

**A LIGHTWEIGHT PROCESS MIGRATION BASED
COMPUTATIONAL OFFLOADING FRAMEWORK
FOR MOBILE DEVICE AUGMENTATION**

ABDULLAH

**FACULTY OF COMPUTER SCIENCE AND
INFORMATION TECHNOLOGY
UNIVERSITY OF MALAYA
KUALA LUMPUR**

2017

A LIGHTWEIGHT PROCESS MIGRATION BASED
COMPUTATIONAL OFFLOADING FRAMEWORK
FOR MOBILE DEVICE AUGMENTATION

ABDULLAH

THESIS SUBMITTED IN FULFILMENT
OF THE REQUIREMENTS
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

FACULTY OF COMPUTER SCIENCE AND
INFORMATION TECHNOLOGY
UNIVERSITY OF MALAYA
KUALA LUMPUR

2017

UNIVERSITI MALAYA

ORIGINAL LITERARY WORK DECLARATION

Name of Candidate: Abdullah

(I.C./Passport No.

Registration/Matrix No.: WHA130018

Name of Degree: Doctorate of Philosophy

Title of Project Paper/Research Report/Dissertation/Thesis ("A Lightweight Process Migration based Computational Offloading Framework for Mobile Device Augmentation"):

Field of Study: Mobile Cloud Computing

I do solemnly and sincerely declare that:

- (1) I am the sole author/writer of this Work;
- (2) This work is original;
- (3) Any use of any work in which copyright exists was done by way of fair dealing and for permitted purposes and any excerpt or extract from, or reference to or reproduction of any copyright work has been disclosed expressly and sufficiently and the title of the Work and its authorship have been acknowledged in this Work;
- (4) I do not have any actual knowledge nor do I ought reasonably to know that the making of this work constitutes an infringement of any copyright work;
- (5) I hereby assign all and every rights in the copyright to this Work to the University of Malaya ("UM"), who henceforth shall be owner of the copyright in this Work and that any reproduction or use in any form or by any means whatsoever is prohibited without the written consent of UM having been first had and obtained;
- (6) I am fully aware that if in the course of making this Work I have infringed any copyright whether intentionally or otherwise, I may be subject to legal action or any other action as may be determined by UM.

Candidate's Signature

Date

Subscribed and solemnly declared before,

Witness's Signature

Date

Name:

Designation:

ABSTRACT

In recent years, the paradigm of mobile cloud computing has been introduced to extend capabilities of mobile devices, by taking advantage of high-speed wireless communications and high-performance cloud platforms to help gather, store and process data for the mobile devices. In this paradigm, the cloud-based mobile applications usually employ computational offloading for the augmentation of mobile device capabilities. Mobile device OS vendors are focused toward native mobile applications lifecycle to improve battery consumption and application execution performance. For example, Google has introduced Android Runtime Environment (ART) featuring Ahead of Time (AHOT) compilation to native instructions in place of Dalvik Virtual Machine (DVM) which consumes extra time and energy because of the Just in Time (JIT) compilation. However, current state-of-the-art offloading solutions do not consider AHOT compilations to native binaries in the ART environment. To address the issue in offloading ART-based mobile applications, we propose a lightweight computational offloading framework. The lightweightness is measured as the overhead energy consumption and application execution time added up by the proposed framework. Further, we explain in details the design and implementation of the proposed prototype framework. The proposed framework requires infrastructural support from the remote computing platforms such as data centers or cloudlets to provide Offloading as a Service (OaaS) for a heterogeneous mobile cloud ecosystem. The proposed framework is evaluated using experimental testbed and validated using statistical modeling. Numerical results from the testbed revealed that the proposed framework saves almost 44% of the execution time and 84% of the energy consumption of the experimental application used.

