estimation for 99% confidence interval, then it is calculated by the following equation:

$$E = 2.58 * (\sigma / \sqrt{n}) \quad 5.1$$

In Equation 5.1, σ denotes the standard deviation in the calculated sample values and n denotes the size of the sample space. The confidence interval for each sample mean of the collected sample data was calculated with 99% confidence interval by using the following equation:

$$\text{Confidence Interval} = \bar{x} \pm E \quad 5.2$$

The following section will explain data collected in experiments in all the three scenarios in real-time mobile computing environment. The data were collected and then manipulated in the first scenario such as by executing the application in the local mobile device. The results (i.e., Execution Time and Energy Consumption) were tabulated. In the second scenario, the application was delegated to the remote servers using traditional computational offloading techniques. In the third scenario, data were collected and tabulated using the proposed lightweight computational offloading method.

4.5.4 Data Collected by Executing Application in Local Mobile Devices

In this scenario, the developed prototype application was executed in the local mobile devices in order to assess the Execution Time or Turnaround Time in milliseconds (ms) and Energy Consumption in Joules (J). The prototype application was run with different computational intensities. As the matrices multiplication is a computational

intensive task, a prototype was developed to multiply two randomly generated matrices ranged between 160 x 160 and 450 x 450. Each computational intensity varied from the previous one with an addition of 10. The 30 different computational intensities were selected and each intensity was run 20 times for validation purpose. Table 4.2 and Table 4.3 show the sample execution time for lowest and highest intensities of the prototype application in the local mobile devices both for the *Samsung Galaxy A5* and the *ASUS Zenfone 5*, respectively.

Two matrices were randomly generated in the mobile devices and multiplied with each other. Then, the resultant matrix was displayed on the screen of each mobile device. The time consumed while generating the two random matrices, the time for processing the computational task using mobile resources and the time of the display of the resultant matrix on the screen of the mobile devices were all collectively referred to as the execution time.

As mentioned earlier, the prototype was run for 30 different computational intensities between 160 x 160 and 450 x 450 with the constant jump of 10 in each intensity. The statistical computation was done in the experiments in order to justify the result for whole population. In this regard, each intensity was iterated 20 times which showed the size of the sample. The mean turnaround time was calculated for each sample, which is the point estimator for whole population.

Table 4.2 Local Execution Time of Prototype Application for *Samsung Galaxy A5*

Matrix Size	Sample mean of ET of Local Execution (ms)	SD of ET	%RSD of ET	Confidence Interval
160x160	11071	346.53	3.13	11071(+/-)800
450x450	111799	4434.77	3.97	111799(+/-)10234

Table 4.3 Local Execution Time of Prototype Application for *ASUS Zenfone5*

Matrix Size	Sample mean of ET of Local Execution (ms)	SD of ET	%RSD of ET	Confidence Interval
160x160	13240	207.36	1.57	13240(+/-)478
450x450	118460	230.22	0.19	118460(+/-)531

In addition, the standard deviation was calculated to show the variation in the execution time while evaluating the experiments of each intensity that were conducted 20 times for each sample space. The percentage (%RSD) for each of the 30 computational intensities was calculated for each sample space which showed the percentage difference in the execution time of each experiment. The 99% confidence interval was calculated for the sample space of each intensity which showed the degree of uncertainty in the turnaround time for the whole population. Table 4.4 and Table 4.5 show the sample energy consumption for lowest and highest intensities of the prototype application in the local mobile devices both for the *Samsung Galaxy A5* and the *ASUS Zenfone 5*, respectively.

The prototype ran for 30 different computational intensities and the energy consumption costs of each time execution of discrete intensity were recorded. The same intensity was iterated for a sample size 20, which is a point estimator for the energy consumption of any size sample or whole population. The standard deviation calculated for each sample space showed the variation in energy consumption while %RSD calculated for each sample space showed the percentage difference in energy consumption of each experiment. The 99 % confidence interval was calculated for all computational intensities, which determined the degree of uncertainty in each sample space.

Table 4.4 Energy Consumption of Prototype Application for *Samsung Galaxy A5* through Local Execution

Matrix Size	Sample mean of EC of Local Execution (J)	SD of EC	%RSD of EC	Confidence Interval
160x160	4.58	0.249	5.44	4.58(+/-)1
450x450	45.4	0.158	0.35	45.4(+/-)0

Table 4.5 Energy Consumption of Prototype Application for *ASUS Zenfone 5* through Local Execution

Matrix Size	Sample mean of ET of Local Execution (J)	SD of EC	%RSD of EC	Confidence Interval
160x160	5.54	0.30	5.35	5.54(+/-)0
450x450	48.4	0.51	1.06	48.4(+/-)1

4.5.5 Data Collected by Offloading Application using Traditional Offloading Techniques

In this scenario, the developed prototype application was offloaded by traditional offloading techniques to the remote servers in order to assess the Execution Time or Turnaround Time in milliseconds (ms) and Energy Consumption in Joules (J). The prototype application ran with different computational intensities. As the matrices multiplication was a computational intensive task, a prototype was developed to multiply two randomly generated matrices that ranged between 160 x 160 and 450 x 450 in the mobile devices and sent to the remote servers for executing the multiplication task, which is a computed intensive task for mobile devices. The resultant matrix was sent back to the mobile device and displayed at the user interface. Each computational intensity varied from the previous one with an addition of 10. The prototype ran with total of 30 different computational intensities. For validation purposes, statistical analyses were performed on the data by selecting a sample of 20 values to verify the results for whole population.

Table 4.6 and Table 4.7 present the results of the sample total Turnaround Time or Execution Time of the prototype application offloaded for execution to the remote servers using traditional computational offloading techniques both for the *Samsung Galaxy A5* and the *ASUS Zenfone 5*, respectively. In the experiments, each of the computational intensity was evaluated 20 times and the mean execution time of sample size 20 was calculated.

Table 4.6 ET of Prototype Application Execution through Traditional Offloading for *Samsung Galaxy A5*

Matrix Size	Sample mean of ET of Local Exe. (ms)	SD of ET	%RSD of ET	Confidence Interval
160x160	9608	535.78	5.58	9608(+/-)1236
450x450	189523	2089.39	1.10	189523(+/-)4822

Table 4.7 ET of Prototype Application Execution through Traditional Offloading for *ASUS Zenfone 5*

Matrix Size	Sample mean of ET of Trad. Offloading (ms)	SD of ET	%RSD of ET	Confidence Interval
160x160	11280	164.32	1.46	11280(+/-)379
450x450	152520	238.75	0.16	152520(+/-)550

The sample mean is the point estimator for the whole population. The standard deviation (*SD*) calculated for each sample size showed the variation in execution time or turnaround time in the sample size of each intensity. The %RSD calculated for each sample showed the percentage difference in turnaround time or execution time of each experiment. The confidence interval 99% was calculated for each computational intensity which showed the degree of uncertainty in each sample size.

Table 4.8 and Table 4.9 show the sample energy consumption costs while processing the prototype application using the traditional computational offloading techniques for both the *Samsung Galaxy A5* and the *ASUS Zenfone 5*, respectively. In the experiments, each of the computational intensity was evaluated 20 times and the mean energy consumption of sample size 20 was calculated.

The sample mean is the point estimator for the whole population. The standard deviation *SD* calculated for each sample size, shows the variation of energy consumption in the sample size of each intensity. The %RSD calculated for each sample showed the percentage difference in energy consumption of each experiment. The 99% confidence interval was calculated for each computational intensity which showed the degree of uncertainty in each sample size.

Table 4.8 EC of Prototype Application Execution through Traditional Offloading for *Samsung Galaxy A5*

Matrix Size	Sample mean of EC of Trad. Offloading (J)	SD of EC	%RSD of EC	Confidence Interval
160x160	7.72	0.421	5.45	7.72(+/-)1
450x450	102.08	0.602	0.59	102.08(+/-)1

Table 4.9 EC of Prototype Application Execution through Traditional Offloading for *Asus Zenfone 5*

Matrix Size	Sample mean of EC of Trad. Offloading (J)	SD of EC	%RSD of EC	Confidence Interval
160x160	8.56	0.36	4.26	8.56(+/-)0
450x450	102.74	0.15	0.15	102.74(+/-)0

4.5.6 Data Collected by Offloading Application using REST–Offload Method

In this scenario, the developed prototype application was offloaded by the proposed Rest-Offload method to the remote servers in order to assess the Execution Time or Turnaround Time in milliseconds (ms) and Energy Consumption in Joules (J). The prototype application ran with different computational intensities. As the matrices multiplication was a computational intensive task, a prototype was developed to multiply two randomly generated matrices that ranged between 160 x 160 and 450 x 450 in the mobile devices and sent to the remote servers for executing the multiplication task which is a computed intensive task for mobile devices. The resultant matrix was then sent back to mobile devices and displayed at the user interface. Each computational intensity varied from the previous one with an addition of 10. The total 30 different computational intensities were run. For validation purposes, statistical analysis were performed on the data by selecting a sample of 20 values to verify the results of whole population data.

Similarly, Table 4.10 and Table 4.11 show the sample turnaround time of prototype application using the proposed lightweight computational offloading REST–Offload method. The time consumed while generating the two random matrices, the time for offloading the matrices to the surrogate servers, the time for processing the computational task at server and the time of to obtain the resultant matrix back on the screen of the mobile devices were all collectively referred to as the Turnaround time or Execution time.

Here, each intensity was calculated for the iterated five times which showed the size of the sample. The mean turnaround time was calculated for each sample, which is the point estimator for the whole population. The SD calculated showed the variation in the turnaround time while evaluating the experiment of each intensity 20 times of each sample space. The %RSD calculated for each sample showed the percentage difference in the turnaround time of each experiment. The 99% confidence interval was calculated for the sample space of each intensity which showed the degree of uncertainty in the turnaround time for the whole population.

Table 4.10 ET of Prototype Application Execution through REST–Offload Method using *Samsung Galaxy A5*

Matrix Size	Sample mean of ET of REST–Offload (ms)	SD of ET	%RSD of ET	Confidence Interval
160x160	7418	222.86	3.00	7418(+/-)514
450x450	49587	274.23	0.55	49587(+/-)633

Table 4.11 ET of Prototype Application Execution through REST–Offload Method using *ASUS Zenfone 5*

Matrix Size	Sample mean of ET of REST–Offload (ms)	SD of ET	%RSD of ET	Confidence Interval
160x160	7780	164.32	2.11	7780(+/-)379
450x450	61520	178.89	0.29	61520(+/-)412

Likewise, Table 4.12 and Table 4.13 show the sample energy consumption cost of prototype application using the proposed lightweight computational offloading REST–Offload method. The energy consumption during the generation of the two random matrices, the energy consumption during the offloading of the matrices to the surrogate servers, the energy consumption for processing the computational task at the server and the energy consumed during reception of the resultant matrix back on the screen of the mobile devices were all collectively referred to as the energy consumption cost. Each intensity was calculated for the iterated five times which showed the size of the sample.

Table 4.12 Energy Consumption Cost of Prototype Application Execution through REST–Offload using *Samsung Galaxy A5*

Matrix Size	Sample mean of EC of REST–Offload (J)	SD of EC	%RSD of EC	Confidence Interval
160x160	3.48	0.1	2.88	3.48(+/-)0
450x450	25.35	0.12	0.48	25.35(+/-)0

Table 4.13 Energy Consumption Cost of Prototype Application Execution through REST–Offload using *Asus Zenfone5*

Matrix Size	Sample mean of EC of REST–Offload (J)	SD of EC	%RSD of EC	Confidence Interval
160x160	4.64	0.29	6.21	4.64(+/-)0
450x450	28.32	0.30	1.07	28.32(+/-)0

The mean energy consumption cost was calculated for each sample, which is the point estimator for the whole population. The standard deviation (*SD*) calculated showed the variation in energy consumption while evaluating the experiment of each intensity 20 times of each sample space. The percentage (% RSD) calculated for each sample showed the percentage difference in the energy consumption cost of each experiment while 99 % confidence interval was calculated for the sample space of each intensity which shows the degree of uncertainty in energy consumption for the whole population.

4.6 Summary

The proposed new lightweight method level computational offloading model was designed in this Chapter. The proposed model was developed in the real mobile cloud environment. A prototype application with three components Local Execution (Android-Local), Traditional Offloading (Android-SOAP) and REST-Offload was developed. All three components were executed and the prototype application was evaluated in the real mobile cloud environment.

The experimental data were collected in three different scenarios. In addition, a computational offloading algorithm was proposed to avoid the unnecessary computational load over mobile devices. It also stopped the unwanted processing and uncertain computational offloading attempt. In other words, it simply reduced the computational load and management hurdles of resource limited mobile devices.

CHAPTER 5

RESULTS AND DISCUSSION

5.1 Overview

This chapter presents the analysis and discussion of experimental results collected in Chapter 4. The chapter is organized as follows. Section 5.2 presents the Local Execution experimental results in terms of Energy Consumption (J) and Execution Time (ms). Section 5.3 presents the results in terms of energy consumption (J) and execution time (ms) while processed the prototype application at remote servers by offloading through traditional computational offloading techniques. Section 5.4 presents the discussion of experimental results in terms of energy consumption (J) and execution time (ms) while executing the prototype application at surrogate server using the proposed lightweight REST-Offload method. Section 5.5 consist of execution time comparison of all three scenarios. Section 5.6 comprises energy consumption cost comparison of all three scenarios. Section 5.7 discusses results against benchmark in terms of Execution Time and Energy Consumption. Section 5.8 discusses comparison of different mobile devices for ET and EC. Section 5.9 gives efficiency comparison of ET and EC against others. Section 5.10 provides comparative analysis. Section 5.11 discusses threats to validity while section 5.12 summarises the chapter.

5.2 Analysis of Application Execution at Local Mobile Device

The prototype application executed at local mobile devices both on Samsung Galaxy A5 and ASUS Zenfone5, in order to estimate the ET in milliseconds (ms) and EC in Joules (J). The experimental results of both the devices presented in Table 4.2, 4.3, 4.4 and 4.5 of Chapter 4 are going to analyse further in this section.

The results gathered by executing task locally are considered benchmark for the proposed solution. Here, the prototype application consists of generating two random matrices at local mobile device which are then multiplied with each other and displayed the resulted matrix. The prototype application ran for 30 different computational intensities, started from 160x160 to 450x450 with the increase of 10 in each following computational intensity. It is observed from the experimental results that the ET/TT of executing the application at local mobile devices varies between different computational intensities.

Figure 5.1 shows the ET in milliseconds (ms) against the multiplication of different matrix sizes, executed on local mobile device Samsung Galaxy A5. The mean ET of processing application carrying the intensity 160x160 for a sample space of 20 is equal to 11,701 ms, with 99 % confidence interval 11701 (+/-) 800 ms which shows the range of possible ET between 12,501 ms and 10,901 ms. The variation in ET determined by calculating SD for each intensity of sample space 20 which is 346.53 ms for 160x160.

Similarly, the ET of computational intensity 170x170 is equal to 8024 ms with 12,675(+/-)807 ms while the SD for sample space 20 is equal to 349.76ms. By a close observation the ET pattern increases with an approximate one second with each higher intensity. The ET of computational intensity 450x450 reached up to 111,799 ms with a SD value 4434 ms and of confidence interval 111,799 (+/-) 10,234 ms.

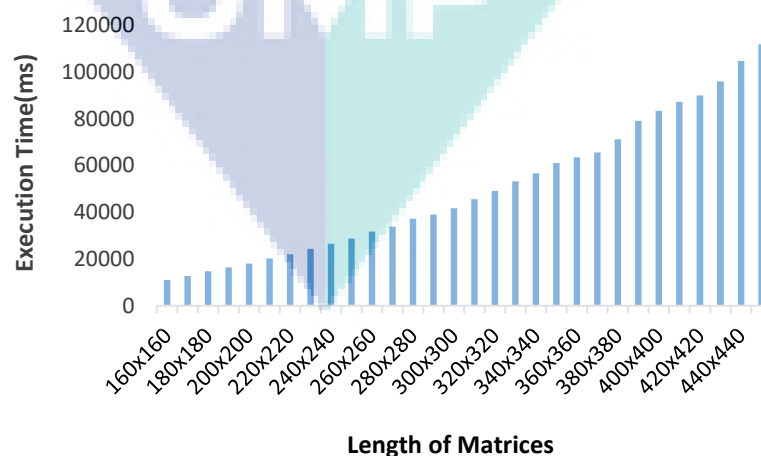


Figure 5.1 Execution Time (ms) of Matrix Multiplication in Local Mobile Device Samsung Galaxy A5

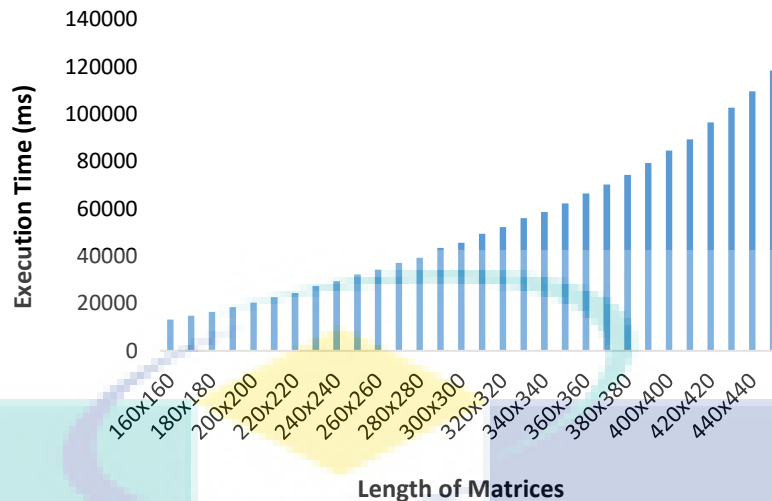


Figure 5.2 Execution Time (ms) of Matrix Multiplication in Local Mobile Device ASUS Zenfone5

Similarly, Figure 5.2 shows the ET in milliseconds (ms) against the multiplication of different matrix sizes, executed on local mobile device ASUS Zenfone5. The mean ET of processing application carrying the intensity 160x160 for a sample space of 20 is equal to 13,240 ms, with 99 % confidence interval 13240 (+/-)478 ms which shows the range of possible ET between 13,718 ms and 12,762 ms. The variation in ET determined by calculating SD for each intensity of sample space 20 which is 207.36 ms for 160x160.

The ET of computational intensity 300x300 is equal to 45,700 ms with 45,700 (+/-)672 ms while the SD for sample space 20 is equal to 291.55ms. With a similar increase to a higher intensity the execution time (ET) pattern increases with approximate 2-3 seconds. The ET of computational intensity 450x450 reaches up to 118,460 ms with a SD value 230.22 ms and of confidence interval 118,460 (+/-)531 ms.

Similarly, in the second part of this section, the Energy Consumption (EC) analysed for both the devices, while executed the task locally. The EC in Joules (J) of executed the prototype application at local mobile devices are presented in Table 4.3 and Table 4.4 of Chapter 4. Figure 5.3 shows the EC in Joules (J) against the multiplication of two random matrices of different sizes at Samsung Galaxy A5. It is observed that the EC varying for computing the matrix multiplication of all intensities. It is lowest for multiplying the lowest size/intensity 160x160 and gradually rosed till the

highest computational intensity 450x450, which seized CPU for a longer time to execute. The observed mean consumption of multiplying the matrices of dimensions 160x160 is 4.58 J which is calculated by running the same intensity for sample size 20. The 99 % confidence interval $4.58(\pm)1$, shows that the possible range of EC fall in between $4.58+1$ J and $4.58-1$ J. The SD shows the variation in observed values of the same sample for the same intensity. It is 0.249 J for the computational intensity 160x160 of the same sample space in each experiment.

Likewise, the mean EC of computational intensity 300x300 observed, which is 19.62 J with 99 % confidence interval $19.62(\pm)0$. The confidence interval with margin of error 0 shows, the standard deviation SD is minimal and the values of calculated EC in the same sample are almost equal. The SD observed for the computational intensity 300x300 is 0.164 J while the % RSD which is 0.84 % shows that the SD value of the same intensity is 0.84 % of the mean value. The lowest the % RSD the closer are the sample values. Also, the observed mean EC of multiplying matrices of dimensions 450x450 is 45.4 J with a confidence interval $45.4(\pm)0$. The EC of the highest intensity in the experiments 450x450 is higher than all the computational intensities. The observed SD value of the sample size for the same intensity is 0.158 J which is lower than 1 therefore the margin of error is rounded to 0. Hence, the calculated 99 % confidence interval for the same intensity is $45.4(\pm)0$ J. The % RSD 0.35 is the percentage Joules of standard deviation to energy consumption of mean.

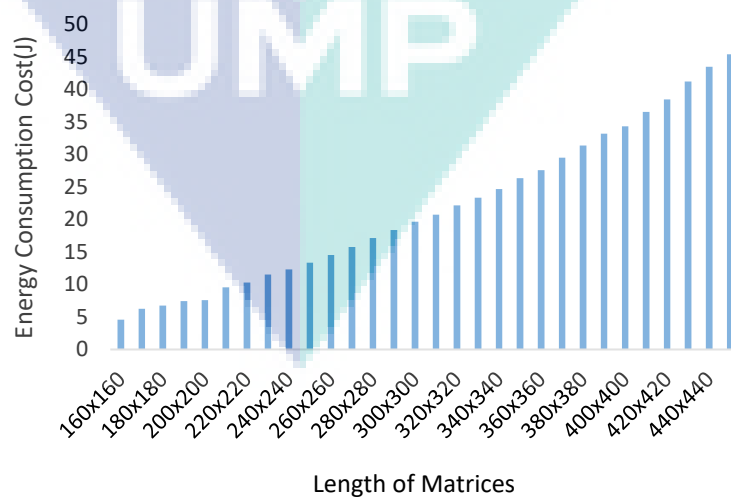


Figure 5.3 Local Energy Consumption (J) of Matrix Multiplication by Samsung Galaxy A5

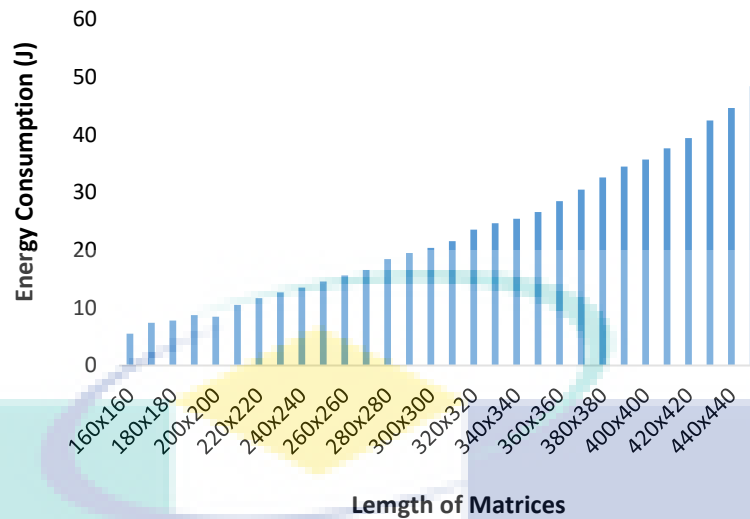


Figure 5.4 Local Energy Consumption (J) of Matrix Multiplication by ASUS Zenfone5

Further, the EC of executing task locally at ASUS Zenfone5 given in Figure 5.4. The mean energy consumption of multiplying the matrices of dimensions 160x160 is 5.54 J with 99 % confidence interval $5.54(+/-) 0$ shows that the possible range of EC fall in between 5.54+0 J and 5.54-0 J. The calculated SD for sample space 20 is 0.30 J. Similarly, the mean EC of last intensity 450x450 is 48.4 J with a SD 0.51 and % RSD 1.06. Comparative analysis of both the devices discussed in Section 5.6 of this chapter. Here, the EC of locally executing the task at ASUS Zenfone5 for all the intensities ranged between 5.48 J to 48.4 J while the same for Samsung Galaxy A5 ranged between 4.58 J to 45.4 J.

5.3 Analysis of Application Executed through Traditional Computational Offloading Methods

The prototype application executed using traditional computational offloading methods in order to estimate the ET in Milliseconds (ms) and EC in Joules (J). The experimental results conducted with both the devices presented in Table 4.6, Table 4.7, Table 4.8 and Table 4.9 of Chapter 4 going to analyse further in this section. The results of executing the porotype application at remote servers through traditional computational offloading methods considered as a second benchmark. Also, the research published in 2013, the DEAP Framework Shiraz *et al.*, (2014) considered as a second benchmark. Similarly, the prototype application consists of generating two

random matrices at local mobile device which are then offloaded to remote servers for multiplication with each other and then displayed the resulted matrix on mobile screen. The ET and EC by using traditional method are higher than the local execution because the communication overhead and extra resources utilization involved in performing the executions. The prototype application ran for 30 different computational intensities started from 160x160 to 450x450 with the increase of 10 in each following computational intensity.

From the experimental results of Table 4.6 of Chapter 4, it is observed that the ET/TT of executing the application using traditional computational offloading method on Samsung Galaxy A5 varies between different computational intensities. Figure 5.5 shows the ET in milliseconds (ms) against the multiplication of different matrix sizes executed on remote servers offloaded through traditional computational offloading method using Samsung Galaxy A5. The mean ET of processing application carrying the intensity 160x160 for a sample space of 20 is equal to 9,608 ms, with 99 % confidence interval $9608 (+/-)1236$ ms which shows the range of possible ET between $9608+1,236$ ms and $9,608-1,236$ ms. The variation in ET determined by calculating SD for each intensity of sample space 20 which is 535.78ms for 160x160.

Likewise, the ET of computational intensity 200x200 is equal to 14286ms with $14,286(+/-) 928$ ms while the SD for sample space 20 is equal to 402.23 ms. By a close observation the ET pattern increases about 1-2 seconds while computing the lower computational intensity under the range 240x240. For higher intensities above than 240x240, the ET quickly increased for each intensity approximate 5 seconds for each.

The ET of computational intensity 450x450 reached up to 189,523 ms with a SD value 2,089 ms and of confidence interval $189,523(+/-) 4822$ ms. It is thus clear from the evaluation and observations of executing all computational intensities that as the computational intensity increases the ET increases. Compare to the local execution, the ET/TT reached up to 189,523 ms which is about 80,000 ms longer than executing the same intensity by local execution at mobile device.

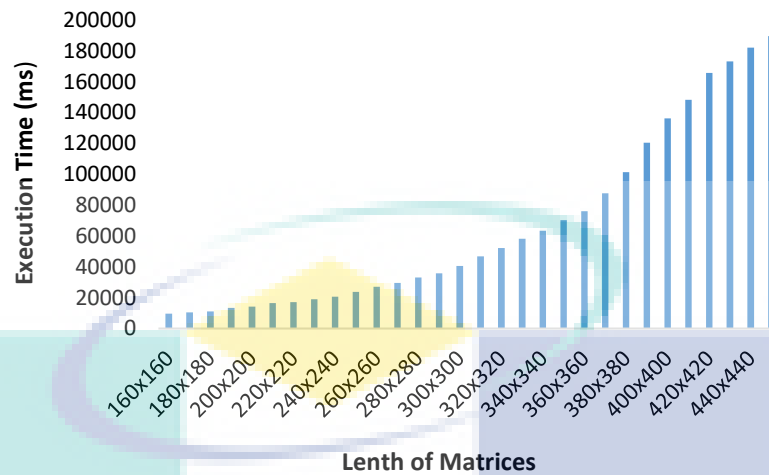


Figure 5.5 Execution Time of Matrix Multiplication in Traditional Offloading by Samsung Galaxy A5

Similarly, from the experimental results of Table 4.7 of Chapter 4, it is observed that the ET/TT of executing the application using traditional computational offloading method on ASUS Zenfone5 varies between different computational intensities. Figure 5.6 shows the ET in milliseconds (ms) against the multiplication of different matrix sizes executed on remote servers offloaded through traditional computational offloading method.

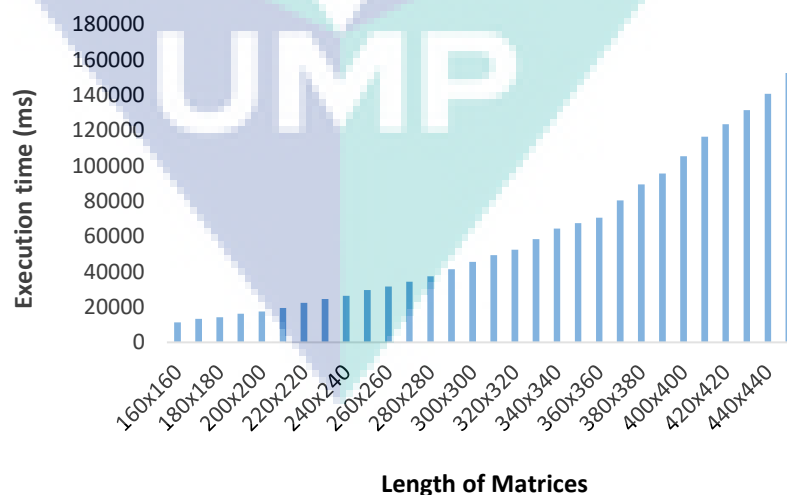


Figure 5.6 Execution Time of Matrix Multiplication in Traditional Offloading by ASUS Zenfone5

The mean ET of processing application carrying the intensity 160x160 for a sample space 20 is equal to 11280ms, with 99 % confidence interval 11,280(+/-)379 ms which shows the range of possible ET between 11,280+379 ms and 11,280-379 ms. The variation in ET determined by calculating SD for each intensity of sample space 20 which is 535.78 ms for 160x160. For computational intensity 300x300 the mean execution time is 45,520 ms with SD 228.04 and 0.50 % RSD. The last intensity 450x450 executed in ASUS Zenfone5, the execution time reaches up to 152,520 ms with 238.75 SD and 0.16 % RSD.

Further, the EC in Joules (J) of executing the prototype application at remote servers by traditional computational offloading methods, for both the devices are presented in Table 4.8 and Table 4.9 of Chapter 4. Figure 5.7 shows the EC in Joules (J) against the multiplication of two random matrices of different sizes in traditional offloading using Samsung Galaxy A5. It is observed that the EC varies of computing the matrix multiplication for all the intensities. EC is lowest for multiplying the lowest size/intensity 160x160 and gradually increasing till the highest computational intensity which is 450x450. The observed mean EC of multiplying the matrices of dimensions 160x160 is 7.72 J which is calculated by running the same intensity for sample size 20. The 99 % confidence interval 7.72(+/-)1 shows that the possible range of energy consumption fall in between 7.72+1 J and 7.72-1 J.

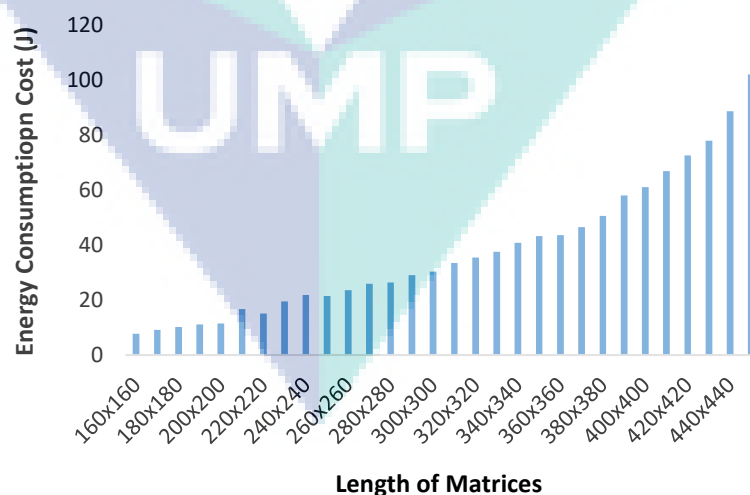


Figure 5.7 Energy Consumption (J) of Matrix Multiplication in Traditional Offloading by Samsung Galaxy A5

SD is 0.421 J for the computational intensity 160x160 of the same sample space in each experiment. Similarly, the mean energy consumption of computational intensity 300x300 is observed which is 30.36 J with 99 % confidence interval 30.36(+/-)1. The confidence interval with margin of error 1 shows the standard deviation SD is minimal and the values of calculated energy consumptions in the same sample are almost equal. The SD observed for the computational intensity 300x300 is 0.241 J while the % RSD which is 0.79 % shows that the standard deviation value of the same intensity is 0.79 % of the mean value.

Furthermore, the observed mean EC of multiplying matrices of dimensions 450x450 is 102.08 J with a confidence interval 102.08(+/-)1. The energy cost of the highest intensity on the experiments, 450x450 is higher than all the computational intensities. The calculated 99 % confidence interval for the same intensity is 102.08 (+/-)1 J. The % RSD 0.59 is the percentage energy consumption in Joules of standard deviation to energy consumption of mean.

Subsequent in Figure 5.8 shows the EC in Joules (J) against the multiplication of two random matrices of different sizes in traditional offloading using ASUS Zenfone 5. By the observations of collected results, it is clear that EC of lower intensities are lowest starting from 8.56 J while the same for higher intensities increases and reaches up to 102.74 J for 450x450.

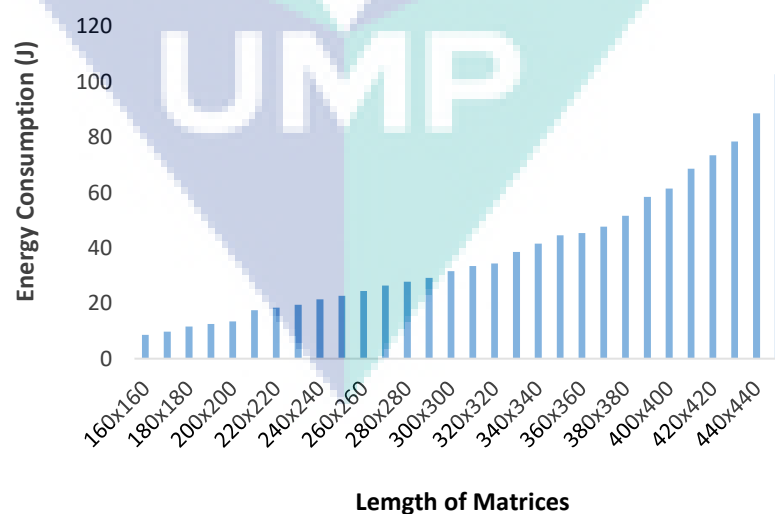


Figure 5.8 Energy Consumption (J) of Matrix Multiplication in Traditional Offloading by ASUS Zenfone5

5.4 Analysis of Application Execution using REST-Offload Method

The prototype application executed by proposed REST-Offload method in order to estimate the ET in milliseconds (ms) and EC in Joules (J). The experiments conducted with both the devices. The detail experimental results presented in Table 4.10, Table 4.11, Table 4.12 and Table 4.13 of Chapter 4 are going to further analyse in this section. The main concern of deploying the surrogate (cloudlet) to reduce distance between client device and remote service, which is on the other hand hit power due to longer RTT. Here, the similar prototype application consists of generating two random matrices at local mobile device which are then offloaded to surrogate servers for multiplication and displays the resulted matrix on mobile screen. The ET and EC by using REST-Offload method are very low than the local execution and of traditional offloading methods because the communication overhead and extra resources utilization are minimized in performing the execution.

From the experimental results, it is observed that the ET of executing the application using the proposed solution REST-Offload computational offloading methods varies between different computational intensities. Figure 5.9 shows the ET in milliseconds (ms) by conducting experiment with Samsung Galaxy A5.

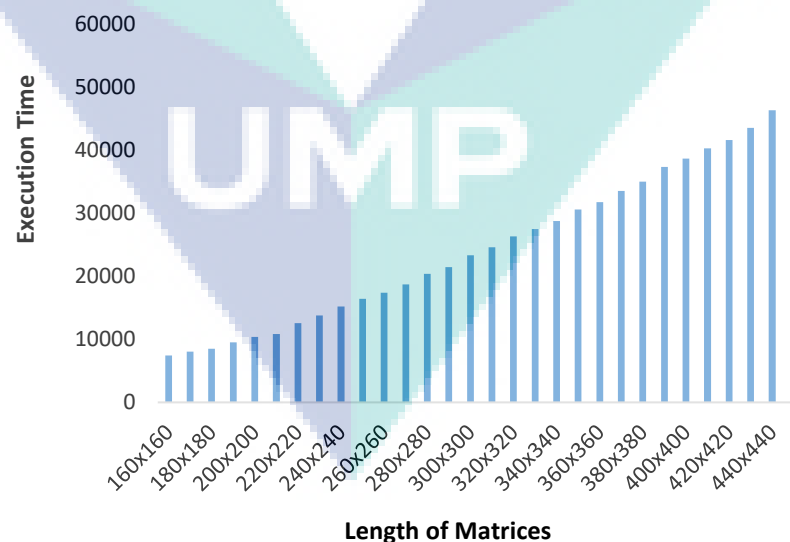


Figure 5.9 Execution Time (ms) of Matrix Multiplication in REST-Offload using Samsung Galaxy A5

The component of prototype application where the multiplication of different matrix sizes, executed on remote servers offloaded through REST-Offload method. The mean ET of processing application carries the intensity 160x160 for a sample space 20 is equal to 7,418 ms, with 99 % confidence interval 7,418(+/-)514 ms. It shows the range of possible ET between 7418+514ms and 7,418-514 ms. The variation in ET determined by calculating SD for each intensity of sample space 20 which is 222.86ms for 160x160.

Similarly, the ET of computational intensity 300x300 is equal to 23365ms with 23,365(+/-) 660 ms while the SD for sample space 20 is equal to 285.93 ms. By a close observation a gradual and constant proliferation noted in the ET pattern in computing each higher computational intensity in the complete set of 30 intensities. This increase is approximate a second or two for each following intensity. The ET of computational intensity 450x450 reached to 49587ms with a SD value 274.23 ms and of confidence interval 49,587(+/-) 633 ms.

It is clear from the evaluation and observations of executing all computational intensities through REST-Offload that, as the computational intensity increases the ET increases. Compare to local execution here the ET reached up to 49,587 ms which is about 61,000 ms less than the local execution of prototype application and 140,000 ms from traditional computational offloading techniques.

Further, the ET of executing prototype application through REST-Offload using ASUS Zenfone5 are shown in Figure 5.10. The results given in Table 4.11 of Chapter 4 are going to analyse here. The ET of executing task using ASUS Zenfone5 and Samsung Galaxy A5 through REST-Offload is identical. The detail comparison of ET of both the devices through REST-Offload is given in Section 5.7 of this Chapter. If we consider the three random intensities as shown in Figure 5.10, the mean ET of intensity 160x160 is 7,780 ms with SD 164.32 and 2.11 % RSD. Similarly, the mean ET of computational intensities 300x300 is 24,320 ms with SD 228.04 and 0.94 % RSD. The last and highest computational intensity of all 30 intensities is 450x450 and the mean ET of this intensity is 61,520 ms with 178.89 SD and 0.29 % RSD. Offloading through REST-Offload is comparatively impressive as it reduces ET about 30 %. The detail comparison is given in Section 5.9 of this Chapter.

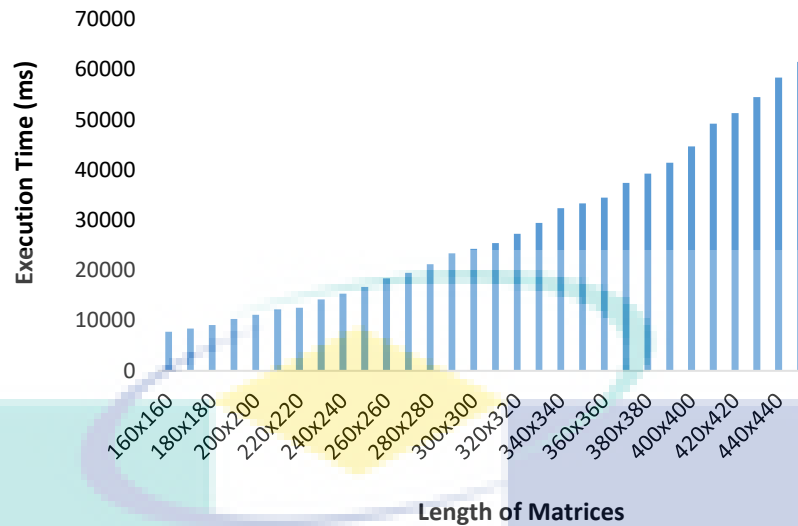


Figure 5.10 Execution Time (ms) of Matrix Multiplication in REST-Offload using ASUS Zenfone5

Likewise, the EC in Joules (J) of executing the prototype application at surrogate servers by offloading through REST-Offload are presented in Table 4.12 and Table 4.13 of Chapter 4. The experimental results of multiplying two random matrices of different sizes with 30 computational intensities using REST-Offload are further analysing in this section. Figure 5.11 shows the Energy Consumption in Joules (J) against the multiplications of two random matrices of different sizes using Samsung Galaxy A5. It is perceived that the EC fluctuating for computing the matrix multiplication of different intensities, started from lowest consumption to higher as the intensity increases.

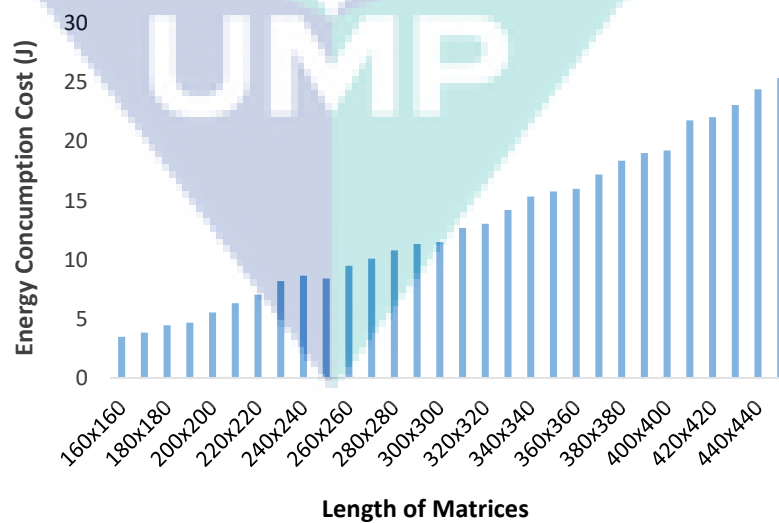


Figure 5.11 Energy Consumption (J) of Matrix Multiplication in REST-Offload using Samsung Galaxy A5

The EC is lowest for multiplying the lowest size/intensity 160x160 and gradually increasing till the highest computational intensity which is 450x450. The observed mean EC of multiplying the matrices of dimensions 160x160 is 3.48 J which is calculated by running the same intensity for sample size 20. The 99 % confidence interval 3.48(+/-)0 shows that the margin of error 0 and therefore the possible range of EC closely fall around 3.48 J. The SD shows the variation in observed values of the same sample for the same intensity. It is 0.1 for the computational intensity 160x160 of the same sample space in each experiment.