ABSTRAK

Dalam tahun-tahun kebelakangan ini, paradigma pengkomputeran awan mudah alih telah diperkenalkan untuk memperluaskan keupayaan peranti mudah alih, dengan mengambil kesempatan daripada komunikasi tanpa wayar berkelajuan tinggi dan platform awan berprestasi tinggi untuk membantu mengumpul, menyimpan dan memproses data untuk peranti mudah alih. Dalam paradigma ini, aplikasi mudah alih berasaskan awan biasanya mengambil kerja pemunggahan pengiraan untuk pembesaran keupayaan peranti mudah alih. peranti mudah alih vendor OS memberi tumpuan ke arah aplikasi mudah alih kitaran hayat berasal dari meningkatkan penggunaan bateri dan prestasi pelaksanaan permohonan. Sebagai contoh, Google telah memperkenalkan gls ART menampilkan gls AHOT kompilasi arahan asli di tempat gls DVM yang menggunakan masa dan tenaga tambahan kerana gls JIT kompilasi. Walau bagaimanapun, penyelesaian pemunggahan state-of-the-art semasa tidak menganggap gls AHOT kompilasi untuk binari asli di persekitaran ART. Untuk menangani isu ini dalam pemunggahan aplikasi mudah alih berdasarkan ART-, kami mencadangkan rangka kerja pemunggahan pengiraan yang ringan. lightweightedness diukur sebagai penggunaan tenaga dan pelaksanaan permohonan kali overhead ditambah oleh rangka kerja yang dicadangkan. Selanjutnya, kami menerangkan dengan terperinci tentang reka bentuk dan pelaksanaan rangka kerja prototaip yang dicadangkan. rangka kerja yang dicadangkan memerlukan sokongan yang padu dari platform pengkomputeran jauh seperti pusat data atau cloudlets untuk menyediakan gls OaaS untuk ekosistem awan mudah alih yang heterogen. rangka kerja yang dicadangkan dinilai dengan menggunakan tapak ujian eksperimen dan disahkan menggunakan pemodelan statistik. Keputusan berangka dari tapak ujian menunjukkan bahawa rangka kerja yang dicadangkan menjimatkan hampir 44 % daripada masa pelaksanaan dan 84 % daripada penggunaan tenaga permohonan eksperimen digunakan.

ACKNOWLEDGEMENTS

I was fortunate enough to meet wonderful peoples and receive a great deal of support from them while pursuing my Ph.D. at University of Malaya. Without them, it would not be possible to finish my dissertation.

First of all, I would like to thank my supervisors, Professor Abdullah Gani and Assoc. Prof. Dr. Rafidah Md Noor. Both not only taught me and gave me insightful advice on my research, but also showed me what a mentor should look like. They have been always supportive, cheerful, and constructive.

I am also grateful to my colleagues and members here at Center for Mobile Cloud Computing, Dr. Anjum Naveed, Ibrar Yaqoob, Junaid Shuja and Raja Wasim, for their valuable comments on my proposal and thesis.

I am deeply grateful to my family. My father and mother taught me important things in my life and always supported me. Last and importantly, I would like to thank my wife for her patience. I would not have been able to go through this without her support. Together, we have made a great journey so far. I am sure that it will be even greater from now on.

TABLE OF CONTENTS

Abstract	iii
Abstrak	iv
Acknowledgements	v
Table of Contents	vi
List of Figures	x
List of Tables	xiii
List of Symbols and Abbreviations	xv
CHAPTER 1: INTRODUCTION	1
1.1 Background	1
1.1.1 Mobile cloud computing	1
1.1.1.1 The conventional MCE	2
1.1.1.2 The Ad-Hoc MCE	2
1.1.1.3 The Hybrid MCE	3
1.1.2 Actors in MCE	3
1.1.2.1 Mobile user	4
1.1.2.2 Cloud provider	4
1.1.2.3 Cloudlets	5
1.1.2.4 Mobile nodes	5
1.1.3 Computational offloading	5
1.1.4 Cloud computing and computational offloading as a service	6
1.1.5 Process migration	7
1.2 Research motivation	7
1.3 Statement of the problem	9
1.4 Statement of objectives	10
1.5 Proposed research methodology	11
1.6 Layout of thesis	12
CHAPTER 2: LITERATURE REVIEW	15
2.1 Taxonomy of computational offloading issues in the client subsystem	15
2.1.1 Mobile device issues	16
2.1.2 Networking issues	17
2.1.2.1 Low bandwidth	17
2.1.2.2 Network reliability	17
2.1.2.3 Heterogeneity	17
2.1.2.4 High access latency	18
2.1.3 Application issues	18
2.1.3.1 Protocol diversity	19
2.1.3.2 Multiple execution formats	19
2.1.3.3 Local resource access	19