Similarly, the mean EC of computational intensity 300x300 is observed 11.50 J with 99 % confidence interval 11.50 (+/-)1. The confidence interval with margin of error 1, shows the standard deviation SD is minimal and the values of calculated EC in the same sample are almost equal. The SD observed for the computational intensity 300x300 is 0.31 J while the % RSD which is 2.75 % shows that the standard deviation value of the same intensity is 2.75 % of the mean value. The lowest the % RSD the closer are the sample values. Also, the observed mean EC of multiplying matrices of dimensions 450x450 is 25.35 J with a confidence interval 25.35(+/-)0. The EC of the highest intensity in the experiment 450x450 is higher than all the computational intensities. The % RSD 0.48 is the percentage EC in Joules of standard deviation to energy consumption of mean.

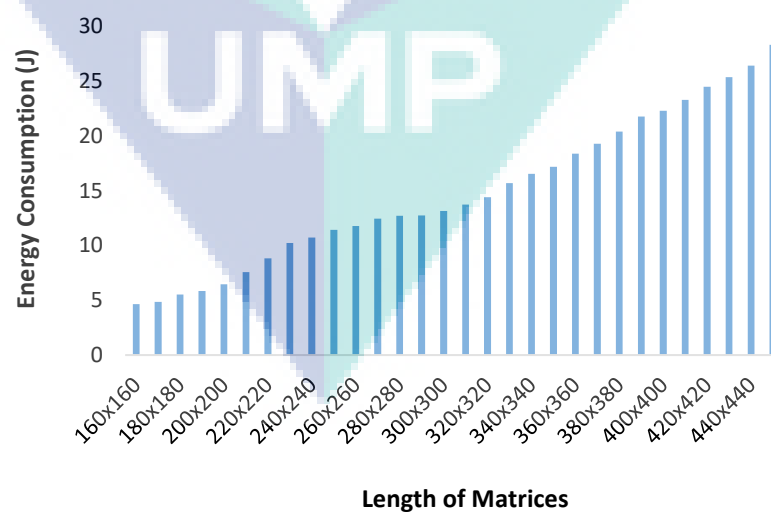


Figure 5.12 Energy Consumption (J) of Matrix Multiplication in REST-Offload by ASUS Zenfone5

Next, the EC cost of executing same intensities using ASUS Zenfone5 shown in Figure 5.12. The results gathered in Table 4.13 of Chapter 4 are further analysing and comparing here. The three random intensities are picked. The EC at ASUS Zenfone5 of 160x160 through REST-Offload is 4.64 J with SD 0.29 and 6.21 % RSD. Similarly, the EC of 300x300 is 13.16J with SD 0.25 and 1.19 % RSD. The EC of the highest computational intensity 450x450 is 28.32 J with SD 0.30 and 1.07 % RSD. EC of ASUS Zenfone5 through REST-Offload is slight higher than the one executed at Samsung Galaxy A5 which is in detail compared and analysed in Section 5.7 of this Chapter.

5.5 Comparison of Execution Time of Matrix Multiplication Service between Local Execution, Traditional Offloading and REST-Offload

This section consists of the results comparison and analysis of both the devices Samsung Galaxy A5 and ASUS Zenfone5 in terms of Execution Time (ET) for all three scenarios Local Execution, Traditional Offloading and REST-Offload.

5.5.1 Execution Time (ET) Result Comparison of Samsung Galaxy A5

Here, the ET of all three scenarios conducted with Samsung Galaxy A5 is going to analyse. Table 5.1 shows sample results comprise of the ET results gathered for all three scenarios with Samsung Galaxy A5. The rest of the result shown in appendix E (Table E.1). Analysing of ET is crucial for two basic reasons. Firstly, it is proved in all the previous researches that, network communication, CPU Processing and longer ET always hit the battery power (Anand *et al.*, 2007). Also, the shorter ET is most important for real time processing as well as user's demand for instant interaction with their gadgets. It is considered further to analyse and compare how long the prototype application took place to execute in each scenario.

Table 5.1 Comparison of ET of Samsung Galaxy A5 between Local Execution, Traditional Offloading and REST-Offload

Matrix Size	ET of Local Execution	ET of Traditional Offloading	ET of REST-Offload Method
160x160	11071	9608	7418
450x450	111799	189523	49587

By evaluating the experiments result of Samsung Galaxy A5, the execution time of prototype application at local mobile device is longer than executing the same with traditional offloading techniques for the computational intensities in the range 160x160 to 300x300. In Contrast, the execution time of computational intensities following 300x300 until 450x450 takes longer ET using traditional techniques. It shows that for lower computational intensity of any task execution which involves less computation can be offloaded to remote surrogate through traditional offloading. However, the tasks which are complex in computation, required longer execution time take longer time to execute. Further, the comparison shows that the execution time of different computational intensities starting from 160x160 until 450x450, in all the three scenarios, execution time of local mobile device is greater than the REST-Offload while less than traditional offload. Similarly, executing the same intensity at traditional computational offloading, the ET is less than the ET of local execution. Similarly, the ET of traditional offloading is greater than ET of the same intensity if execute it through REST-Offload method. Figure 5.13 shows the complete comparison of the three scenarios. It is clear that ET of conducting offloading through REST-Offload is lower than attempted at local mobile device or executing the same through traditional offloading techniques.

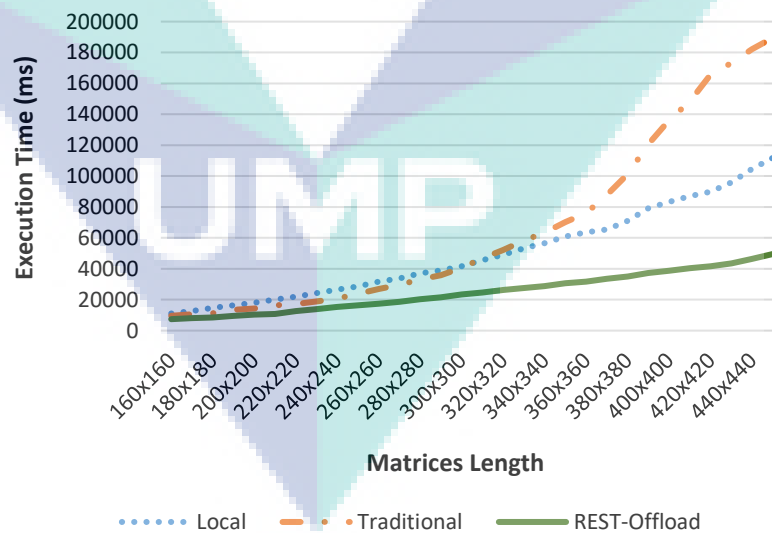


Figure 5.13 Execution Time (ms) Comparison of Matrix Multiplication of all Three Scenarios using Samsung Galaxy A5

Table 5.2 P% of ET of REST-Offload Method for Samsung Galaxy A5 using the equation, $Y = P\% * X$, against Local Execution and Traditional Offloading

Computational Intensity	% ET of REST-Offload against Local Execution	% ET of Traditional Offloading against Local Execution	% ET of REST-Offload Method against Traditional Offloading
160x160	67.00	86.79	77.21
300x300	56.02	97.40	57.51
450x450	44.35	169.52	26.16

To go more specific in comparison of all three scenarios the percentage Execution Time calculated of the three random computational intensities. Table 5.2 shows the P% of Execution Time of REST-Offload Method using the equation, $Y = P\% * X$, against Local Execution and Traditional Offloading. It also included Execution Time using Traditional Offloading against Local Execution. It will give the percentage execution of REST-Offload against Local Execution and Traditional Offloading for the three random computational intensities. The percentage Execution time of computational intensity 160x160 executing prototype application using REST-Offload against local execution is 67 % which is decreasing as the computational intensity increases, such as for 300x300 is 56.02 % and for 450x450 is 44.35 %.

By the comparison of the results for all three scenarios using Samsung Galaxy A5 as a DUT, it is clear that as the complexity increases, execution of computational intensive task at REST-Offload becoming useful in terms of Execution Time. Similarly, if consider the ET of traditional computational offloading, which is 86.79 % for 160x160 against local execution which reaches up to 169.52 % for the computational intensity 450x450. It shows that for a task with less computational load can be executed quickly at traditional offloading method while for complex execution task the local execution can process them fast.

5.5.2 Execution Time (ET) Result Comparison of ASUS Zenfone5

The Execution Time (ET) results collected in Table 5.3 consist of sample results of ASUS Zenfone 5 for all three scenarios Local Execution, Traditional Offloading and REST-Offload are going to analyse. Rest of the results are in appendix E (Table E.2).

Table. 5.3 Comparison of ET of ASUS Zenfone5 between Local Execution, Traditional Offloading and REST-Offload

Matrix Size	ET of Local Execution	ET of Traditional Offloading	ET of REST-Offload Method
160x160	13240	11280	7780
450x450	118460	152520	61520

By evaluating the experiments results, the ET of prototype application at local mobile device is longer than executing the same with traditional offloading techniques for the computational intensities in the range 160x160 to 300x300. In Contrast, the execution time of computational intensities after 300x300 until 450x450 takes longer ET using traditional techniques. It shows that for lower computational intensity of any task execution, which involves less computations can be offloaded to remote surrogate through traditional offloading, while for tasks which required higher/complex computations takes longer execution time if delegate the task with Traditional Offloading Methods. The results in terms of Execution Time of ASUS Zenfone5 is almost identical to the result of Samsung Galaxy A5, which is further explained in Section 5.7 of this Chapter.

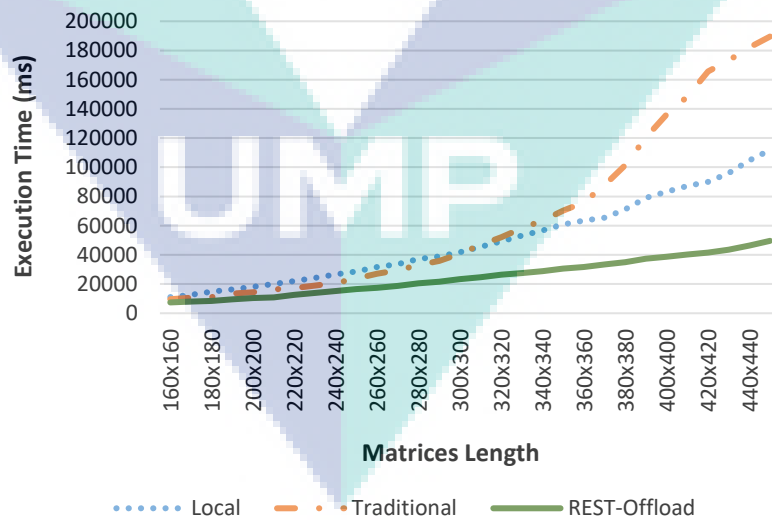


Figure 5.14 Execution Time (ms) Comparison of Matrix Multiplication of all Three Scenarios using ASUS Zenfone5

Table 5.4 P% of ET of REST-Offload Method for ASUS Zenfone5 using the equation, $Y = P\% * X$, against Local Execution and Traditional Offloading

Computational Intensity	% ET of REST-Offload against Local Execution	% ET of Traditional Offloading against Local Execution	% ET of REST-Offload Method against Traditional Offloading
160x160	58.76	85.20	68.97
300x300	53.22	99.61	53.43
450x450	51.93	128.75	40.34

Further, the comparison shows that the ET of different computational intensities, executing 160x160 at all three scenarios, ET of local mobile device is greater than the rest two. Similarly, executing the same intensity at traditional computational offloading way, the ET is less than the ET of local mobile device while greater than ET of same intensity if execute it through REST-Offload method. Figure 5.14 shows the complete comparison of the three scenarios. It is clear that ET of attempting execution through REST-Offload is lower than the one attempted at local mobile device or executing the same through traditional offloading techniques.

To go more specific in comparison of all three scenarios the percentage ET calculated of the three random computational intensities. Table 5.4 shows the P% of ET of REST-Offload Method using the equation, $Y = P\% * X$, against Local Execution and Traditional Offloading. It also included Execution Time using Traditional Offloading against Local Execution. It will give the percentage execution of REST-Offload against Local Execution and Traditional Offloading for the three random computational intensities. The percentage ET of computational intensity 160x160 executing prototype application using REST-Offload against local execution is 58.76 %. It decreases with the computational intensity increase, such as for 300x300 is 53.22 % and for 450x450 is 51.93 %.

By the comparison of the results for all three scenarios using ASUS Zenfone5 as a *DuT*, it is clear that as the complexity increases, execution of the computational intensive tasks at REST-Offload becoming useful in terms of Execution Time. Two types of pattern observed here in the results. If compare all three scenarios, then ET of REST-Offload is lower than the rest two and therefore is more efficient.

In the second pattern, if consider the execution time of traditional computational offloading, which is 85.20 % for 160x160 against local execution and reaches up to 128.75 % for the computational intensity 450x450. It shows that for tasks with less computational load can be executed quickly at traditional offloading methods while for complex tasks the local execution can process faster.

5.6 Comparison of Energy Consumption (EC) of Matrix Multiplication Service between Local Execution, Traditional Offloading and REST-Offload

This section consists of the results comparison and analysis in terms of Energy Consumption (EC) between all the three scenarios Local Execution, Traditional Offloading and REST-Offload. The results of the three scenarios for both devices are distinctly compared as given in following sub-sections.

5.6.1 Energy Consumption (EC) Result Comparison of Samsung Galaxy A5

The EC of prototype application at local mobile device, offloaded to remote servers, through traditional computational method and offloaded through REST-Offload method are collected using Samsung Galaxy A5 in Chapter 4. Table 5.5 shows the sample results of the Energy Consumption Cost gathered for all the three scenarios. The details of results attached in appendix E (Table E.3). Further analysis and comparison of energy consumption costs of different computational intensities evaluated using all three scenarios. By evaluating the experiments results the energy consumption cost of prototype application at local mobile device is greater than executing the same application using REST-Offload techniques, for all the 30 different computational intensities starting from 160x160 to 450x450.

Table. 5.5 Comparison of Energy Consumption Cost between Local Execution, Traditional Offloading and REST-Offload for Galaxy A5

Matrix Size	EC (J) in Local Execution	EC (J) in Traditional Offloading	EC (J) in REST-Offload Method
160x160	4.58	7.72	3.48
450x450	45.4	102.08	25.35

It is also observed that the EC of local execution is less than the energy consumption costs of traditional offloading techniques, for all the 30 computational intensities 160x160 to 450x450. It shows that for the used test case, traditional computational offloading methods are complex and energy intensive.

Moreover, the Traditional Offloading techniques are complex as it needed extra mobile's resources for computational offloading which is discussed in detail in Chapter 3. The inference component, dynamic partitioning and virtual machine migrations are few of the heavyweight procedures which turns the offloading techniques computational intensive. The energy costs for executing all the 30 computational intensities using REST-Offload method are very low compare to the rest two.

Further, comparison of the energy consumption costs of few selected cases of computational intensities for all the three scenario attempted. It shows, that the energy consumption cost of local mobile device is 4.58 J which is less than the energy consumption cost of traditional method 7.72 J while is greater than the REST-offload method 3.48 J. Similarly, the energy consumption cost of computational intensity 300x300, executed at local mobile device is 19.62 J. It is less than the energy consumption cost 30.36 J of executing same intensity through traditional offloading method while greater than the energy consumption cost 11.50 J of executing through REST-Offload. In case of executing 450x450 in all three scenarios, the energy consumption costs of local execution is less than the traditional offloading method while greater than executing the same through REST-Offload.

Figure 5.15 shows the complete comparison of the three scenarios. It is clear from the results that, the Energy Consumption Costs of local execution in executing the same intensity for all the scenarios less than the energy consumption cost at Traditional Methods, while greater than executing the same at REST-Offload. The range of energy consumption cost of local execution fall in the range 4.58-45.4 J for all 30 computational intensities. Similarly, traditional offloading energy consumption cost is in the range of 7.72-102.08 J while REST-offload fall in the range 3.48-25.35 J which comparatively very low of the other two scenarios.

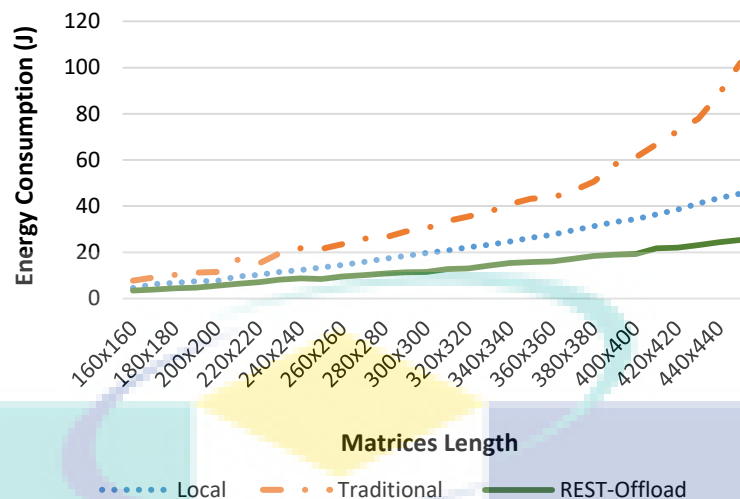


Figure 5.15 Energy Consumption (J) Comparison of Matrix Multiplication of all Three Scenarios using Samsung Galaxy A5

To go more specific in comparison of all the three scenarios the percentage EC calculated of three randomly chosen computational intensities. Table 5.6 shows the P% of REST-Offload Method using the equation, $Y = P\% * X$, against Local Execution and Traditional Offloading. It also included EC of Traditional Offloading against Local Execution. It will give the percentage EC of REST-Offload against Local Execution and Traditional Offloading for the three random computational intensities. The percentage EC of computational intensity 160x160 executing prototype application using REST-Offload against local execution is 75.98 % which is decreasing as the computational intensity increases, such as for 300x300 is 58.61 % and for 450x450 is 55.84 %. It shows that, as the complexity increases, the execution of computational intensive task at REST-Offload getting more useful in terms of Energy Consumption.

Table 5.6 P% of Energy Consumption of REST-Offload Method for Samsung Galaxy A5 using the equation, $Y = P\% * X$, against Local Execution and Traditional Offloading

Computational Intensity	% EC of REST-Offload against Local Execution	% EC of Traditional Offloading against Local Execution	% EC of REST-Offload Method against Traditional Offloading
160x160	75.98	168.56	45.08
300x300	58.61	154.74	37.88
450x450	55.84	224.85	24.83

Similarly, if consider the Energy Consumption Cost of traditional computational offloading which is 168.56 % for 160x160 against local execution, depicts a significant increase in energy consumption compare to local execution. It reaches up to 224.85 % for the computational intensity 450x450. It is clear that, for any kind of intensive task execution through traditional offloading methods are time and energy intensive and drain power more than any other method. The energy consumption cost of REST-Offload is comparatively very low against the energy consumption cost of both the scenarios which are given in percentage in Table 5.6.

5.6.2 Energy Consumption (EC) Result Comparison of ASUS Zenfone5

The EC results of prototype application at local execution, traditional computational offloading techniques and REST-Offload method, are gathered using ASUS Zenfone5 in Chapter 4. Table 5.7 shows the sample results of EC gathered for all three scenarios. It is further considered to analyse and compare EC of different computational intensities evaluated using all the three scenarios. Rest of the results are in appendix E (Table E.4).

By evaluating the experiments results, the EC of prototype application at local mobile device is greater than EC of executing the same application at REST-Offload method. It is also observed that the EC of local execution is less than the EC of executing prototype application through traditional offloading techniques for all the 30 computational intensities 160x160-450x450. The results affirm that, for the used test case, the energy costs of traditional computational offloading methods are energy intensive. The reason of energy intensity of traditional computational offloading methods is the extra mobile's resources utilization in computational offloading which is already discussed in Chapter 3.

Table 5.7 Comparison of Energy Consumption Cost between Local Execution, Traditional Offloading and REST-Offload for ASUS Zenfone5

Matrix Size	EC (J) in Local Execution	EC (J) in Traditional Offloading	EC (J) in REST Offload Method
160x160	5.54	8.56	4.64
450x450	48.4	102.74	28.32

The inference component, dynamic partitioning and virtual migration are few of the heavyweight procedure which turns offloading to a resource intensive solution. The energy cost for executing all 30 different computational intensities using REST-Offload method are very low compare to the rest two. Further, the comparison of the EC of few individual cases of computational intensities such as, for the intensity 160x160 the observed EC of all three scenarios is $4.64\text{ J} < 5.54\text{ J} < 8.56\text{ J}$ shows that Rest-Offload $<$ Local-Execution $<$ Traditional-Offload. Similarly, the EC of computational intensity 300x300 executed at local mobile device is 20.38J which is less than the EC 31.6J of executing same intensity through traditional offloading method while greater than the EC 13.16 J of executing it by REST-Offload. In case of executing 450x450 in three scenarios the EC of local execution again is less than the traditional offloading method while greater than executing the same through REST-Offload.

Figure 5.16 shows the complete comparison of the three scenarios. It shows that the EC of local execution for the same intensity is less than the EC at traditional Methods, while greater than executing the same at REST-Offload. The range of EC of local execution falls in the range 5.54J-48.4J for all 30 computational intensities. Similarly, EC in traditional offloading falls in the range of 8.56J-102.74J while REST-offload falls in the range 6.64J-28.32J which is comparatively very low than the other two scenarios. To go more specific in comparison of all three scenarios the percentage EC calculated of three randomly chosen computational intensities.

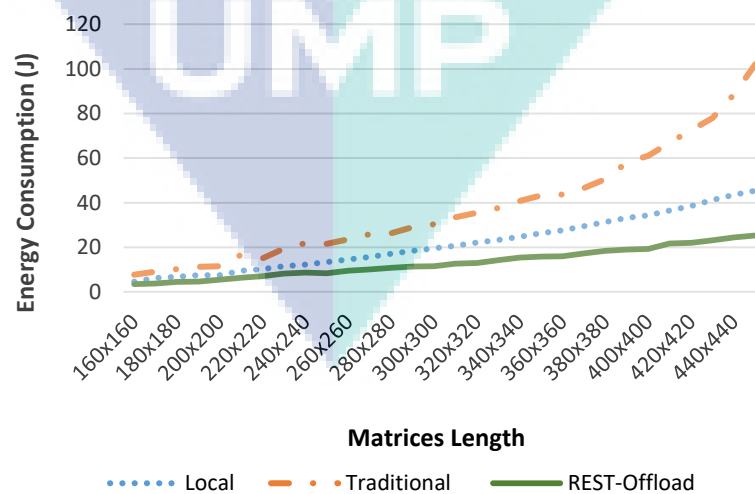


Figure 5.16 Energy Consumption (J) Comparison of Matrix Multiplication of all Three Scenarios using ASUS Zenfone5

Table 5.8 P % of Energy Consumption of REST-Offload Method of ASUS Zenfone5 using the equation, $Y = P\% * X$, against Local Execution and Traditional Offloading

Computational Intensity	% EC of REST-Offload against Local Execution	% EC of Traditional Offloading against Local Execution	% EC of REST-Offload Method against Traditional Offloading
160x160	75.98	168.56	45.08
300x300	58.61	154.74	37.88
450x450	55.84	224.85	24.83

Table 5.8 shows the P% of REST-Offload Method using the equation, $Y = P\% * X$, against Local Execution and Traditional Offloading. It also included Energy Consumption Cost using Traditional Offloading against Local Execution. It gives the percentage Energy Consumption Cost of REST-Offload against Local Execution and Traditional Offloading for the three randomly chosen computational intensities.

The percentage EC of computational intensity 160x160 executing prototype application using REST-Offload against local execution is 75.98 %. It is decreasing as the computational intensity increasing, such as for 300x300 is 58.61 % and for 450x450 is 55.84 %. It shows that as the complexity increases, the execution of computational intensive task at REST-Offload getting more efficient in terms of Energy Consumption. Similarly, if consider the percentage Energy Consumption Cost of traditional computational offloading against local execution, which is 169.56 % for 160x160. It shows a huge increase in energy consumption compare to local execution. It reaches up to 224.85 % for the computational intensity 450x450. It is clear from the results that for any kind of intensive task execution through traditional offloading methods are time and energy intensive and drain power more than any other method. The EC of REST-Offload is comparatively very low against the energy consumption cost of both the scenarios which are given in percentage in Table 5.8.

5.7 Comparison of Execution Time (ET) and Energy Consumption (EC) between REST-Offload and DCOF Framework

DCOF Framework proposed by Shiraz *et al.*, in (2013), as a lightweight solution for addressing the resources intensity and communication overhead. DCOF is considered as a second benchmark in this study. Both, REST-Offload and DCOF (Distributed Computational Offloading Framework) are going to analyse in this section in terms of ET and EC. Also, the data set of DCOF Framework is considered in REST-Offload for testing ET and EC. The intention of selecting same data set was the comparison of results against DCOF (the benchmark). It was also intended to find out the efficiency of REST-Offload. The first part of this section is going to discuss and compare the ET results of DCOF and REST-Offload model. The second part consist of discussion and comparison of both the approaches for EC. Table 5.9 shows the sample ET results of executing the computational intensities started from 160x160 until 450x450. The detail results are in appendix E (E.5).

DCOF Framework shows 72 % ET efficiency against Traditional Computational Offloading techniques, while REST-Offload shows 104.2 % ET efficiency against Traditional Computational Offloading Techniques. Figure 5.17 illustrates the comparison of all 30 computational intensities between REST-Offload and DCOF. DCOF solution initially proposed the elimination of unnecessary utilization of mobile's resources during offloading of computational task to remote servers. However, the results collected from both the approaches shows that, by the deployment of Virtual Machine in computational offloading technique demands extra resources at mobile devices. Also, the delegation of VM instance at runtime needs to transfer a huge data to remote servers using the available bandwidth, which takes more time to complete the task.

Table 5.9 Comparison of ET between REST-Offload and DCOF Framework

Matrix Size	ET (ms) in DCOF Framework	ET (ms) of REST-Offload	Difference (ms)
160x160	4241	7418	-3177
450x450	97887	49587	48300

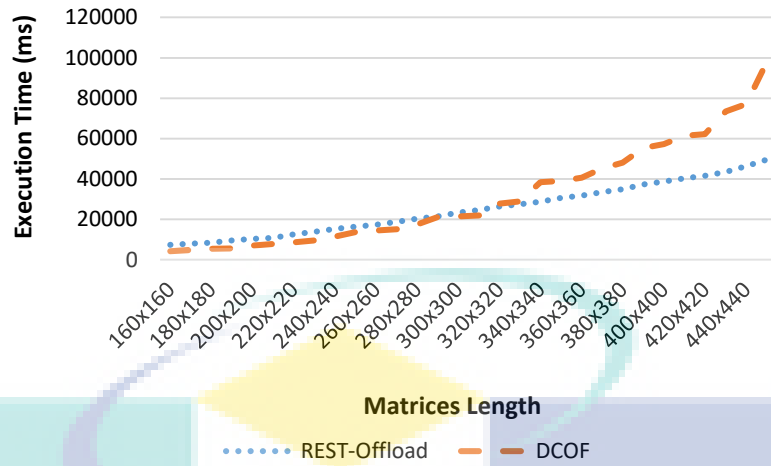


Figure 5.17 ET (ms) Comparison of Matrix Multiplication of between DCOF and REST-Offload

Figure 5.17 shows that for all the tasks which are lightweight can be executed quicker at DCOF while as the complexity of the task increases the resources utilization and communication overhead increases, which increases ET. The ET of DCOF while executing intensities 160x160 to 320x320 is less than REST-Offload whereas, the ET intersects the ET of REST-Offload after computational intensities 340x340 to 450x450, which clearly shows that for higher complexity REST-Offload is efficient than DCOF.

Next, this section also discusses the EC of matrix multiplication operation between both the approaches. Table 5.10 shows the sample result comparison of EC while executing computational intensities 160x160 to 450x450 through DCOF and REST-Offload. Referred to appendix E (Table E.6) for details results. Previously, by examining the results of local execution against REST-Offload, the EC increases as the ET increases. The EC given in Table 5.10 plotted to show a precise comparison of both, as shown in Figure 5.18. Against the results of ET which intersects until mid, the EC here is higher in DCOF from the first intensity 160x160 until last intensity 450x450, regardless of the twisting of ET at mid.

Table 5.10 Comparison of EC between REST-Offload and DCOF Framework

Matrix Size	EC (J) in DCOF Framework	EC (J) of REST-Offload	Difference (J)
160x160	10.8	3.48	7.32
450x450	65.3	25.35	39.95

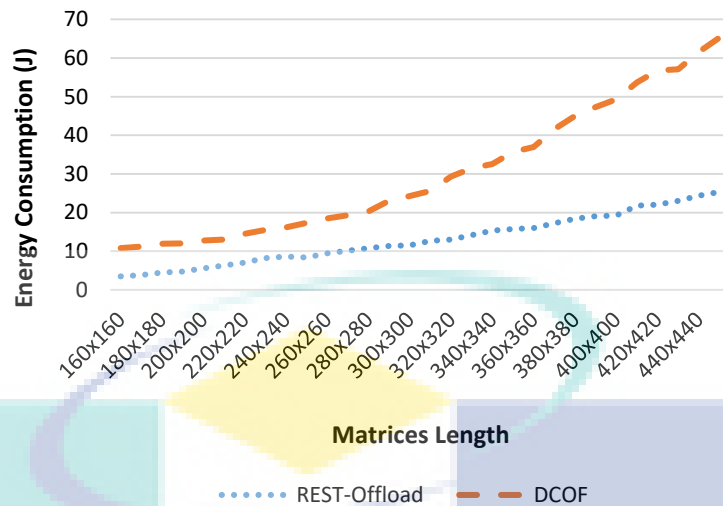


Figure 5.18 EC (J) Comparison of Matrix Multiplication of between DCOF and REST-Offload

The EC range of DCOF starting from 10.8 J for intensity 160x160 and reaches up to 65.3 J for the last intensity 450x450. Similarly, the EC range of REST-Offload is 24.35 J, starting from 3.48 J for the lowest intensity while 25.35 J is the EC for highest intensity 450x450. The ET efficiency of REST-Offload against the benchmarks Local Execution and DCOF is 54.63 % and 35.01 % respectively. Also, the EC efficiency of the proposed REST-Offload model against both the benchmarks Local Execution and DCOF is 43 % and 60.14 % respectively. The observation and comparison of results in all three scenarios such as Local Execution, REST-Offload and of DCOF shows that, REST-Offload is far efficient in ET and EC both than DCOF and Local Execution.

5.8 ET and EC Comparison of Samsung Galaxy A5 with ASUS Zenfone5 for all Three Scenarios

Two different state-of-the-art mobile devices of two different vendors used as *DuT* in the experiments. The intension of using different mobile devices with different specifications to know the effect of lower/higher speed processor and different network interfaces on battery consumption. Also to know the efficiency of proposed light weight method by changing specification of mobile models and brands. This section consists of the results comparison between two different devices, in terms of ET and EC. In the first part the comparison in terms of Execution Time (ET) is presented in Table 5.11 as:

Table 5.11 ET Comparison of Samsung Galaxy A5 and ASUS Zenfone5

Matrix Size	ET (ms) of Local Exec.		ET (ms) of Traditional-Off.		ET (ms) of Rest-Off	
	Galaxy A5	ASUS Z5	Galaxy A5	ASUS Z5	Galaxy A5	ASUS Z5
160x160	11071	13240	9608	11280	7418	7780
170x170	12675	14880	10544	13220	8042	8420
180x180	14696	16460	11152	14260	8503	9140
190x190	16331	18425	13448	16260	9511	10360
200x200	18088	20300	14286	17420	10374	11160
210x210	20146	22620	16406	19420	10844	12260
220x220	22004	24460	17200	22420	12572	12540
230x230	24290	27420	18931	24500	13814	14220
240x240	26542	29380	20687	26400	15236	15360
250x250	28654	32240	23727	29620	16448	16700
260x260	31666	34280	26968	31620	17410	18440
270x270	33751	37160	29593	34360	18719	19540
280x280	37176	39220	33056	37400	20403	21260
290x290	38941	43540	35777	41420	21483	23380
300x300	41711	45700	40627	45520	23365	24320
310x310	45589	49400	46807	49280	24627	25480
320x320	49128	52300	52168	52380	26354	27300
330x330	53249	56060	58184	58360	27505	29460
340x340	56637	58680	63449	64380	28799	32380
350x350	60939	62300	70335	67460	30591	33360
360x360	63523	66540	76010	70562	31768	34480
370x370	65479	70240	87753	80360	33536	37460
380x380	71061	74320	101338	89480	35023	39260
390x390	79131	79340	120554	95560	37374	41440
400x400	83313	84640	136166	105340	38703	44720
410x410	87112	89360	148291	116460	40305	49240
420x420	89920	96540	165687	123500	41631	51320
430x430	95844	102760	173176	131360	43565	54540
440x440	104633	109600	182196	140640	46347	58440
450x450	111799	118460	189523	152520	49587	61520

The ET of processing task in all three scenario of Samsung Galaxy A5 is a bit lower than the ET for the same using ASUS Zenfone5. There are many reasons of differences in the observed results and few key reasons are the specification differences (CPU architectures, clock speeds, number of cores, and amount of RAM) (Shin *et al.*,

2013). Specification of the devices against each other are given in Table 5.12. For a decent performance, the CPU and RAM specifications must be good enough as well as the device need to be in good operating condition. The device condition can also be considered one of the reasons, which effect the device’s performance.

Devices according to the conditions are categorized as, brand new, good condition, reliable, poor condition, broken. The *DuT* in the experiments are Samsung Galaxy A5 which is good in condition after passing through the initial test. A good phone is the one having no visible scratches, battery health good and not much used. A reliable phone is the one which works 100 % but having some visible scratches, used longer and battery condition not very good. Also, due to continuous use of the device, processing speed get slower due to heated up ICS. The ASUS Zenfone5 is reliable in condition which affects the experiment’s result.

Now to distinctly analyse the differences of results, the processor of Samsung Galaxy A5 is Quad-core while ASUS Zenfone5 is Dual-core. In Addition, the condition of Samsung Galaxy A5 is good while ASUS Zenfone5 is reliable. The RF CAL (manufacturing year) of Samsung Galaxy A5 shows the device is latest while ASUS Zenfone5 used longer and reliable only in condition. Due to all these specification’s differences and the devices conditions, the ET results of both the devices in all three scenarios are slightly different. The ET results of Samsung Galaxy A5 is slightly better than ASUS Zenfone5, as shown in Figure 5.19, Figure 5.20 and Figure 5.21.

Table 5.12 Specifications of Samsung Galaxy A5 and ASUS Zenfone5

D. Name	Processor	RAM	Storage	Battery	D. Condition	RF CAL
Galaxy A5	Quad-core 1.2 GHz	2 GB	16 GB	2300mAh	Good	2015.02.06
ASUS Z5	Dual-core 1.2 GHz	2 GB	8 GB	2110mAh	Reliable	2014.01.22

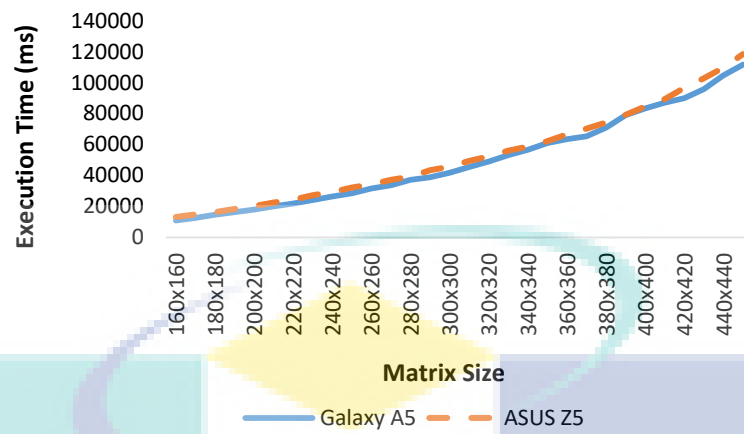


Figure 5.19 Execution Time (ms) Comparisons of Galaxy A5 and ASUS Z5 in Local Execution

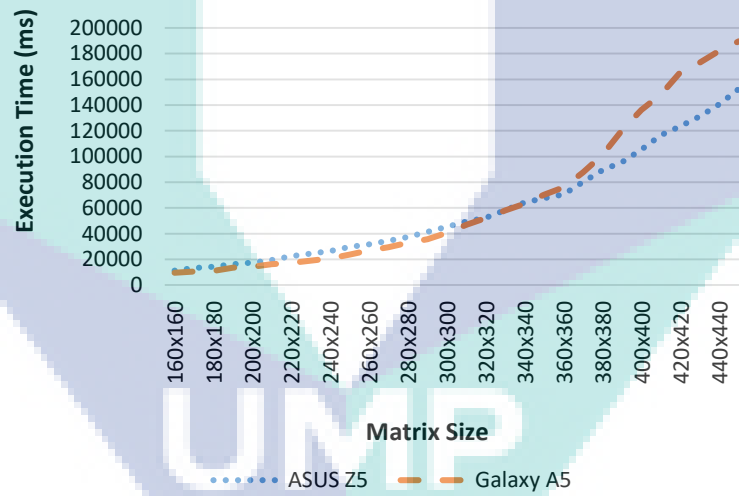


Figure 5.20 Execution Time (ms) Comparisons of Galaxy A5 and ASUS in Traditional Offloading

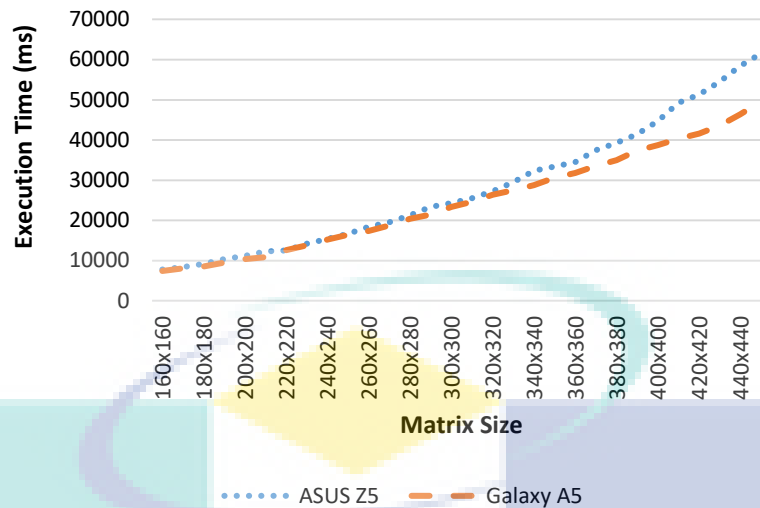


Figure 5.21 Execution Time (ms) Comparisons of Galaxy A5 and ASUS in REST-Offload

Based on the results and specifications of the devices, it is concluded that Samsung Galaxy A5 for all three scenarios is better in executing the task compare to ASUS Zenfone5. Here, in the second part of this section, the comparisons of both the devices in terms of energy consumption (EC) in all three scenarios is going to analyse. Table 5.13 consist of the EC results of all three scenarios conducted with both the devices. Similar to ET, the EC results of Samsung Galaxy A5 observed in all three scenarios are lower than that of ASUS Zenfone5. In other words, executing a complex task in any of the given scenario, Samsung Galaxy A5 drain less power and therefore is energy efficient compare to ASUS Zenfone5.

The energy consumption cost of executing the task at surrogate server, offloaded through Traditional Offloading methods observed closely identical using both the devices. There is only (1-2) J of jump of each higher intensity while processing each intensity. In case of executing the task locally, the EC of Galaxy A5 and ASUS Z5 having a jump of 1-2 J during processing of each intensity. Similarly, during REST-Offload method the EC of Galaxy A5 is efficient then ASUS Z5. For lower intensities the EC difference between both the devices is about 1 J while for higher intensities the difference reaches up to 3 J. The overall analysis of comparison as shown in Table 5.13, shows that Galaxy A5's results are better and efficient than ASUS Z5.