2.1.4	Monitoring	19
2.1.5	Ad-hoc mobile cloud management	20
	2.1.5.1 Formation and incentive management	20
	2.1.5.2 Heterogeneity and interoperability	20
2.2	Taxonomy of computational offloading Functions in MCE	21
2.2.1	Migration mechanism	21
	2.2.1.1 VM/Phone clone migration	22
	2.2.1.2 Code migration and delegation	22
	2.2.1.3 Thread state migration	23
2.2.2	Migration decision	24
2.2.3	Application partitioning	24
2.2.4	Resource discovery	25
2.2.5	Link selection	25
2.2.6	Deployment model	25
2.3	State-of-the-art computational offloading mechanisms in MCE	26
2.3.1	AlfredO	26
2.3.2	Misco	27
2.3.3	MAUI	27
2.3.4	CloneCloud	28
2.3.5	ThinkAir	29
2.3.6	Cuckoo	29
2.3.7	COSMOS	30
2.3.8	Hyrax	30
2.3.9	VM-based cloudlet	31
2.3.10	COMET	31
2.3.11	Virtual smartphone	32
2.3.12	AIOLOS	33
2.4	Discussion on computational offloading frameworks	33
2.5	Research challenges	35
2.5.1	Energy efficiency	36
2.5.2	Process migration based computational offloading	36
2.5.3	Mobility-assisted server-to-server computational offloading	37
2.5.4	Secure computations on server side	38
2.5.5	Incentive management and resource discovery	38
2.5.6	Automatic application partitioning	39
2.5.7	Scheduling partitions	41
2.5.8	Better ARM emulation in cloud	41
2.6	Conclusion	42
CHAPTER 3: PROBLEM ANALYSIS		44
3.1	Computation offloading benefit analysis	44
3.2	Analyzing the existing computational offloading mechanisms	51
	3.2.1 VM/Phone clone migration	51
	3.2.2 Code migration and delegation	54
	3.2.3 Thread state migration	58
3.3	Why process migration based computational offloading?	60
3.4	Feasibility analysis of process migration based computational offloading	62
	3.4.1 Experiment details	65
3.5	Conclusion	66

CHAPTER 4: LIGHTWEIGHT COMPUTATIONAL OFFLOADING FRAMEWORK FOR MOBILE DEVICE AUGMENTATION	68
4.1 A Lightweight process migration based computational offloading framework for mobile device augmentation	68
4.1.1 System requirements	70
4.1.2 Process execution	73
4.2 Proposed Components in PMCO framework	74
4.2.1 Migration Preference Manager (MPM)	74
4.2.2 Application Migration Manager (AMM)	75
4.2.3 Application Migration Coordinator (AMC)	76
4.2.4 Server side admission control	78
4.2.5 Server side application migration manager	78
4.2.6 Proposed PMCO Algorithms	79
4.2.7 PMCO execution flow using sequence diagram	85
4.3 Data design	88
4.3.1 Evaluation metrics	88
4.4 Conclusion	90
CHAPTER 5: EVALUATION	91
5.1 Experimental setup	91
5.2 Evaluation methods	96
5.2.1 Descriptive statistics	96
5.2.2 Confidence interval	96
5.2.3 Paired samples t-test	97
5.2.4 Linear regression	97
5.3 Parametric evaluations	98
5.3.1 Data transmission	99
5.3.2 Execution time	106
5.3.2.1 Statistical modeling of execution time in local execution mode	107
5.3.2.2 Statistical modeling of execution time in PMCO execution mode	110
5.3.3 Energy consumed	114
5.3.3.1 Statistical modeling of energy consumed in local execution mode	115
5.3.3.2 Statistical modeling of energy consumed in PMCO execution mode	118
5.3.4 Compute power	121
5.4 Conclusion	123
CHAPTER 6: RESULTS AND DISCUSSION	124
6.1 Performance evaluation results	124
6.1.1 Execution time	124
6.1.1.1 Validation	133
6.1.2 Consumed energy	138
6.1.2.1 Validation	147
6.1.3 Compute power	151
6.2 Comparative evaluations	158
6.3 Conclusion	160

CHAPTER 7: CONCLUSION	162
7.1 Retrospection of the research objectives	162
7.2 Research contributions	164
7.2.1 Taxonomy of computational offloading issues in the client sub-system	165
7.2.2 Taxonomy of computational offloading functions	165
7.2.3 Empirical analysis of existing computation offloading solutions	165
7.2.4 Lightweight process migration based computational offloading framework	166
7.2.5 Evaluation and validation of the proposed solution	166
7.3 Significance and limitations of the proposed solution	167
7.4 Future prospects	169
7.5 Scholarly publications	170
References	172