Table 5.13 EC Comparisons of Samsung Galaxy A5 and ASUS Zenfone5

Matrix Size	EC (J) of Local Exec.		EC (J) of Traditional-Off.		EC (J) of Rest-Off	
	Galaxy A5	ASUS Z5	Galaxy A5	ASUS Z5	Galaxy A5	ASUS Z5
160x160	4.58	5.54	7.72	8.56	3.48	4.64
170x170	6.26	7.4	9.14	9.72	3.83	4.86
180x180	6.725	7.78	10.24	11.64	4.45	5.52
190x190	7.44	8.72	11.18	12.58	4.68	5.86
200x200	7.56	8.44	11.5	13.4	5.55	6.46
210x210	9.54	10.54	16.74	17.46	6.33	7.58
220x220	10.3	11.66	15.06	18.4	7.05	8.82
230x230	11.5	12.68	19.48	19.52	8.18	10.24
240x240	12.3	13.5	21.8	21.4	8.65	10.72
250x250	13.36	14.54	21.52	22.7	8.40	11.42
260x260	14.54	15.62	23.56	24.48	9.48	11.78
270x270	15.74	16.58	25.92	26.44	10.08	12.46
280x280	17.12	18.46	26.34	27.8	10.78	12.72
290x290	18.36	19.52	29	29.14	11.33	12.74
300x300	19.62	20.38	30.36	31.6	11.50	13.16
310x310	20.72	21.54	33.52	33.46	12.68	13.74
320x320	22.16	23.58	35.48	34.34	13.03	14.4
330x330	23.32	24.68	37.6	38.54	14.20	15.68
340x340	24.66	25.46	40.82	41.6	15.33	16.54
350x350	26.34	26.6	43.28	44.58	15.78	17.18
360x360	27.58	28.48	43.62	45.4	16.00	18.38
370x370	29.48	30.5	46.56	47.62	17.20	19.3
380x380	31.36	32.62	50.6	51.62	18.38	20.4
390x390	33.16	34.52	58.08	58.42	19.03	21.76
400x400	34.3	35.7	61.08	61.38	19.23	22.3
410x410	36.5	37.64	66.88	68.54	21.78	23.3
420x420	38.46	39.42	72.56	73.5	22.05	24.48
430x430	41.2	42.48	77.96	78.46	23.08	25.36
440x440	43.46	44.66	88.66	88.62	24.40	26.414
450x450	45.4	48.4	102.08	102.74	25.35	28.32

Local Execution, Traditional Offloading and REST-Offload results of Table 5.13 plotted in Figure 5.22, Figure 5.23 and Figure 5.24 respectively. Galaxy A5 drains less battery than ASUS Z5 and again of many other reasons like specification and condition of the device, the battery status and health is important to consider which

affects battery consumption. Similarly, in case of ASUS Z5 the battery health is normal only compare to a good battery health of Galaxy A5.

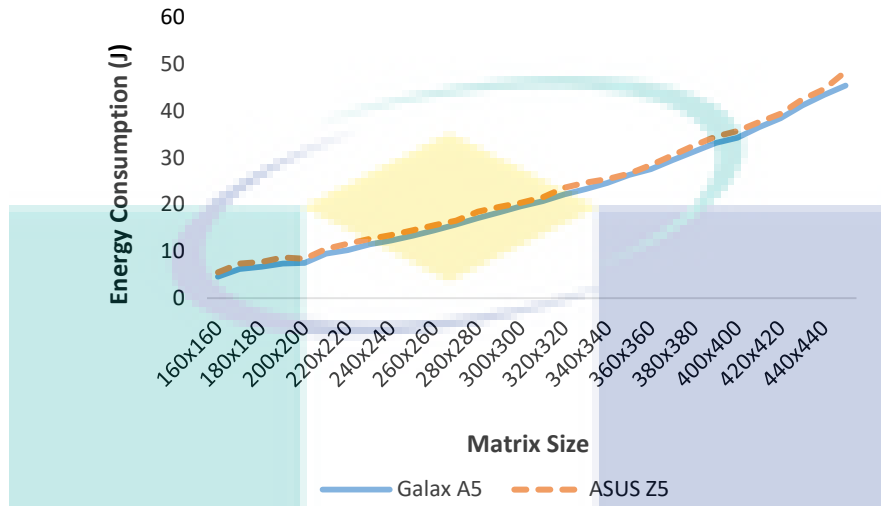


Figure 5.22 Energy Consumption (J) Comparisons of Galaxy A5 and ASUS Z5 in Local Execution

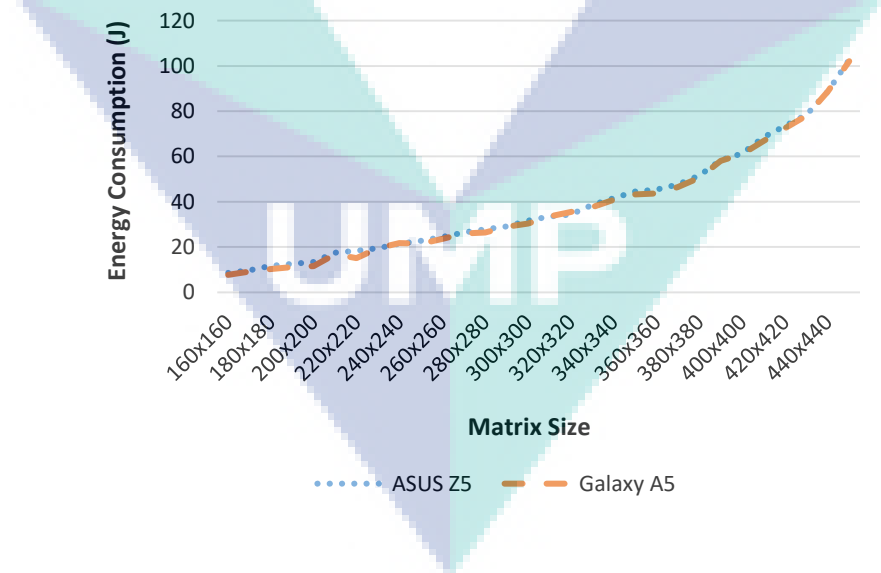


Figure 5.23 Energy Consumption (J) Comparisons of Galaxy A5 and ASUS Z5 in Traditional Offloading

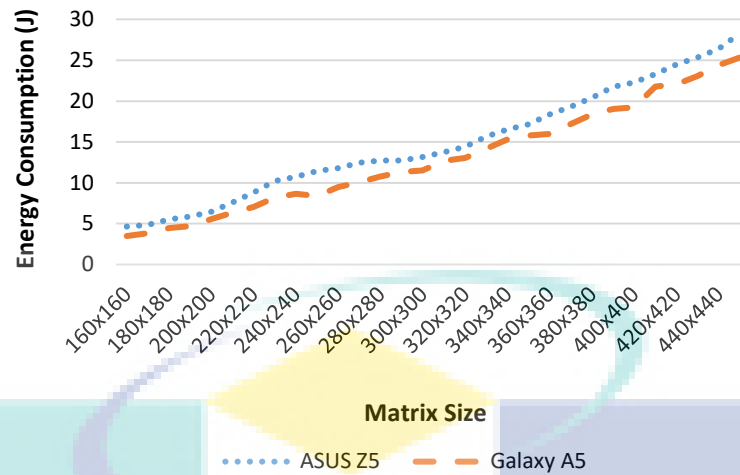


Figure 5.24 Energy Consumption (J) Comparisons of Galaxy A5 and ASUS Z5 in REST-Offload

5.9 Efficiency Comparisons of REST-Offload against Existing Framework

In order to compare the efficiency of REST-Offload against different method level computational offloading frameworks/models the results collected of three method level computational offloading models. The result evaluated based on the execution time and energy consumption in both the scenarios; local execution and offloaded execution. The average execution time and energy consumption values calculated and the efficiency determined for each method level model against the local execution. Firstly, the Table 5.14 consist of the prototype application, the developed model, the average local execution time and remote execution time, and the efficiency calculated. Similarly, Table 5.15 shows, the energy consumption comparison; the average local energy and offloaded energy calculated and efficiency observed for all the three method level offloading approaches.

5.9.1 Efficiency Comparisons of Execution Time

Table 5.14 consist of the results and efficiency comparisons of previous method level offloading solutions against the proposed lightweight REST-Offload Model. It included the prototype application developed for testing. Based on the observed average values of local execution and remote executions the efficiency calculated. Kosta *et al.*, (2009) developed N-queen puzzle for puzzling numbers. The average local execution time of the puzzle game is 15s while the same offloaded to remote server at distant

Table 5.14 Efficiency Comparison of Execution Time

Framework/Model	Prototype	%Average Ex. Time of Local and Offload Execution(s)		Efficiency in Percentage (%)
		Local	Offload	
ThinkAir Framework (Kosta <i>et al.</i> , 2012)	N-Queen Puzzle	15	9	40%
Cuckoo (Kemp <i>et al.</i> , 2010)	eyeDentify object-recognition application	100	50	50%
DECOF (Shiraz <i>et al.</i> , 2014)	Matrices Multiplication	108	97	11%
REST-Offload	Matrices Multiplication	111	49	56%

cloud and the average execution time observed is 9s. It shows 40 % efficiency of execution time against local execution.

Similarly, Kemp *et al.*, (2010) developed an *eyeDentify* object recognition application. The application tested for both scenarios. The local average execution time observed is 100s while the remote execution time is 50s. It shows 50 % efficiency against the local execution. Likewise, Shiraz *et al.*, (2014) used an application for multiplying two matrices and observed the execution time. The local execution time is 108s while the remote execution time is 97s. It gives 11 % efficiency against the local execution of the same application.

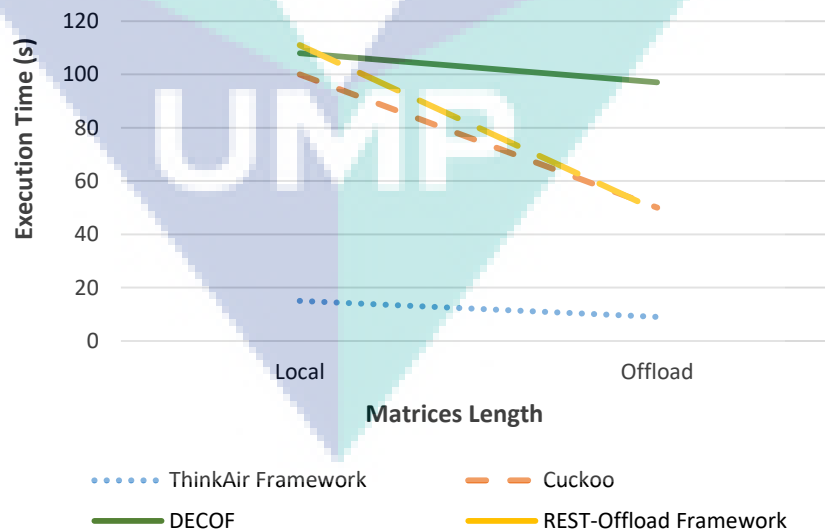


Figure 5.25 Efficiency Comparison of Execution Time

In contrast to the previously developed three method level frameworks/models, the proposed REST-Offload discussed at the end. REST-Offload tested for the same matrix multiplication application. The results of local execution observed is 111s while of remote execution is 49s. The efficiency is 56 % against the local execution. In addition, the comparison of REST-Offload ET results against all the three method level offloading approaches, are shown in Figure 5.25. The calculated efficiency of all the three methods depicts that REST-Offload model is efficient than all the rest approaches with the highest execution time efficiency of 56 %.

5.9.2 Efficiency Comparisons of Energy Consumption

Table 5.15 consists of the results and efficiency comparisons of energy saving against previous method level offloading frameworks and models. Based on the observed average values of local execution and remote executions the efficiency in terms of energy consumption calculated. The energy consumption values gathered for N-queen puzzle of Kosta *et al.*, (2009). The average local energy consumptions of the puzzle game is 78 J while the same offloaded to remote server at distant cloud and the average energy consumption observed is 41 J. It gives 48 % efficiency of energy saving against the local execution.

Similarly, the *eyeDentify* object recognition application of Kemp *et al.*, (2010) shows, 100 J of energy consumption in local execution while 50 J in offloaded execution. It shows 50 % energy efficiency against the local executions. Likewise, matrix multiplication of Shiraz *et al.*, (2014) tested for energy consumption during local execution and it gives the consumptions 91 J while the same execution at remote is 65 J. The efficiency derived here is 29 %.

Table 5.15 Efficiency Comparison of Energy Consumption

Framework/Model	Prototype	% Average Energy Consumption of Local and Offload Execution(J)		Efficiency in Percentage (%)
		Local	Offload	
ThinkAir Framework (Kosta <i>et al.</i> , 2012)	N-Queen Puzzle	78	41	48%
Cuckoo (Kemp <i>et al.</i> , 2010)	eyeDentify object-recognition application	100	50	50%
DECOF (Shiraz <i>et al.</i> , 2014)	Matrices Multiplication	91	65	29%
REST-Offload	Matrices Multiplication	45	25	45%

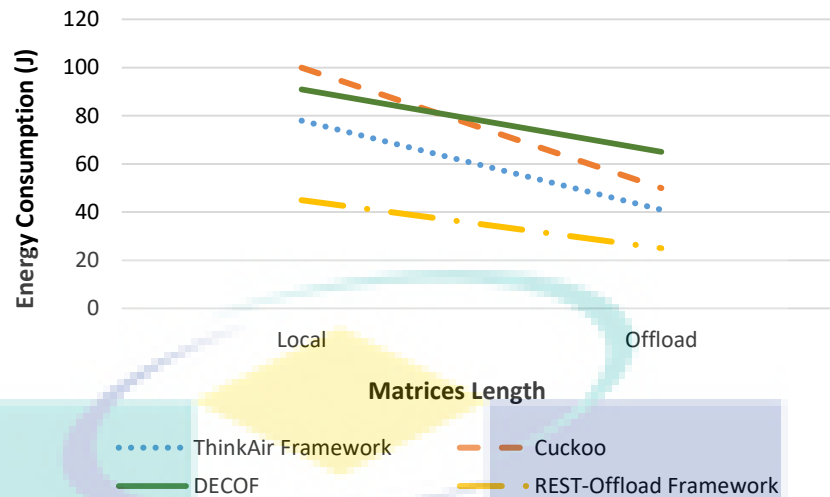


Figure 5.26 Efficiency Comparison of Energy Consumption

On the other hand, REST-Offload tested for a similar matrix multiplication application. The results observed of energy consumption in local execution is 45 J while of remote execution is 25 J. The efficiency is 45 % against the local execution. In addition, the comparison of REST-Offload energy consumption against all the three method level offloading approaches are shown in Figure 5.26. The calculated energy efficiency of three methods depicts that REST-Offload Model falling in the third place in terms of energy efficiency, compare to the rest. The energy consumption of different applications is different based on the level of complexity of application. As the matrix multiplication application is more complex than the N-Queen Puzzle and *eyeDenetiy* application, therefore the energy consumption is slightly higher than both.

5.10 Specification Comparisons of REST-Offload and Existing Approaches

The specification of REST-Offload is compared with the previously developed method level offloading frameworks and models as shown in Table 5.16. Seven main components are selected here to compare and contrast REST-Offload against others. Five main components such as partitioning, data size reduction, service call, predefine parameters selection mechanism and remote execution environment are considered. All the previous works developed the models based on these parameters where REST-offload model addresses the lacking features of the exiting models to propose a different approach for each component.

Table 5.16 Specification Comparison of REST-Offload against Existing Frameworks / Models

Framework / Model	Partition	Service Call	Communication Medium	Remote Server	Predefine Parameter Mechanism	Data Size Reduction	Contribution
Dynamic Compilation and Method Execution (Chen <i>et al.</i> , 2004)	Dynamic	SOAP	3G/4G	Cloud Server	No	Data Compression (e.g. java.util.zip)	Energy Saving
Cuckoo (Kemp <i>et al.</i> , 2010)	Dynamic	SOAP (RPC)	Wi-Fi	Cloud Server	No	n/a	Reduce Energy Consumption
DCOF (Shiraz <i>et al.</i> , 2014)	Static	SOAP (RPC)	Wi-Fi	Cloud Server	No	Deployment of SaaS Model and remote services	Reduce Data size and energy consumption
REST-Offload Proposed REST-Offload Model	Static	REST	Wi-Fi	Surrogate Server	Yes	Replaced XML by JSON	Reduce Energy Consumption

All the existed solutions and the proposed model are going to briefly describe according to the models/framework specifications, as given in Table 5.16.

- a) Partitioning: This point is to know which method is adopted to partition the application before offloading. There are two possible ways to partition application static partition, dynamic partition.
- b) Service Call: It defines what type of service call adopted to delegate the computational intensive tasks. It could be SOAP, REST or RPC.
- c) Communication Medium: It describes the channels used to delegate the task for remote execution. It is important to consider, as a low bandwidth network will hit the communication interface and will waste mobile resource. Ultimately, the battery will drain.
- d) Remote Server: Which type of remote server configured to receive intensive tasks and execute. Server could be a cloud server or cloudlet. Cloud server resides at multi hop distance which increases the communication time and therefore increases RTT.
- e) Predefine Parameters Mechanism: This component defines the basic parameters to include in taking decision before offload. Most of the

frameworks/models lacks in considering this component, therefore the ultimate results are, sometime not efficient in energy savings.

- f) Data Size Reduction: It is very important to reduce the data size before offload the task. If the communication data size is huge, it needs more time to offload.
- g) Strategy for energy offloading; It defines the model strictly goal that is, the model developed to save energy or to achieve any other goal.

To address the lacking (dynamic partitioning of application, huge communication data size, remote service call and longer RTT) of previously developed models and frameworks this study proposed a lightweight computational offloading model. REST-Offload deployed five distinct components to counter the gaps of those frameworks/models. Further, unlike the mentioned method level computational offloading approaches, this research study is focus to understand first where and how the energy drains. Based on the observation of power analysis and critical review of the existing models, a lightweight method level REST-Offload model proposed. In order to reduce the consumption of local resources the novel dynamic application partitioning technique indulged which carries some static features.

Further, RTT reduction is considered the main focus because increasing RTT has a key effect on battery consumption (Cuervo *et al.*, 2010). Also by Teka *et al.*, (2004), minimizing the RTT between mobile device and remote server results in increasing the benefits of computational offloading. Therefore, remote execution environment is configured at a single hop distance which eliminates the long run RTT. Secondly, REST deployed as a service call carrier protocol which replaces SOAP. REST is lightweight and eliminates the XML data carrier by JSON. JSON is easy to parse and easy to read. JSON is smaller in size than XML and hence it reduces the data size to be offloaded. An additional component deployed for considering the predefine parameters before offload. As computational offloading always not energy efficient, it was observed by the experimental results of analysis of power consumption of mobile devices. Therefore, an algorithm proposed in order to select the best possible time based on the available bandwidth, availability of remote server, computations required and available battery level to offload. Hence, by reducing the communication data size, deploying a server at single hop, static partitioning of application and REST as a service call, the REST-Offload model is expected to

produce efficient results compare to the previous developed method level computational offloading models.

5.11 Threats to Validity

Empirical research sometime encounters many threats to validity which leads to unwarranted conclusions. In this research sufficient efforts have been taken to minimize those threats. Further, the implementation of this study was done based on three main experiments Local Execution, Traditional Offloading and REST-Offload. All three the experiments are evaluated based on Execution Time (ET) and Energy Consumption (EC).

Prior to discuss about the threats, it is important to recall the experiments setup here. First Experiment: Local Execution where the application is thoroughly executed in the mobile device. Second Experiment: Traditional Offloading where the experiments are designed based on offloading the intensive tasks to multi-hop cloud server using SOAP techniques. Lastly, REST-Offload is the proposed setup where the intensive tasks are offloaded to a single-hop server using REST techniques.

The validity threats to each set of experiment are arranged as: Firstly, the *Usage Scenario Threats* which perhaps varies the results while running the same simulated tool in all the three experiments. Secondly, the *DuTs Specification Threats* which if change may possibly change the results in all three experiments. Thirdly, *Single-hop Surrogate Threat*, which affect REST-Offload only. Lastly, *EC and ET Estimation Threats* where a Stopwatch used to observe the ET and a Power Meter used to observe EC in all three experiments.

5.11.1 Usage Scenario Threats

Power consumption strongly relates to the using location, different age groups, different generation of Smartphones, climate and geographical regions. Moreover, the energy consumption of different mobile components also relates to the user activity, for instance, a user playing video game offline will hit the CPU and LCD only in term of consumption while an online video game player will hit the Wi-Fi as main energy

drainer. These are some of the factors which affect battery consumption. In the experiment design phase of this research, the mobile device used to be in a single physical location. Further, the usage scenario varied with different age groups. It is established that young age group's usage timing and the choice of interactions with mobile device are different than the elder group of users and it inevitably affect the consumptions level. This level of threats probably affects the results in all three experiments.

5.11.2 DuTs Threats

The Devices under Test (*DuTs*) threats comprises of mobile device's specification threats and battery's specification threats. The mobile devices selected are *Samsung Galaxy A5* and *Asus Zenfone5*. The reason of running the simulations on two different devices was to counter the specification differences and the observed variations of results. Likewise, battery condition of each mobile device needs to investigate, as if the condition of batteries change due to calendar fade or cycle fade, it affects the consumption level.

5.11.2.1 Mobile Device's Specifications Threats

In fact, mobile devices manufactured by different companies carrying considerable differences. The differences are in terms of power consumption owing to different versions of operating systems and features incorporated in different models. Few of the mobile components like LCDs, Air Interfaces, Sensors, and Audio/Video Calls are considered power-intensive in various analysis and experiments. Few of the researchers established screen brightness is one of the key factors which drain battery too fast, such as, display screens of mobile systems usually consume a considerable amount of energy.

The display along with (backlight, touch screen and LCD panel) consumes about 400 mW, which is one of the energy-intensive component. This research also established that the contents displayed on screen affects the total LCD energy consumption such as 33.1mW energy consumption with white screen while 74.2mW with a black screen. Therefore, the screen brightness level and the background graphics

are kept constant throughout the experiments. Moreover, speakers, touch key's light, vibration, sensors, system sounds and network types are the adjustable parameters which may considerably change the battery level of consumption. Hence, all these parameters are kept constant while offloading the tasks to surrogates.

GSM, 3G/4G and Wi-Fi: A faster medium always reduces consumption compare to a slow transmitting medium. The GSM voice services are 46 % better in energy saving compare to UMTS (3G) networks. However, 3G+ (4G) technologies are more energy efficient for transmitting big volumes of data. Selection of transmission medium affects the ET and EC results, this research therefore opted the communication medium to be W-Fi in order to reduce the transmission time of data.

As, mobile device is the main component in each type of experiment, therefore the mobile specifications threats are expected to affect all three type of experiments.

5.11.2.2 Mobile's Battery Specifications Threats

Amongst other threats to validity, battery specification is one of the main threat which likely affect the battery consumption level. Some specification threats of the battery are:

Battery Model: Non-removable, 2300mAh: In the experiments 2300mAh and 2400mAh batteries investigated. The mAh is taken as a parameter which change device to device. More mAh (Milliamp per Hour) means more energy the battery can supply on a full charge and more energy supply is directly proportional to a longer battery life, although more current the battery produces, the more voltage across the internal resistor drops according to Ohm's law ($V=IR$). Thus, the higher the voltage the more charge to consume and it will therefore change the consumption level.

Battery Health: A battery is known to be in good health if it stays active and keep the systems running till sufficient hours during usage, or if it satisfies the mentioned working hours in the device's manual. Two factors deteriorate the performance of battery, one is time and the other is usage. The performance weakening over time is called "Calendar Fade", while the performance deterioration with usage is known as

"Cycle Fade". Furthermore, the lifespan or the battery calendar life is the elapsed time before the battery become useless whether it is in use or not. The batteries used in this research were brand new in one device and used one in the other device. The power consumption of both the devices is therefore slightly different from each other. Hence, battery health is important which may vary the consumption level.

Battery Temperature: One of the main factors influencing the battery calendar life is the battery temperature. Lithium Ion batteries perform poorly if it gets warm or stays warm, leaving to a damaging affect. Having a protective case on device does not allow the heat to escape and to decrease battery temperature. It will then result in cell oxidation which shrinks the capacity and shortening battery's lifespan. Once the battery is damaged by heat, the capacity cannot be restored. Smartphone devices are intended to perform well in an extensive variety of encompassing temperatures, with 62° to 72° F or 16° to 22° C as the perfect safe place. It is particularly imperative to abstain the device from surrounding temperatures higher than 95° F 35°C, where the battery capacity can permanently be damaged. In case of this research the mobile device used in optimal temperature between 16° to 22° C to counter the threat of battery temperature.

Battery Voltage: Lithium-Ion batteries affect from low voltage. It is essential to partially charge or drain the batteries like from 20 % to 90 % than to fully charge and fully drain. Complete charge or fully draining affect the cycle life of battery, which shrinks and reduces the overall efficiency. Additionally, the relationship between voltage and current produced by a battery has no affect together on the amount of energy either the values increases or decreases by the same inverse ratio. Such as, high voltage and low current equals to low voltage and high current. Power in watts is still the same. For example, Battery A: $3600\text{mAh} * 3.7\text{V} = 13\text{Wh}$ and Battery B: $3200\text{mAh} * 4.3\text{V} = 13\text{Wh}$. If a Smartphone consumes 10W per hour it will work 1 hour and 18 minutes with battery A or B. The consumption of mobile device may change due to any of the above battery related parameters.

As both the *DuTs*, which were equipped with Li-Ions batteries, used to conduct the experiments, therefore, this threat is common to all three type of experiments. To counter the threat, the simulation ran 20 consecutive times in order to acquire the mean consumption level which is expected to be closer to the real consumptions.

5.11.3 Single-Hop Surrogate Threats

The single hop surrogate is the modified concept of existing cloudlet. The hop distance would affect RTT. This had been analyzed by Aiguo *et al.*, (1998) who stated in their findings that if hop count increased, the packets would have to go through many routers. At each router, the packets would have to consume a certain amount of time to be routed for the next router and this would be repeated continuously until reaching the destination. Thus, at each router, packet delay would occur and this would increase the overall delay as the hop count increase.

The multi-hop which consists of unlimited hops is the initial concept where the distant cloud server has to serve as a remote computer. The cloudlet concept presented by Satyanarayanan *et al.*, (2009) has brought the cloud closer to the computing environment which is multi-hop (limited) and thus reduced RTT. Single hop, is the modified concept of this research which brings the computing to a single hop and reduces RTT further. Due to changing of geographical position and distance between mobile device and the counterpart server, the RTT varies which is considered threat to validity in this research. This threat relates to the experiments of REST-offload only, where the intensive tasks are delegated for execution to a single hop server.

5.11.4 Energy and Time Estimating Tools Threats

To estimate the EC and ET, there are different tools available. Some tools are in the form of software while some are off-target hardware tools. Using different tools possibly grasp slightly different values for the same data set. The EC and ET threats are discussed as:

5.11.4.1 Energy Consumption Estimation Threats

PowerTutor, which is an open source built in software power estimating tool for Android devices available in Google Play Store. It uses information about the power discharge rate of the voltage-curve to calculate the power consumption. As it works manually, such as, with each instance of the experiment it needs to start by clicking run and stop by clicking break. The calculated time between stop and start is unpredictable

due to possible occurrence of few milliseconds difference. Moreover, *PowerTutor* runs in the background and log data on power utilization for each application by combining all the hardware power modes. While running in the background the power consumption of the tool itself sometime affect the estimated power consumptions. Instead to use the software estimating tool an off-target device works more accurate.

Therefore, an off-target Monsoon Power Monitor was used to observe better results and exclude the milliseconds difference which is usual to happen in *Power Tutor*. The *Monsoon Power Monitor* application provides the popular off-target power consumption estimation. It is also capable of measuring the current, voltage and power and then connected to a special Monsoon Power Application (Computer Software) which gives control over power data and collect and display the data in the form of a graph automatically. As, *Monsoon Power Monitor* used to observe the power consumption during each experiment therefore, this threat relates to all three experiments.

5.11.4.2 Execution Time Estimation Threats

To record the ET of each offloading task a stop watch *TimeLeft* used. It ran with each offload concurrently. There is a possibility of fractional difference of ET in each attempt due to manual use, while recording the ET. To surface the ET difference in results, each task is therefore run approximate 20 times and calculated the mean ET. Further, ET is one of the main parameter to evaluate in all the experiments, therefore, this threat is common to all three type of experiments.

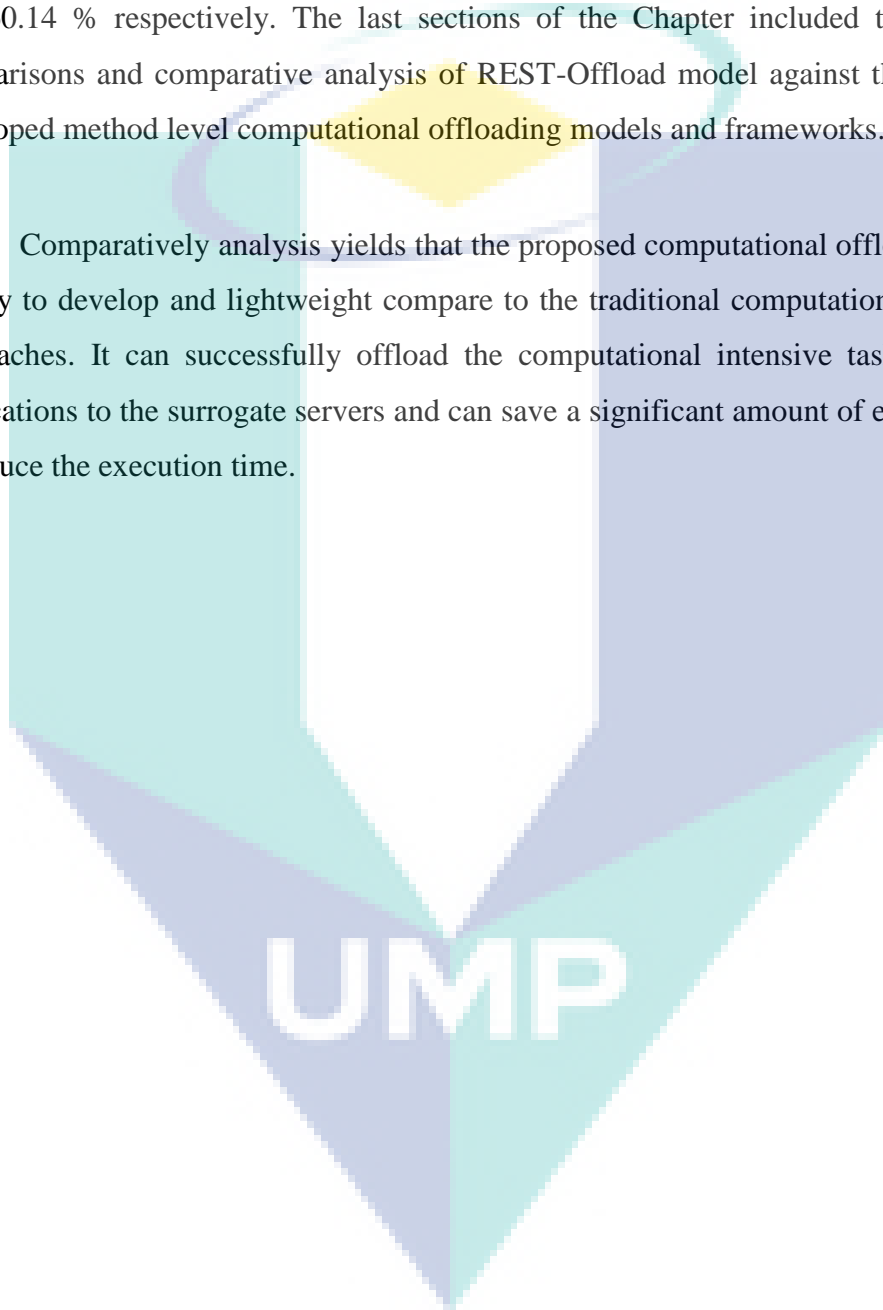
5.12 Summary

The two developed components; Local Execution and Traditional Offloading of the prototype application created to generate bench mark for the proposed solution REST-Offload. The focus was to check the Execution Time (ET) and Energy Consumption (EC) of the prototype application which generates two random matrices at mobile device, multiplies the matrices and displays the resulted matrix at UI. The prototype application ran in all three scenarios Local Execution, Traditional Offloading

and REST-Offload. The ET efficiency of REST-Offload against the benchmarks Local Execution and Traditional Offloading is 54.63 % and 62.01 % respectively.

If consider the energy consumption cost, the EC efficiency of the proposed REST-Offload model against both the benchmark Local Execution and DCOF is 43 % and 60.14 % respectively. The last sections of the Chapter included the efficiency comparisons and comparative analysis of REST-Offload model against the previously developed method level computational offloading models and frameworks.

Comparatively analysis yields that the proposed computational offloading model is easy to develop and lightweight compare to the traditional computational offloading approaches. It can successfully offload the computational intensive tasks of mobile applications to the surrogate servers and can save a significant amount of energy as well as reduce the execution time.



CHAPTER 6

CONCLUSION

6.1 Overview

This chapter re-evaluates the problem statement, research objectives and reviews the contribution of the research. It also reflects the limitations and future research work of this study. The chapter is organized into six main sections. Section 6.2 consists of the discussion part which includes goal of the research and re-assessment of the research objectives. Section 6.3 presents the revisiting of objectives. Section 6.4 presents overview of the contributions. Section 6.5 describes the scalability of the research. Section 6.6 discusses the limitations and future research work.

6.2 Discussion

The focus of this research was to: curtail the use of extra resources of mobile device during offloading; to reduce the size of communication data; and to utilize the closest computing environment in order to curtail battery consumption. Several research frameworks/models have been developed in the past with the intension to reduce the computational load of mobile devices through computational offloading. In principal, it is true that computational offloading may free the processor of mobile device from processing the given task. However, it should be kept in mind that mobile device has to spend a significant amount of energy while establishing the connection to remote server, send the request and offloading data through a fluctuating bandwidth. It also has to wait until the result comes back to the device. If the energy consumption during this whole process is less than the one consumed in processing the task locally, then computational offloading will be an energy saving solution for mobile devices. Unfortunately, many of

the previously developed models failed to encompass the three basic parameters B, C, and D, hence have become energy harvesting approach instead of energy saving.

This research is all about addressing the limitation of mobile devices through a new lightweight method level computational offloading model. The previous computational offloading frameworks/models were based on VM migration, whole application migrations and traditional method level offloading. The traditional offloading frameworks based on either static or dynamic partitioning while XML used as a carrier file, have been simulated and observed experimentally. The experiments have been conducted in order to know which component of the existing approaches causes overhead computation and hence causing the offloading to be a resources intensive approach. In order to achieve the goal, a lightweight method level computational offloading model has been designed and developed.

6.3 Revisit of the Research Objectives

There are three main objectives which collectively accomplish the process to reach the goal; a lightweight method level computational offloading model. The first objective is, **to develop a novel dynamic application partitioning method, for reduction of computations and handling of the dynamic network changes**. To achieve the first objective, an analysis has been conducted and the cost estimation models of existing static and dynamic application partitioning techniques have been developed. Based on the analysis, a novel approach was proposed, which is to combine the positive features of both static and dynamic. The existing static and dynamic techniques are concluded to be inefficient in reducing RTT and hence cannot reduce the power consumptions. Static approach does not need any additional computations due to compile time partitioning. Dynamic partitioning however, is good in bringing automation and coping the dynamic changes, yet increases computations due to continuous changing of the execution pattern.

Therefore, this research proposed a novel dynamic technique which is dynamic in nature, however, inherits some static features. In the proposed partitioning technique, the application is partitioned statically by a manual annotation similar to the static partitioning approach. In addition, an algorithm is proposed to deploy a mechanism for

the selection of predefined parameters. This algorithm copes with the changes that occur in the environment dynamically. It is important because offloading every time by using any bandwidth network to any status of remote server is not energy efficient. The novel dynamic partitioning is a middle level approach, which costs more than the static technique while less than the dynamic technique, in terms of Execution Time (ET) and Energy Consumption (EC).

The second objective of the research is, **to design the cloud server/surrogate machine for the execution of intensive tasks to reduce long run RTT**. The previous research mostly configured the remote server at distant cloud. It is a fact, that cloud environment is resources rich. The server at cloud is equipped with powerful resources, however, the delegation of tasks to distant cloud increases RTT. It is also established by the previous researches, that long run RTT drains power. To reduce RTT, this research modifies the concept of cloudlet presented by Satyanarayanan *et al.*, (2009). Satyanarayanan proposed Cloudlet where the distant cloud is brought closer to the mobile environment in order to reduce RTT. As the hop count decreases, it decreases the RTT, yet the Cloudlet is still situated in multi hop distance. This research has designed the cloudlet/surrogate to be more closer to the mobile device. A surrogate machine at a single hop distance was configured next to IEEE 802.11 access point. The surrogate machine is further connected to cloud to fetch any necessary data from the internet. Hence, the RTT is reduced further and this has ultimately reduced the power consumption during delegation of tasks to remote (surrogate) computer.

The last objective of the study is, **to develop and evaluate a lightweight offloading method for size of communication data reduction**. The existing method for offloading the task to remote executions normally become intensive due to increasing size of communication data and due to additional computation required at mobile device. In the previous research works, the SOAP offload trigger was normally used. SOAP offload supports XML file to trigger with offload for carrying data from client device to remote servers. With the analysis of SOAP testing, as a carrier protocol, it is observed that SOAP is heavy to execute and complex to parse. Although SOAP and XML based offloading is considered more secure compared to the RPC, RIC and REST, it however increases the communication data and hence increasing computations at mobile device.

Therefore, in order to reduce the data size during communication at both sides of mobile device and remote server, a new lightweight offloading technique REST-offloading is proposed, based on the analysis carried out between many different offloading protocols. Normally, XML and JSON have been used in many different problems, however, a technique based on the combination of REST, JSON and WSDL is proposed here to reduce complexity, computations and size of communication data. The intention of using REST and JSON here is due to the low computing ability of mobile device. This technique reduces the communication data size and cuts the computations and RTT. REST is simple to write due to HTTP and some CRUD (Create, Read, Update, and Delete) operations. Nevertheless, REST is less secure as compared to SOAP, as REST inherits the security from the underlying transport while SOAP defines its own WS-Security (Web services Security).

After accomplishing each objective, the model was then implemented in lab environment by developing a prototype application. The prototype application consists of Local Execution, Traditional Computational Offloading Method and the proposed REST-Offload. Each component of the prototype consists of matrix multiplication where two random matrices were taken as input by the application, the matrices were then offloaded to surrogate/cloud server to multiply and the result was taken back on the mobile screen. It was tested for ET in milliseconds (ms) and power consumption in Joules (J) as discussed in Chapter 4 (Section 4.6). The results of three pre-defined scenarios which are Local Execution, Traditional Computational Offloading Method and the proposed REST-Offload Method were collected and compared to validate the proposed lightweight method level computational offloading model. The REST Offload is significantly useful compared to both local execution and traditional methods and it has saved about 50% ET and reduced approximately 38% energy consumption compared to ET and EC of task locally executed.

The comparative analysis has been conducted in two ways. Firstly, the efficiency of REST-Offload was compared to the rest in terms of ET and EC. Secondly, the REST-Offload model was compared by specification against the three closely related method level computational framework/models. It has been observed from the efficiency comparison results, that REST-Offload is 56 % efficient in ET against local

execution, which is highest in efficiency compared to all the three frameworks/models. Similarly, in terms of the energy efficiency of REST-Offload, it shows that, REST-Offload is 45 % efficient in energy consumption against local execution of the task. The efficiency of REST-Offload is higher than ThinkAir and Cuckoo, while less than DCOF.

6.4 Contributions of the Research

This research ought to contribute sufficiently to the sphere of research which are further discussed in this section. This research contributes in the existing method level computational offloading frameworks/models in order to improve the existing frameworks/models to be a lightweight solution. The main contribution of this research is A Model for Power Efficiency of Mobile Device through Lightweight Method Level Computational Offloading. This model addresses the overhead computation of traditional method level frameworks/models. It provides a lightweight solution for executing the computational intensive tasks at surrogate server. The model consists of two main components; client component and server component. The client component is deployed at client device while server component is configured at surrogate server. Both are synchronized to communicate for execution of intensive tasks. Sub contributions of the research are:

There are few sub contributions for the achievement of main contributions. First sub contribution is the development of novel dynamic application partitioning technique. It eliminates the overhead computation by inheriting the static feature of existing techniques. It also carries the dynamic features by deploying an algorithm for the selection of pre-defined parameters. This technique reduces the additional computation and copes with the dynamic changes in execution environment at the same time. The second sub contribution is the development of single hop surrogate model. Based on the existing cloud computing models and cloudlet, a single hop surrogate model is developed which reduces the long run RTT. The third sub contribution of the research is the development of lightweight REST-Offloading technique. This technique eliminates the additional computational and communication overhead due to easy configuration and of reduced data size. It costs less than the available techniques in term of ET and EC.

There are two supportive contributions. First, a prototype REST-Offload application is developed. REST-Offload implements the proposed algorithm and lightweight method level computational offloading model. REST-Offload has been used in real mobile cloud environment to observe the intensity of application processing in terms of ET and EC. Second, the proposed algorithm for the selection of the predefined parameters. Offloading in any circumstances is not always energy efficient, therefore a monitoring mechanism is needed to check all the predefined parameters and take right decision before delegating intensive components. For this purpose, a lightweight computational offloading algorithm has been proposed, to monitor and verify all the predefined conditions before offloading. It also ensures to statically partition the application and delegate the computational intensive components only which releases the resources management and communication overhead.

6.5 The Scalability of Proposed Model

REST-Offload is scalable and can adopt changes due to mobility or changing network topology. The mechanism defined for the selection of predefined parameters supports the model to work in heterogeneous kind of environment. Although there are some limitations such as 3G network being excluded as a medium due to its limited bandwidth, the model can still be used to operate in the environment of 4G, WiMAX and Wi-Fi. The surrogate needs to be active and available with each access point. In case of the absence of surrogate server, the model turns the application to run in local environment.

6.6 Limitations and Future Work

Conclusively, this research has studied different approaches towards the conservation of energy in resources constrained mobile devices. The research is first carried out by examining the fundamental energy-related issues with respect to executing the tasks on local mobile device. Subsequent intention was to resolve the energy consumption issues with local processing through a new proposed and developed lightweight method level offloading model. Secondly, a thorough study of the computational offloading frameworks took place and the results were compared with local consumption of the device while task executed locally. From the critical analysis

and observation of the results, a gap has been found, which is, minimizing the resources management and transmission overhead can save a significant amount of energy. This research started the work to fill in that gap.