University of Malaya

LIST OF FIGURES

Figure 1.1: A view of conventional Mobile Cloud Environment (MCE).	2
Figure 1.2: A view of Ad-hoc MCE.	3
Figure 1.3: A view of Hybrid MCE.	4
Figure 1.4: Computational offloading sub-systems in MCE.	6
Figure 2.1: Taxonomy of computational offloading issues on the client subsystem in MCE.	16
Figure 2.2: Taxonomy of computational offloading functions in MCE.	21
Figure 2.3: VM based augmentation of mobile devices.	22
Figure 2.4: Remote execution using code migration and delegation.	23
Figure 2.5: Generalized representation of thread synchronization used in MCE.	24
Figure 2.6: (a) The original ART schematics (b) Modified ART schematics.	40
Figure 2.7: Schedule of automatically generated application partitions on mobile cloud resources.	42
Figure 2.8: (a) Schematics of QEMU translation process from guest to host code blocks(b) Schematics of translation process from guest to host code blocks of envisioned emulator.	43
Figure 3.1: Energy consumption on mobile device in scenario 1.	47
Figure 3.2: Energy consumption on mobile device in scenario 2.	48
Figure 3.3: Energy benefit indication while varying upload and download transmission data sizes.	48
Figure 3.4: Energy benefit indication while varying upload and download transmission data sizes.	49
Figure 3.5: Energy benefit indication while varying upload and download transmission data rates.	50
Figure 3.6: Energy benefit indication while varying upload and download transmission data rates.	51
Figure 3.7: Phone image sizes of Samsung Galaxy SII-i9100g.	52
Figure 3.8: Phone image backup and restore time.	52
Figure 3.9: Phone image sizes of Xiaomi Mi4i.	53
Figure 3.10: The energy consumption of WiFi connectivity vs. 3G connectivity.	54
Figure 3.11: Code migration framework for Android devices.	55
Figure 3.12: Network topology of code migration experimental setup.	56
Figure 3.13: Impact on the execution time of experimental application using code migration.	57
Figure 3.14: Impact on the energy consumption of experimental application using code migration.	57
Figure 3.15: COMET discrepancies with same applications from different developers.	59
Figure 3.16: Performance comparison of modified DVM vs. stock DVM using Linpack benchmarks (in Mega FLOPS).	60
Figure 3.17: File type of Dalvik compiled DEX file.	61
Figure 3.18: File type of ART compiled DEX file.	62
Figure 3.19: Comparison of emulator performance with real device.	63
Figure 3.20: Experimental setup.	64
Figure 3.21: Execution time impact (in milliseconds).	66

Figure 4.1: Bird eye view of proposed Process Migration based Computational Offloading (PMCO) for mobile device augmentation.	71
Figure 4.2: Interactive flow of the proposed PMCO based mobile device augmentation framework.	72
Figure 4.3: Application execution in our proposed framework.	73
Figure 4.4: Checkpoint message from coordinator to the user process.	74
Figure 4.5: Flow diagram of process migration from mobile device to remote computing device.	84
Figure 4.6: Restarting a process from the checkpoint file.	84
Figure 4.7: Flow diagram of process migration from remote computing device to mobile device.	84
Figure 4.8: Sequence diagram of offloading a migration-aware application using the proposed PMCO based mobile device augmentation framework.	85
Figure 4.9: Sequence diagram of offloading a non migration-aware application using the proposed PMCO based mobile device augmentation framework.	86
Figure 5.1: Network topology of experimental setup.	93
Figure 5.2: Scatter plot of matrix multiplication noisy data transfer time, with the linearity correlation determined by the line of best fit.	102
Figure 5.3: Scatter plot of matrix multiplication noisy data reception time, with the linearity correlation determined by the line of best fit.	102
Figure 5.4: Scatter plot of matrix multiplication normalized data transfer time, with the linearity correlation determined by the line of best fit.	103
Figure 5.5: Scatter plot of matrix multiplication normalized data reception time, with the linearity correlation determined by the line of best fit.	104
Figure 5.6: Scatter plot of Local matrix multiplication execution time, with the linearity correlation determined by the line of best fit.	109
Figure 5.7: Scatter plot of server side matrix multiplication execution time, with the linearity correlation determined by the line of best fit.	112
Figure 5.8: Scatter plot showing linearity correlation between local execution time and consumed energy.	117
Figure 5.9: Scatter plot showing linearity correlation between PMCO execution response time and consumed energy.	120
Figure 6.1: Execution time for matrix multiplication workloads generated via experimentation.	130
Figure 6.2: Scattered plot with interpolation lines for matrix multiplication execution time.	133
Figure 6.3: Breakdown of the contributing factors of remote execution time using PMCO.	134
Figure 6.4: Impact of the contributing factors on remote execution time using PMCO.	134
Figure 6.5: Execution time validation by experimentation vs. statistical model.	136
Figure 6.6: Scattered plot with interpolation lines for matrix multiplication execution time in local mode using experiments vs. statistical model.	137
Figure 6.7: Scattered plot with interpolation lines for Matrix multiplication execution time in PMCO mode using experiments vs. statistical model.	137
Figure 6.8: Energy consumed for matrix multiplication workloads gathered via PowerAPI.	143

Figure 6.9: Scattered plot with interpolation lines for matrix multiplication energy consumption.	145
Figure 6.10: Scatter plot showing linearity correlation between local execution time and consumed energy.	145
Figure 6.11: Scatter plot showing linearity correlation between PMCO execution response time and consumed energy.	146
Figure 6.12: Energy consumed for benchmark application workloads gathered via PowerAPI.	147
Figure 6.13: Consumed energy validation by experimentation vs. statistical model.	149
Figure 6.14: Scattered plot with interpolation lines for matrix multiplication energy consumption in local mode using experiments vs. statistical model.	150
Figure 6.15: Scattered plot with interpolation lines for matrix multiplication energy consumption in PMCO mode using experiments vs. statistical model.	150
Figure 6.16: Mean MFLOPS values for 30 observations of Linpack and Scimark benchmarks generated via Local, Local_PMCO and PMCO executions.	156
Figure 6.17: Mean MIPS and MWIPS values for 30 observations of Dhrystone and Whetstone benchmarks generated via Local, Local_PMCO and PMCO executions.	157
Figure 6.18: Comparison between Comet and PMCO when applications cannot be offloaded.	159

LIST OF TABLES

Table 2.1: Comparison of Computational Offloading Mechanism used in MCE	33
Table 2.2: Pseudo-codic Illustration of Unmodified Original Code with no synchronization points and the modified code with synchronization points.	40
Table 3.1: Simulation Settings	46
Table 3.2: Data Observation while varying Upload and Download Transmission Sizes	49
Table 3.3: Data Observation while varying Upload and Download Transmission Rates	50
Table 4.1: Metrics for Performance Assesment of the Proposed Framework	88
Table 5.1: Benchmark Matrix Multiplication Granularity	92
Table 5.2: Average amount of data transfer and received, along with the average time it takes, based on 95% confidence interval	100
Table 5.3: Regression statistics summary for data transfer time of matrix multiplication in PMCO mode	104
Table 5.4: Regression statistics summary for data reception time of matrix multiplication in PMCO mode	105
Table 5.5: Validation of data transfer time statistical model using split-sample approach	106
Table 5.6: Validation of data reception time statistical model using split-sample approach	106
Table 5.7: Mean execution time of matrix multiplication workloads with 95% confidence interval, Using three execution modes	107
Table 5.8: Regression Statistics Summary for Local execution time of matrix multiplication	109
Table 5.9: Validation of Local Execution Time Statistical Model using Sample Split Approach	110
Table 5.10: Regression Statistics Summary for server side execution time of matrix multiplication	113
Table 5.11: Validation of server side execution time statistical model using sample split approach	114
Table 5.12: Workload Energy Consumption in Joules with 95% confidence interval, Using three execution modes	115
Table 5.13: Regression Statistics Summary for energy consumption in local execution mode	117
Table 5.14: Validation of local device energy consumption model using sample split approach	118
Table 5.15: Regression Statistics Summary for energy consumption in PMCO execution mode	120
Table 5.16: Validation of PMCO energy consumption model using sample split approach	121
Table 5.17: Benchmark workloads descriptive statistics depicting the improvement in the compute power	122
Table 6.1: Execution Time with 95% Confidence Interval in Local Execution Mode Generated via Experiments	126

Table 6.2: Execution Time with 95% Confidence Interval in Local_PMCO Execution Mode Generated via Experiments	127
Table 6.3: Execution Time with 95% Confidence Interval in PMCO Execution Mode Generated via Experiments	128
Table 6.4: Descriptive Statistics of Execution Time Data Generated by standard experimentation	129
Table 6.5: t-Test: Paired Two Sample for Mean Execution Time of Local and PMCO	131
Table 6.6: The execution time data generated via statistical modeling for local and PMCO execution modes	135
Table 6.7: Consumed energy observations with 95% confidence interval in local execution mode gathered via PowerAPI during experimentation	139
Table 6.8: Consumed Energy observations with 95% Confidence Interval in Local_PMCO Execution Mode gathered via PowerAPI during experimentation	140
Table 6.9: Consumed Energy observations with 95% Confidence Interval in PMCO Execution Mode gathered via PowerAPI during experimentation	141
Table 6.10: Descriptive Statistics of Consumed Energy Data gathered via PowerAPI	142
Table 6.11: t-Test: Paired Two Sample Mean of Consumed Energy of workloads of Local and PMCO Mode of execution	144
Table 6.12: The energy consumption data generated via statistical modeling for local and PMCO execution modes	148
Table 6.13: Compute Power Observations in Local Execution Environment Generated via Standardized Benchmarking Mechanisms	152
Table 6.14: Compute Power Observations in Local_PMCO Execution Environment Generated via Standardized Benchmarking Mechanisms	153
Table 6.15: Compute Power Observations in PMCO Execution Environment Generated via Standardized Benchmarking Mechanisms	154
Table 6.16: Descriptive Statistics of Compute Power Data Generated by standard benchmarking applications	155
Table 6.17: t-Test: Paired Two Sample for Mean Compute power in local execution mode and PMCO mode	158
Table 6.18: Comparison when benchmark are executed locally using COMET and PMCO	159
Table 6.19: Comparison between conventional code offloading and PMCO	160