Therefore, the focus of this research is to address the limitations of mobile devices through lightweight method level computational offloading in Mobile Cloud Computing. This research emphasizes to minimize the additional resources utilization of mobile computing devices in traditional computational offloading frameworks. However, this research lacks in addressing issues related to security and services availability through low bandwidth networks. The resources scarcity or mobile inefficiency in computations and several other limitations exist in mobile devices due to portability and mobility are required to be further addressed in the ongoing research. Nevertheless, the scope of this work is focused to address the issues related to ET and battery consumption. This research also lacks in addressing the issues related to mobility and services availability in case of limited bandwidth. The medium of communication used is Wi-Fi and by research observations, the limited bandwidth networks are not suitable for offloading due to longer RTT, therefore 3G as an offloading medium has been avoided here. Security and consistency in parallel execution between client device and remote servers are inadequate for heterogeneous mobile devices. The future research will include addressing the low bandwidth issues and to make computational offloading services available for any kind of mobile device anywhere. The security concerns will also be the focus to make data processing safe at remote location.

REFERENCES

- Aamazon S3 (March 14, 2016). Retrieved from ([http:// www.amazon.com](http://www.amazon.com)).
- Abolfazli, S., Sanaei, Z., & Gani, A. (2012). Mobile cloud computing: A review on smartphone augmentation approaches. *The 1st International Conference on Computing, Information Systems, and Communications*, Singapore.
- Abolfazli, S., Sanaei, Z., Alizadeh, M., Gani, A., & Xia, F. (2014). An experimental analysis on cloud-based mobile augmentation in mobile cloud computing. *IEEE Transactions on Consumer Electronics*, 60(1), pp. 146-154.
- Adrees, M. S., Omer, M. K. A., & Sheta, O. E. (2016). Cloud Computing architecture for higher education in the third world countries (republic of the sudan as model).
- Ahmed, E., Gani, A., Sookhak, M., Ab Hamid, S. H., & Xia, F. (2015). Application optimization in mobile cloud computing: Motivation, taxonomies, and open challenges. *Journal of Network and Computer Applications*, 52, pp. 52-68.
- Anand, A., Manikopoulos, C., Jones, Q., & Borcea, C. (2007). A quantitative analysis of power consumption for location-aware applications on smart phones. *The 2007 IEEE International Symposium on Industrial Electronics*, Vigo Spain.
- Ardito, L., Procaccianti, G., Torchiano, M., & Migliore, G. (2013). Profiling power consumption on mobile devices. *The 3rd International Conference on Smart Grids, Green Communications and IT Energy-aware Technologies*, Lisbon Portugal.
- Balan, R., Flinn, J., Satyanarayanan, M., Sinnamohideen, S., & Yang, H.-I. (2002). The case for cyber foraging. *The 10th Workshop on ACM Sigops European workshop*, pp. 87-92, France.
- Balan, R. K. (2004). Powerful change part 2: reducing the power demands of mobile devices. *IEEE Pervasive Computing*, 3(2). pp. 71-73.
- Balan, R. K., Gergle, D., Satyanarayanan, M., & Herbsleb, J. (June 11-14, 2007). Simplifying cyber foraging for mobile devices. *The 5th International Conference on Mobile Systems, Applications and Services*, pp. 272-182, New York USA.
- Balan, R. K., Satyanarayanan, M., Park, S. Y., & Okoshi, T. (2003). Tactics-based remote execution for mobile computing. *The 1st international Conference on Mobile Systems, Applications and Services*, pp. 271-286, San Francisco, CA, USA.

- Balasubramanian, N., Balasubramanian, A., & Venkataramani, A. (2009). Energy consumption in mobile phones: a measurement study and implications for network applications. *The 9th ACM Sigcom Conference on Internet Measurement Conference*, pp. 280-293, Chicago USA.
- Begum, Y. M., & Mohamed, M. M. (April 12-14, 2010). A DHT-based process migration policy for mobile clusters. *The 7th International Conference on Information Technology: New Generations (ITNG)*, pp.12-14 DOI: 10.1109/ITNG.2010.157, Las Vegas, Nevada, USA.
- Bertozzi, D., Raghunathan, A., Benini, L., & Ravi, S. (2003). Transport protocol optimization for energy efficient wireless embedded systems. *The Conference on Design, Automation and Test in Europe*, Muenchen, Germany.
- Bheda, H. A., & Lakhani, J. (2013). Application Processing Approach for Smart Mobile Devices in Mobile Cloud Computing. *International Journal of Advanced Research in Computer Science and Software Engineering*, 3(8).
- Buyya, R., Yeo, C. S., Venugopal, S., Broberg, J., & Brandic, I. (2009). Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems*, 25(6), pp. 599-616.
- Carroll, A., & Heiser, G. (2010). An Analysis of Power Consumption in a Smartphone. *The Annual Technical Conference on Usenix 2010*, pp. 21-21, Boston, USA.
- Chen, G., Kang, B.-T., Kandemir, M., Vijaykrishnan, N., Irwin, M. J., & Chandramouli, R. (2004). Studying energy trade offs in offloading computation/compilation in java-enabled mobile devices. *IEEE Transactions on Parallel and Distributed Systems*, 15(9), pp. 795-809.
- Chu, H.-h., Song, H., Wong, C., Kurakake, S., & Katagiri, M. (2004). Roam, a seamless application framework. *Journal of Systems and Software*, 69(3), pp. 209-226.
- Chun, B.-G., Ihm, S., Maniatis, P., & Naik, M. (2010) Clonecloud: boosting mobile device applications through cloud clone execution. arXiv preprint arXiv:1009.3088. v(2).
- Chun, B.-G., Ihm, S., Maniatis, P., Naik, M., & Patti, A. (2011). Clonecloud: elastic execution between mobile device and cloud. *The 6th International Conference on Computer Systems*, pp. 301-314, Universitätsaula Salzburg, Austria.
- Chun, B.-G., & Maniatis, P. (2009). Augmented Smartphone Applications through Clone Cloud Execution. *The 12th conference on Hot Topics in Operating Systems*. pp. 8-8, Monte Verità, Switzerland.
- Chun, B.-G., & Maniatis, P. (15-18 June 2010). Dynamically partitioning applications between weak devices and clouds. *The 1st ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond*, San Francisco, USA.

- Chun, W. (June 2016). What is google app engine? Euro Python 2011. Retrieved from (<https://ep2013.europython.eu/conference/talks/google-app-engine-best-practices-latest-features>).
- Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update 2014–2019 White Paper (June 2016). Retrieved from (http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white_paper_c11-520862.html).
- Claybrook, B. (July 2016). Mobile cloud apps vs. native apps: The developer's Perspectives. Retrieved from <http://searchcloudapplications.techtarget.com/feature/Mobile-cloud-apps-vs-native-apps-The-developers-perspective>).
- Cuervo, E., Balasubramanian, A., Cho, D.-k., Wolman, A., Saroiu, S., Chandra, R., & Bahl, P. (2010). MAUI: making smartphones last longer with code offload. *The 8th International Conference on Mobile Systems, Applications, and Services*, pp. 49-62, Kraków, Poland.
- Cui, Y., Ma, X., Wang, H., Stojmenovic, I., & Liu, J. (2013). A survey of energy efficient wireless transmission and modelling in mobile cloud computing. *Mobile Networks and Applications*, 18(1), pp. 148-155.
- De Giorgio, T., Ripa, G., & Zuccalà, M. (2010). An approach to enable replacement of SOAP services and REST services in lightweight processes. *Current Trends in Web Engineering*, pp. 338-346.
- De Lara, E., Wallach, D. S., & Zwaenepoel, W. (2001). Puppeteer: Component-based Adaptation for Mobile Computing. *The 3rd Conference on Usenix Symposium on Internet Technologies and Systems*, vol-3, pp. 14-14, San Francisco, California, USA.
- Dinh, H. T., Lee, C., Niyato, D., & Wang, P. (2013). A survey of mobile cloud computing: architecture, applications, and approaches. *Wireless Communications and Mobile Computing*, 13(18), pp. 1587-1611.
- Dou, A., Kalogeraki, V., Gunopulos, D., Mielikainen, T., & Tuulos, V. H. (2010). Misco: a mapreduce framework for mobile systems. *The 3rd International Conference on Pervasive Technologies Related to Assistive Environments*, Samos, Greece.
- Ewens, W. J., & Grant, G. R. (2001). Computationally intensive methods. *In Statistical Methods in Bioinformatics*, pp. 349-363, New York USA.
- Fernando, N., Loke, S. W., & Rahayu, W. (2013). Mobile cloud computing: A survey. *Future Generation Computer Systems*, 29(1), 84-106.

- Ferreira, Denzil, Anind Dey, and Vassilis Kostakos (2011). "Understanding human-smartphone concerns: a study of battery life." *Pervasive Computing*, 19-33.
- Fielding, R. T. (2000). Architectural styles and the design of network-based software architectures. *Doctoral Dissertation Architectural Styles and the Design of Network-based Software Architectures University of California, Irvine*.
- Flinn, J., Park, S., & Satyanarayanan, M. (2002). Balancing performance, energy, and quality in pervasive computing. *The International Conference on Distributed Computing Systems*, pp. 217, 2002, Vienna, Austria.
- Flinn, J., & Satyanarayanan, M. (1999). Energy-aware adaptation for mobile applications. *The 17th ACM Symposium on Operating Systems Principles*. Vol. 33(5), pp. 48-63, Charleston, South Carolina, USA.
- 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), pp. 80-88.
- Frattasi, S., Fathi, H., Fitzek, F. H., Prasad, R., & Katz, M. D. (2006). Defining 4G technology from the users perspective. *IEEE Network*, 20(1), pp. 35-41.
- Garlan, D., Siewiorek, D. P., Smailagic, A., & Steenkiste, P. (2002). Project aura: Toward distraction-free pervasive computing. *IEEE Pervasive Computing*, 1(2), pp. 22-31.
- Giurgiu, I., Riva, O., Juric, D., Krivulev, I., & Alonso, G. (2009). Calling the cloud: enabling mobile phones as interfaces to cloud applications. *The 10 International Conference on Distributed Systems Platforms and Open Distributed Processing*, pp. 83-102, Urbana, IL USA.
- Goraczko, M., Liu, J., Lymberopoulos, D., Matic, S., Priyantha, B., & Zhao, F. (2008). Energy-optimal software partitioning in heterogeneous multiprocessor embedded systems. *The 45th Annual Design Automation Conference*. pp. 191-196, Anaheim, CA, USA.
- Goyal, S., & Carter, J. (2004). A lightweight secure cyber foraging infrastructure for resource-constrained devices. *The 6th IEEE Workshop on the Mobile Computing Systems and Applications*. pp. 186-195, Cumbria UK.
- Griera Jorba, M. (2013). Improving the reliability of an offloading engine for Android mobile devices and testing its performance with interactive applications, *Master Thesis*, Freie Universität Berlin.
- Gu, X., Nahrstedt, K., Messer, A., Greenberg, I., & Milojevic, D. (2004). Adaptive offloading for pervasive computing. *IEEE Pervasive Computing*, 3(3), pp. 66-73.

- Guillaume. (2016). How heat and loading affect battery Life. Retrieved from: <http://batteryuniversity.com/learn/article/how-heat-and-harsh-reduces-battery-life> . Accessed Date: July 2016).
- Herrmann, R., Zappi, P., & Rosing, T. S. (2012). Context aware power management of mobile systems for sensing applications. *The 11th ACM/IEEE International Conference on Information Processing in Sensor Networks*, April 16–20, 2012, Beijing, China.
- Huang, D., Zhang, X., Kang, M., & Luo, J. (2010). MobiCloud: building secure cloud framework for mobile computing and communication. *The 5th IEEE International Symposium on Service Oriented System Engineering*, pp. 27-34, Nanjing, China.
- Huerta-Canepa, G., & Lee, D. (2008). An adaptable application offloading scheme based on application behavior. *The 22nd International Conference on Advanced Information Networking and Applications*, GinoWan, Okinawa, Japan.
- Huerta-Canepa, G., & Lee, D. (2010). A virtual cloud computing provider for mobile devices. *The 1st ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond*, San Fransisco USA.
- Hung, S.-H., Shih, C.-S., Shieh, J.-P., Lee, C.-P., & Huang, Y.-H. (2012). executing mobile applications on the cloud: Framework and issues. *Computers & Mathematics with Applications*, 63(2), pp. 573-587.
- Izaki, M. (2002). U.S. Patent No. 6,405,062. Washington, DC: U.S. Patent and Trademark Office.
- Kemp, R., Palmer, N., Kielmann, T., & Bal, H. (2010). Cuckoo: a computation offloading framework for smartphones. *The International Conference on Mobile Computing, Applications, and Services.*, vol-76. pp. 59-69, CA USA.
- Kim, K.-H., Min, A. W., Gupta, D., Mohapatra, P., & Singh, J. P. (2011). Improving energy efficiency of Wi-Fi sensing on smartphones. *The IEEE Proceedings Infocom*, Changhai China.
- 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. *Proceeding of IEEE Conference Infocom*, Florida USA.
- Kravets, R., & Krishnan, P. (2000). Application-driven power management for mobile communication. *Wireless Networks*, 6(4), pp. 263-277.
- Kristensen, M. D. (2007). Enabling cyber foraging for mobile devices. *Proceedings of the 5th MiNEMA Workshop: Middleware for Network Eccentric and Mobile Applications*. Magdeburg University.

- Kristensen, M. D. (2008). Execution plans for cyber foraging. *The 1st Workshop on Mobile Middleware: Embracing the Personal Communication Device*. Article no. 2.
- Kristensen, M. D. (2010). Empowering mobile devices through cyber foraging. *Aarhus University (Ph. D. Thesis)*, Department of Computer Science, Aarhus University, pp. 242.
- Kristensen, M. D. (2010). Scavenger: Transparent development of efficient cyber foraging applications. *The 8th IEEE International Conference on Pervasive Computing and Communications*. Mannheim Germany.
- Kumar, K. (2011). Application-based energy efficient mobile and server computing. *PhD Dissertation*, Purdue University West Lafayette, Indiana.
- Kumar, K., Liu, J., Lu, Y.-H., & Bhargava, B. (2012). A Survey of Computation Offloading for Mobile Systems. *Mobile Networks and Applications*, 18(1), pp. 129-140. Doi: 10.1007/s11036-012-0368-0.
- Kumar, K., & Lu, Y.-H. (2010). Cloud computing for mobile users: can offloading computation save energy? *Computer*, 43(4), pp. 51-56.
- Lai, C.-C., & Ko, R.-S. (2010). Dishes: A distributed shell system designed for ubiquitous computing environment. *International Journal of Computer Networks & Communications*, 2(1), pp. 66-83.
- Liu, J., Kumar, K., & Lu, Y.-H. (2010). Trade-off between energy savings and privacy protection in computation offloading. *The 16th ACM/IEEE International Symposium on Low Power Electronics and Design*, Austin TX USA.
- Liu, Q., Jian, X., Hu, J., Zhao, H., & Zhang, S. (2009). An optimized solution for mobile environment using mobile cloud computing. *The 5th International Conference on Wireless Communications, Networking and Mobile Computing*, Beijing China.
- Liu, X., Shenoy, P., & Corner, M. (2005). Chameleon: application level power management with performance isolation. *The 13th Annual ACM International Conference on Multimedia*, Singapore.
- Lu, Y., Li, S., & Shen, H. (2011). Virtualized Screen: A Third Element for Cloud Computing; *Mobile Convergence. IEEE Multimedia*, 18(2), pp. 4-11.
- Mack, C. A. (2011). Fifty years of Moore's law. *IEEE Transactions on Semiconductor Manufacturing*, 24(2), pp. 202-207.
- Magurawalage, C. M. S., Yang, K., Hu, L., & Zhang, J. (2014). Energy-efficient and network-aware offloading algorithm for mobile cloud computing. *Computer Networks*, V(74), pp. 22-33.

- Mell, P., & Grance, T. (2011). The NIST definition of Cloud Computing. *Special Publication (NIST SP)*, pp. 800-145.
- Messer, A., Greenberg, I., Bernadat, P., Milojevic, D., Chen, D., Giuli, T. J., & Gu, X. (2002). Towards a Distributed Platform for Resource-constrained Devices. *The 22nd International Conference on Distributed Computing Systems*, Vienna Austria.
- Metri, G. (2014). Energy efficiency analysis and optimization for mobile platforms. *PhD Dissertation Wayne State University January*.
- Mollick, E. (2006). Establishing Moore's law. *IEEE Annuals of the History of Computing*, 28(3), pp. 62-75.
- Moghimi, M., Venkatesh, J., Zappi, P., & Rosing, T. (2012). Context-aware mobile power management using fuzzy Inference as a service. *The 7th International Conference on Mobile Computing, Applications, and Services*, pp . 314-327, Krakow, Poland.
- Murarasu, A. F., & Magedanz, T. (2009). Mobile middleware solution for automatic reconfiguration of applications. *The 6th International Conference on Information Technology: New Generations*, pp. 1049-1055, Las Vegas, Nevada.
- New semiconductor research may extend integrated circuit battery life tenfold,. (Aug, 2016). retrieved from (<http://phys.org/news/2013-01-semiconductor-circuit-battery-life-tenfold.html>).
- Newton, R., Toledo, S., Girod, L., Balakrishnan, H., & Madden, S. (2009). Wishbone: profile-based partitioning for sensornet applications. *The 6th USENIX Symposium on Networked Systems Design and Implementation*. pp. 395-408. Boston Massachusetts USA.
- Orlando, F. (October 8, 2013). Gartner Identifies the top 10 strategic technology trends for 2014. Retrieved from (<http://www.gartner.com/newsroom/id/2603623>).
- Othman, M., Madani, S. A., & Khan, S. U. (2014). A Survey of Mobile Cloud Computing application models. *IEEE Communications Surveys & Tutorials*, 16(1), pp. 393-413.
- Ou, S., Yang, K., & Liotta, A. (2006). An adaptive multi-constraint partitioning algorithm for offloading in pervasive systems. *The 4th Annual IEEE International Conference on Pervasive Computing and Communications*, pp. 116-125, Pisa, Italy.
- Ou, S., Yang, K., & Zhang, Q. (2006). An efficient runtime offloading approach for pervasive services. *The IEEE Wireless Communications and Networking Conference*. Volume: 4, pp. 3-6, Las Vegas NV, USA.

- Peltonen, E., Lagerspetz, E., Nurmi, P., & Tarkoma, S. (2015). Energy modeling of system settings: A crowdsourced approach. *The IEEE International Conference on Pervasive Computing and Communications*, St. Louis, Missouri, USA.
- Perrucci, G. P., Fitzek, F. H., Sasso, G., Kellerer, W., & Widmer, J. (2009). On the impact of 2G and 3G network usage for mobile phones' battery life. *The European Wireless Conference, 2009*, Aalborg Denmark.
- Perrucci, G. P., Fitzek, F. H., & Widmer, J. (2011). Survey on energy consumption entities on the smartphone platform. *The 73rd IEEE Vehicular Technology Conference (VTC Spring)*, Hungary.
- Prelas, M., Boraas, M., Aguilar, F. D. L. T., Seelig, J.-D., Tchouaso, M. T., & Wisniewski, D. (2016). Potential applications for nuclear batteries and radioisotopes, *Springer International Publishing*. pp. 285-305.
- Qi, H., & Gani, A. (2012). Research on mobile cloud computing: Review, trend and perspectives. *The 2nd International Conference on Digital Information and Communication Technology and its Applications*, Bangkok, Thailand.
- Rachuri, K. K., Mascolo, C., Musolesi, M., & Rentfrow, P. J. (2011). Sociable sense: exploring the trade-offs of adaptive sampling and computation offloading for social sensing. *The 17th Annual International Conference on Mobile Computing and Networking*, pp. 73-84, Las Vegas, Nevada USA.
- Rahman, M., Gao, J., & Tsai, W.-T. (2013). Energy saving in mobile cloud computing. *The IEEE International Conference on Cloud Engineering*, Boston, Massachusetts, USA.
- Ravi, N., Scott, J., Han, L., & Iftode, L. (2008a). Context-aware Battery Management for mobile phones. *The 6th Annual IEEE International Conference on Pervasive Computing and Communications*. pp. 224-233. doi:10.1109/percom.2008.108, Hong Kong.
- Ravi, N., Scott, J., Han, L., & Iftode, L. (2008b). Context-aware battery management for mobile phones. *The 6th Annual IEEE International Conference on Pervasive Computing and Communications*, pp. 17-21, Hong Kong.
- Rim, H., Kim, S., Kim, Y., & Han, H. (2006). Transparent method offloading for slim execution. *The 2006 1st International Symposium on Wireless Pervasive Computing*, Hilton Phuket Thailand.
- Rudenko, A., Reiher, P., Popek, G. J., & Kuenning, G. H. (1998). Saving portable computer battery power through remote process execution. *ACM Mobile Computing and Communications Review*, 2(1), pp. 19-26.