LIST OF SYMBOLS AND ABBREVIATIONS

AHOT	: Ahead of Time.
ALVM	: Application Level Virtual Machine.
AOSP	: Android Open Source Project.
ART	: Android Runtime Environment.
CLR	: Common Language Runtime.
DSM	: Distributed Shared Memory.
DVM	: Dalvik Virtual Machine.
ELF	: Executable and Linking Format.
GCC	: GNU Compiler Collection.
GPRS	: General Packet Radio Service.
IDE	: Integrated Development Environment.
JIT	: Just in Time.
JVM	: Java Virtual Machine.
LTE	: Long Term Evolution.
MCC	: Mobile Cloud Computing.
MCE	: Mobile Cloud Environment.
OaaS	: Offloading as a Service.
PMCO	: Process Migration based Computational Offloading.
QoE	: Quality of Experience.
QoS	: Quality of Service.
SOA	: Service Oriented Architecture.
TCP	: Transmission Control Protocol.
TDP	: Thermal Design Power.
UDP	: User Datagram Protocol.
VM	: Virtual Machine.
VMM	: Virtual Machine Monitors.
WAN	: Wide Area Network.
WiFi	: Wireless Fidelity.

CHAPTER 1: INTRODUCTION

This chapter presents an overview of the research carried out in this thesis. We present motivation in undertaking the research in this thesis and state the research problem that is investigated and addressed in this research. Our research aim and objectives also are presented in this chapter. Furthermore, the research methodology that is proposed to address the research problem is described.

The remainder of this chapter is as follows. Section 1.1 presents the domain background of Mobile Cloud Computing (MCC), computational offloading, and process migration. Section 1.2 presents the motivation of research followed by Section 1.3 that presents the identified and established research problem. We state the research aim and objectives in Section 1.4 and describe our proposed methodology to address the research problem in Section 1.5. Finally, Section 1.6 presents the layout of this thesis.

1.1 Background

This section describes the preliminary concepts and actors in MCE, its association with cloud computing systems along with the essentials of computational offloading in MCEs.

1.1.1 Mobile cloud computing

The concept of MCC was introduced just after the introduction of 'cloud computing' (Dinh, Lee, Niyato, & Wang, 2013). Since then, it has been attracting researchers from academia and industry to improve the mobile application execution parameters such as energy consumption, response time, and monetary cost (Abolfazli, Sanaei, Ahmed, Gani, & Buyya, 2014; Ahmed, Gani, Khan, Buyya, & Khan, 2015; Shiraz, Gani, Khokhar, & Buyya, 2013).

From a definition point of view, MCC is defined by (Dinh et al., 2013) as *an infrastructure where both the data storage and data processing happen outside of the mobile*

device on a remote computing infrastructure. Currently, there are three types of deployment models in MCC ecosystem, i.e. (i) The conventional MCE, (ii) The Ad-Hoc MCE, and (iii) The Hybrid MCE. Each of these models is briefly explained in the following subsections.

1.1.1.1 The conventional MCE

In the conventional MCE as illustrated in Figure 1.1, the computational and storage capabilities of the smart mobile devices are augmented through stationary computing resources. These resources are either provisioned through public or private datacenters or local computing infrastructure (Satyanarayanan, Bahl, Caceres, & Davies, 2009).

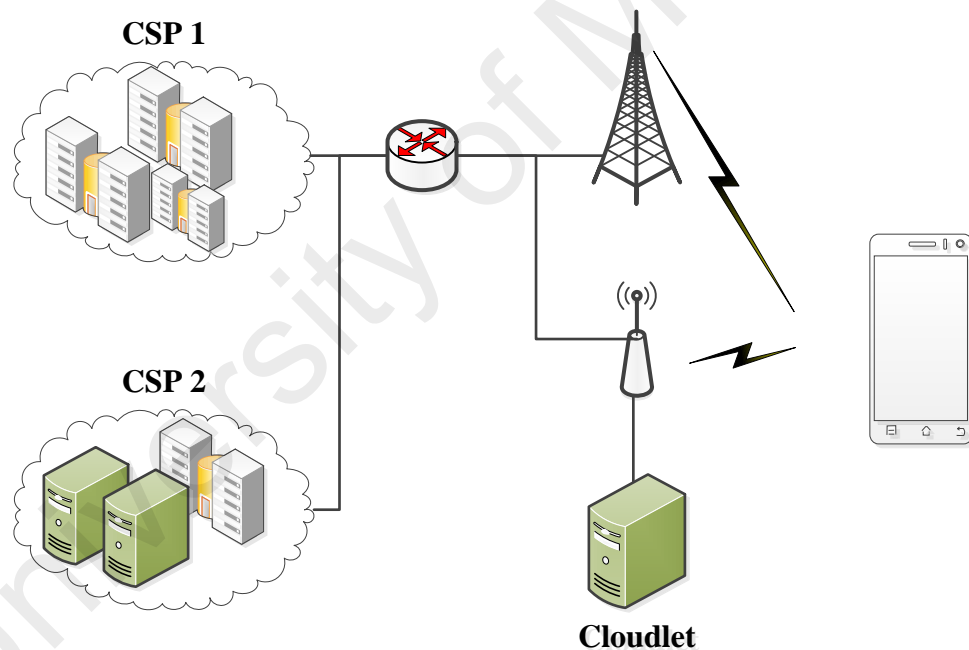


Figure 1.1: A view of conventional MCE.

1.1.1.2 The Ad-Hoc MCE

The computational capabilities of mobile devices are increasing day by day, and these devices can form a self-organizing mobile ad-hoc network of nearby devices and offer their resources as on-demand services to available nodes in the network such a configu-

ration is presented in Figure 1.2 . The ad-hoc MCE works on the principle of cooperation and collaboration enhance the compute capabilities of a group of connected smart mobile devices.

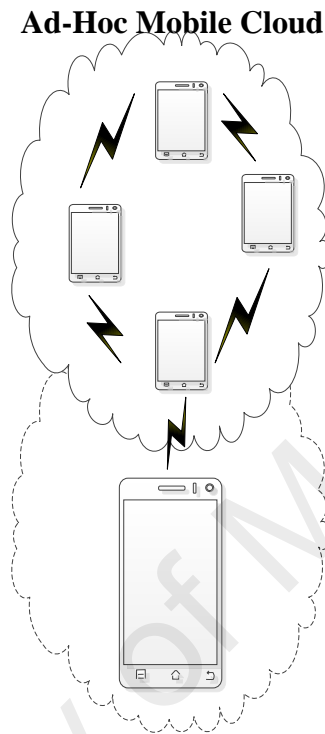


Figure 1.2: A view of Ad-hoc MCE.

1.1.1.3 The Hybrid MCE

Hybrid MCE is an MCC system where the mobile device which has been augmented or considered for augmentation is connected to both stationary and mobile computing devices as explained in 1.1.1.1 and 1.1.1.2. In illustration of such an environment is presented in Figure 1.3.

1.1.2 Actors in MCE

The ecosystem of the computational offloading process contains different types of resources and may use any of them depending on availability and the scheduling decisions. These resources are presented in a general MCC diagram shown in Figure 1.3. These