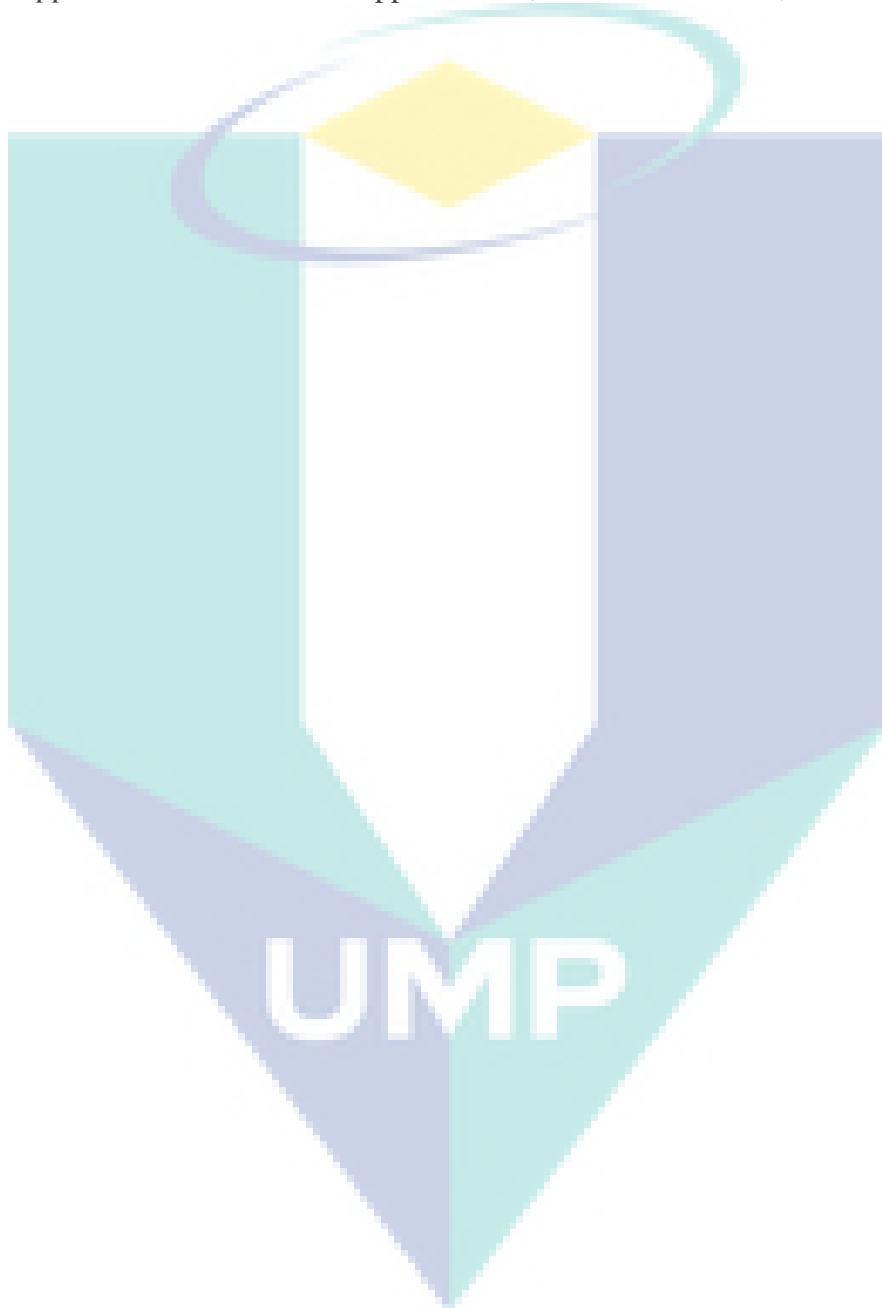
- Saha, D., & Mukherjee, A. (2003). Pervasive Computing: a paradigm for the 21st century. *Computer*, 36(3), pp. 25-31.
- Sathan, D., Meetoo, A., & Subramaniam, R. (2009). Context aware lightweight energy efficient framework. *International Journal of Humanities & Social Sciences*, pp. 64-70.
- Satyanarayanan, M. (1996). Fundamental challenges in mobile computing. *The 15th Annual ACM Symposium on Principles of Distributed Computing*. pp. 1-7, Philadelphia, PA, USA.
- Satyanarayanan, M. (2001). Pervasive computing: vision and challenges. *IEEE Personal Communications*, 8(4), pp. 10-17.
- Satyanarayanan, M., Bahl, P., Caceres, R., & Davies, N. (2009). The case for vm-based cloudlets in mobile computing. *IEEE Pervasive Computing*, 8(4), pp. 14-23.
- Shin, Y., Lee, H.-J., Shin, K., Kenkae, P., Kashyap, R., Seo, D., Millar, B., Kwon, Y., Iyengar, R., & Kim, M.-s. (2013). 28nm high-K metal gate heterogeneous quad-core CPUs for high-performance and energy-efficient mobile application processor. *The International SoC Design Conference*, Busan Korea.
- Shiraz, M., Abolfazli, S., Sanaei, Z., & Gani, A. (2013). A study on virtual machine deployment for application outsourcing in mobile cloud computing. *The Journal of Supercomputing*, 63(3), pp. 946-964.
- Shiraz, M., & Gani, A. (2014). A lightweight active service migration framework for computational offloading in mobile cloud computing. *The Journal of Supercomputing*, 68(2), pp. 978-995. doi:10.1007/s11227-013-1076-7.
- Shiraz, M., Gani, A., Ahmad, R. W., Shah, S. A. A., Karim, A., & Rahman, Z. A. (2014). A lightweight distributed framework for computational offloading in mobile cloud computing. *PloS one*, 9(8), e102270.
- Shiraz, M., Gani, A., Khokhar, R. H., & Buyya, R. (2013). A review on distributed application processing frameworks in smart mobile devices for mobile cloud computing. *IEEE Communications Surveys & Tutorials*, 15(3), pp. 1294-1313.
- Shiraz, M., Gani, A., Shamim, A., Khan, S., & Ahmad, R. W. (2015). Energy efficient computational offloading framework for mobile cloud computing. *Journal of Grid Computing*, 13(1), pp. 1-18.
- Sharifi, M., Kafaie, S., & Kashefi, O. (2012). A survey and taxonomy of cyber foraging of mobile devices. *IEEE Communications Surveys & Tutorials*, 14(4), 1232-1243.

- Shuja, J., Gani, A., Naveed, A., Ahmed, E., & Hsu, C.-H. (2016). Case of ARM emulation optimization for offloading mechanisms in Mobile Cloud Computing. *Future Generation Computer Systems*. <http://dx.doi.org/10.1016/j.future.2016.05.037>.
- Shuja, J., Gani, A., ur Rehman, M. H., Ahmad, R. W., Ahmed, E., Madani, S. A., Khan, M. K., & Ko, K. (2016). Towards native code offloading based MCC frameworks for multimedia applications: A Survey. *Journal of Network and Computer Applications*. DOI: 10.1016/j.jnca.2016.08.021.
- Shye, A., Scholbrock, B., & Memik, G. (2009). Into the wild: studying real user activity patterns to guide power optimizations for mobile architectures. *The 42nd Annual International Symposium on Microarchitecture*, New York USA.
- Son, Y., & Lee, Y. (2017). Offloading method for efficient use of local computational resources in mobile location-based services using clouds. *Mobile Information Systems*, 2017.
- Su, Y.-Y., & Flinn, J. (2005). Slingshot: deploying stateful services in wireless hotspots. *The 3rd International Conference on Mobile Systems, Applications, and Services*, pp. 79-92, Seattle, WA, USA.
- Nick, T. (2015). How to tell if your smartphone's battery is healthy or bad (iPhone and Android guide). Retrieved from (http://www.phonearena.com/news/How-to-tell-if-your-smartphones-battery-is-healthy-or-bad-iPhone-and-Android-guide_id65591).
- Teka, F. A. (2014). Seamless live virtual machine migration for cloudlet users with. carleton University Ottawa. *Master Thesis in Electrical and Computer Engineering*, Carleton University Ottawa, Ontario.
- Tsirkel, A., Bradski, G., & Davies, R. (2001). Method and apparatus to adjust the brightness of a display screen: *Google Patents*.
- Vallina-Rodriguez, N., & Crowcroft, J. (2013). Energy management techniques in modern mobile handsets. *IEEE Communications Surveys & Tutorials*, 15(1), pp.179-198.
- Wang, K., Rao, J., & Xu, C.-Z. (2011). Rethink the virtual machine template. *The 7th ACM International Conference on Virtual Execution Environments*. pp. 39-50. Newport Beach, CA, USA.
- Wang, W., & Dey, T. (2011). A survey on arm cortex a processors. Retrieved from (<http://www.cs.virginia.edu/~skadron>. Accessed Date: July 2016).
- What is Windows Azure? (2010). retrieved from (<http://www.microsoft.com/bizspark/azure/>. Accessed Date: 13th June 2016).

- Wolski, R., Gurun, S., Krintz, C., & Nurmi, D. (2008). Using bandwidth data to make computation offloading decisions. The 2008. *The IEEE International Symposium on Parallel and Distributed Processing*, Rome Italy.
- Woollaston, V. (2014). Forget 3D screens and fingerprint scanners, customers really want better battery life and waterproof screens, poll reveals. Retrieved from (<http://www.dailymail.co.uk/sciencetech/article-2715860/Mobile-phone-customers-really-want-better-battery-life-waterproof-screens-poll-reveals.html>). Accessed Date: 23rd June 2016).
- Xian, C., Lu, Y.-H., & Li, Z. (2007). Adaptive computation offloading for energy conservation on battery-powered systems. *The International Conference on Parallel and Distributed Systems*, National Tsing Hua University Hsinchu, Taiwan.
- Xiao, Y., Hui, P., Savolainen, P., & Ylä-Jääski, A. (2011). CasCap: cloud-assisted context-aware power management for mobile devices. *The 2nd International Workshop on Mobile cloud Computing and Services*. pp. 13-18, Bethesda, MD, USA.
- Yang, K., Ou, S., & Chen, H.-H. (2008). An effective offloading services for resource-constrained mobile devices running heavier mobile internet applications. *IEEE Communications Magazine*, 46(1), pp. 56-63.
- Yang, L., Cao, J., Yuan, Y., Li, T., Han, A., & Chan, A. (2013). A Framework for partitioning and execution of data stream applications in fobile cloud computing. *ACM SIGMETRICS Performance Evaluation Review*, 40(4), pp. 23-32.
- Zhang, L. (2013). Power, performance modelling and optimization for mobile system and applications. *PhD Dissertation in Computer Science and Engineering*. The University of Michigan.
- Zhang, L., Tiwana, B., Qian, Z., Wang, Z., Dick, R. P., Mao, Z. M., & Yang, L. (2010). Accurate online power estimation and automatic battery behavior based power model generation for smartphones. *The 8th IEEE/ACM/IFIP International Conference on Hardware/software Code Sign and System Synthesis*. pp. 105-114, Scottsdale, AZ, USA.
- Zhang, X., Jeong, S., Kunjithapatham, A., & Gibbs, S. (2010). Towards an elastic application model for augmenting computing capabilities of mobile platforms. *The 3rd International Conference on Mobile Wireless Middleware, Operating Systems, and Applications*. pp. 161-174, Chicago IL, USA.
- Zhang, X., Kunjithapatham, A., Jeong, S., & Gibbs, S. (2011). Towards an elastic application model for augmenting the computing capabilities of mobile devices with cloud computing. *Mobile Networks and Applications*, 16(3), pp. 270-284.

Zhao, B., Xu, Z., Chi, C., Zhu, S., & Cao, G. (2010). Mirroring smartphones for good: A feasibility study. *The 7th International Conference on Mobile and Ubiquitous Systems: Computing, Networking, and Services*, pp. 26-38, Sydney Australia.

Zhuang, Z., Kim, K.-H., & Singh, J. P. (2010). Improving energy efficiency of location sensing on smartphones. *The 8th International Conference on Mobile Systems, Applications, and Services*. pp. 315-330, San Francisco CA, USA.



Appendix A

Table A.1 Energy Consumption Cost EC1 of Offloading Matrix Multiplication Service in Traditional Computational Offloading

Matrix Size	Energy Consumption Cost (J)				Standard Deviation (SD)	Confidence Interval
	CPU (J)	LCD (J)	Wi-Fi (J)	Total consumption (J)		
160x160	3.9	1.4	2.5	7.72	0.420714	7.8(+/-)0.966
170x170	4.1	1.5	3.2	9.14	0.568331	8.8(+/-)1.311
180x180	4.8	1.9	3.4	10.24	0.585662	10.1(+/-)1.351
190x190	4.9	2.2	3.5	11.18	1.712308	10.6(+/-)3.951
200x200	6.0	2.4	3.4	11.5	0.223607	11.8(+/-)0.516
210x210	6.8	3.0	6.9	16.74	0.450555	16.7(+/-)1.040
220x220	7.3	3.3	6.7	15.06	1.301153	17.3(+/-)3.003
230x230	7.8	3.8	6.8	19.48	1.645296	18.4(+/-)3.797
240x240	8.6	3.9	7.7	21.8	1.189538	20.2(+/-)2.745
250x250	9.8	3.9	8.1	21.52	0.216795	21.8(+/-)0.500
260x260	11.0	4.1	8.4	23.56	0.371484	23.5(+/-)0.857
270x270	11.7	4.5	8.9	25.92	1.077961	25.1(+/-)2.488
280x280	13.2	4.4	8.4	26.34	0.31305	26(+/-)0.722
290x290	15.2	4.8	8.7	29	0.738241	28.7(+/-)1.704
300x300	16.2	5.2	8.9	30.36	0.240832	30.3(+/-)0.556
310x310	18.5	5.4	9.1	33.52	0.356371	33(+/-)0.822
320x320	19.7	5.4	9.5	35.48	0.766159	34.6(+/-)1.768
330x330	21.4	5.8	9.8	37.6	0.424264	37(+/-)0.979
340x340	23.2	6.1	10.4	40.82	1.482228	39.7(+/-)3.420
350x350	26.3	6.2	10.8	43.28	0.549545	43.3(+/-)1.268
360x360	26.9	6.4	10.3	43.62	0.363318	43.6(+/-)0.838
370x370	29.5	6.7	10.7	46.56	0.31305	46.9(+/-)0.722
380x380	33.2	7.0	10.1	50.6	0.524404	50.3(+/-)1.210
390x390	37.8	7.2	12.4	58.08	0.637966	57.4(+/-)1.472
400x400	41.2	7.1	12.5	61.08	0.756307	60.8(+/-)1.745
410x410	45.7	7.5	13.8	66.88	1.037786	67(+/-)2.395
420x420	51.7	7.4	13.2	72.56	0.270185	67(+/-)0.623
430x430	55.4	7.6	14.5	77.96	0.630872	77.5(+/-)1.456
440x440	64.8	7.8	15.1	88.66	0.95289	87.7(+/-)2.199
450x450	78.4	7.9	15.2	102.08	8.517159	101.5(+/-)1.388

Appendix B

Table B.1 Time Consumption Cost TC1 of Offloading Matrix Multiplication Service in Traditional Computational Offloading

Matrix Size	Execution Time Milliseconds (ms)	Standard Deviation (SD)	Confidence Interval
160x160	9608	535.7891	9608 (+/-)1236.39
170x170	10544.8	313.4002	10544.8 (+/-)723.21
180x180	11152.8	171.0956	11152.8 (+/-)394.82
190x190	13448.2	382.5588	13448.2 (+/-)882.80
200x200	14286.2	402.2328	14286.2 (+/-)928.20
210x210	16406.8	326.287	16406.8 (+/-)752.94
220x220	17200.2	140.4482	17200.2 (+/-)324.10
230x230	18931.8	183.9489	18931.8 (+/-)424.48
240x240	20687.8	442.4536	20687.8 (+/-)1021.15
250x250	23727	506.9122	23727 (+/-)1169.76
260x260	26968.2	370.9699	26968.2 (+/-)856.05
270x270	29593.8	528.0177	29593.8 (+/-)1218.46
280x280	33056.6	631.9401	33056.6 (+/-)1458.27
290x290	35777	606.8299	35777 (+/-)1400.33
300x300	40627.4	1042.291	40627.4 (+/-)2405.21
310x310	46807.4	1081.437	46807.4 (+/-)2495.54
320x320	52168.8	909.6099	52168.8 (+/-)2099
330x330	58184.4	849.8919	58184.4 (+/-)1961.23
340x340	63449.8	701.4454	63449.8 (+/-)1618.67
350x350	70335.2	1286.772	70335.2 (+/-)2969.81
360x360	76010.6	1611.518	76010.6 (+/-)3718.77
370x370	87753.2	1720.307	87753.2 (+/-)3969.81
380x380	101338.6	4000.621	101338.6 (+/-)9231.92
390x390	120554.6	3948.49	120554.6 (+/-)9111.62
400x400	136166	2742.36	136166 (+/-)6328.33
410x410	148291.4	1998.916	148291.4 (+/-)4612.74
420x420	165687.2	2785.926	165687.2 (+/-)6428.86
430x430	173176.8	1495.201	173176.8 (+/-)3450.35
440x440	182196.4	1792.104	182196.4 (+/-)4135.49
450x450	189523.8	2089.393	189523.8 (+/-)4821.52

AUTHOR'S BIODATA

Mushtaq Ali was born in Shangla (Swat) Pakistan. He received his Bachelor degree in Computer Sciences from University of Peshawar, Pakistan, in 2003; Master from Hazara University Mansehra Pakistan, in 2006. Currently he is pursuing his PhD Candidature under Scholarship from the Ministry of Higher Education, Malaysia and support from University Malaysia Pahang (UMP). He started his career as an IT-Instructor in 2007 at Pak Swiss Technical Center Mingora Swat Pakistan. He held last position as a Network Administrator in 2012 at AL-Khayrin Group Trading & Construction W.L.L, Doha Qatar. He has published 9 Journal articles in different Journal including ISI impact factor Journal and presented 4 papers in national & International conferences. His Research interests includes Cloud Computing, Distributed Systems, Mobile Security, Smartphone Efficiencies and Water Marking.

The logo of University Malaysia Pahang (UMP) is a large, stylized downward-pointing arrow. The arrow is composed of several overlapping, semi-transparent shapes in shades of teal, light blue, and yellow. The letters 'UMP' are printed in a bold, white, sans-serif font across the center of the arrow's shaft.

UMP

THE LIST OF PUBLICATIONS

JOURNALS

- Mushtaq Ali, Jasni Muhamed Zain, Muhamad Fadli Zolkipli, & Gran Badshah, (2015). Mobile Cloud Computing & Mobile's Battery Efficiency Approaches: a Review. *Journal of Theoretical and Applied Information Technology*, 79(1), pp. 153.
- Mushtaq Ali, Jasni Muhamed Zain, Muhamad Fadli Zolkipli, & Gran Badshah (2015). Taxonomy of Computational Offloading in Mobile Devices. *World Applied Sciences Journal*, 33(12), pp. 1798-1805.
- Mushtaq Ali, Jasni Muhamed Zain, Muhamad Fadli Zolkipli, & Gran Badshah (2018). Analysis of Power Consumption in Mobile Devices. *Journal of Computer Communications*, Elsevier, 2016 (ISI Impact Factor 2.38). (Under Review)
- Mushtaq Ali, Mazlina Abdul Majid, Jasni Mohamad Zain, Mohamad Fadli Zolkipli, Shahid Anwar Safi (2018). "Mobile Cloud Computing with SOAP and REST Web Services." *Journal of Physics: Conference Series*. (Accepted).
- Gran Badshah, Siau-Chuin Liew, Jasni Mohamad Zain, Mushtaq Ali (2015). "Watermark Compression in Medical Image Watermarking Using Lempel-Ziv-Welch (LZW) Lossless Compression Technique." *Journal of Digital Imaging*, 29(2), pp. 216.
- Gran Badshah, Siau-Chuin Liew, Jasni Mohamad Zain, Mushtaq Ali (2015). "Secured Telemedicine Using Whole Image as Watermark with Tamper Localization and Recovery Capabilities." *Journal of Information Processing Systems*, 11(4), pp. 601-615.
- Gran Badshah, Siau-Chuin Liew, Jasni Mohamad Zain, Mushtaq Ali (2016). "Watermarking of ultrasound medical images in tele radiology using compressed watermark." *Journal of Medical Imaging*, 3(1), pp 25.
- Juluis Odili, Mohamad Nizam Mohamad Kahar, Shahid Anwar, Mushtaq Ali (2017). "Tutorials on African Buffalo Optimization for Solving the Travelling Salesman Problem." *International Journal of Software Engineering and Computer Sciences*" v (3).
- Shahid Anwar, Mohamad Fadli Zolkipli, Jasni Mohamd Zain, Zakira Inayat, Julius Odili, Mushtaq Ali (2018). Android Botnets: A Serious Threat to Android Devices." *Pertanika Journal of Science and Technology*" v 26(1).

CONFERENCES

Mushtaq Ali, Jasni Muhamed Zain, Muhamad Fadli Zolkilpli and Gran Badshah (2014). Mobile Cloud Computing and Mobile's Battery Augmentation Techniques. *IEEE Student Conference on Research and Development (SCOReD) 2014*. Pages 1-6, Penang Malaysia.

Mushtaq Ali, Jasni Muhamed Zain, Muhamad Fadli Zolkilpli and Gran Badshah (2015). Battery Efficiency of Mobile Devices through Computational Offloading. *IEEE Student Conference on Research and Development (SCOReD) 2015*, pp 317 – 322, Kuala Lumpur Malaysia.

Muhammad Wasfi Nabeel, Abdullah Embong, Mushtaq Ali (2015). "Computational Offloading in Smart Internet Devices." *4th International Conference on Software Engineering and Computer Systems*, Kuantan Malaysia.



UMP