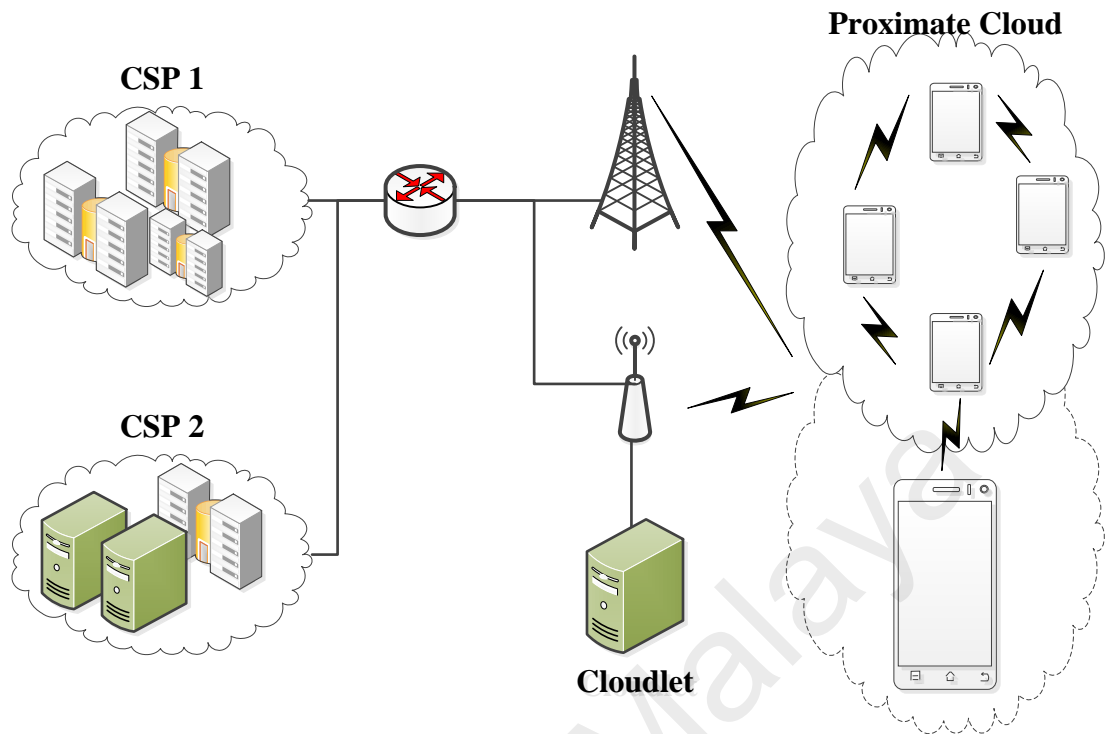


Figure 1.3: A view of Hybrid MCE.

resources lead to different actors in the ecosystems which are briefly described below.

1.1.2.1 Mobile user

The mobile device which has the application that needs remote computation and is the subject of the computational offloading to any of the available remote resources. She plays a direct role in offloading decision making and can influence the resource management decisions of the remote computing infrastructure on which the application is offloaded.

1.1.2.2 Cloud provider

This actor provides infrastructure, platform, and software services to the Mobile Users for migration of application/execution control from the end user mobile device to the cloud computing system.

1.1.2.3 Cloudlets

This Cloudlet (Satyanarayanan et al., 2009) actor is a trusted, resource-rich, and transiently customized proximate computing infrastructure available to the nearby Mobile User over one-hop network latency available for use by the nearby Mobile User.

1.1.2.4 Mobile nodes

This actor represents the resources in the local proximate mobile cloud which are based on the formation of an ad-hoc network of mobile devices within vicinity to collectively serve each other either by Wireless Fidelity (WiFi) or Bluetooth network interfaces.

1.1.3 Computational offloading

Smartphones gained enormous popularity in recent years, more and more new mobile applications such as face recognition, natural language processing, interactive gaming, and augmented reality are emerging and attract significant attention (Cohen, 2008; Soyata, Muraleedharan, Funai, Kwon, & Heinzelman, 2012). These kind of mobile applications are typically resource-hungry, demanding intensive computation and high energy consumption. Due to the physical size constraint, mobile devices, in general, have limited computational resources and limited battery life. The tension between resource-hungry applications and resource-constrained mobile devices hence poses a significant challenge for the future mobile platform development (Cuervo et al., 2010).

In this context, computational offloading is a software-level solution which to some extent mitigates the problem of resource constraint mobile devices by remote execution of the complete or partial application at available remote computing infrastructure (Chun & Maniatis, 2009; Kovachev, Cao, & Klamma, 2012). Previously it experiments that remote execution can potentially reduce the power consumption and execution time for applications executing on weak smart mobile devices (Chun & Maniatis, 2009). The computational offloading frameworks manage the remote execution of mobile cloud ap-

plications.

A basic computational offloading architecture as illustrated in Figure 1.4 is composed of a client subsystem configured and executed on the mobile device and a server subsystem executing in the remote cloud infrastructure (Chun, Ihm, Maniatis, Naik, & Patti, 2011; Cuervo et al., 2010). The client subsystem performs three major tasks to optimize the net system utility i.e. i) The client subsystem observes and estimates the network performance metrics for the mobile device. ii) The client subsystem monitor, estimates, and analyze the resource requirements of mobile applications regarding CPU time on both the mobile device and the cloud server. iii) Using information from the previously explained two tasks the client subsystem decide to migrate a mobile application to execute in the remote cloud infrastructure so that the total application execution time is minimized (Shi et al., 2014). The server subsystem is explained in the next subsection.



Figure 1.4: Computational offloading sub-systems in MCE.

1.1.4 Cloud computing and computational offloading as a service

Cloud computing is a business model introducing a XaaS service delivery model while simultaneously charging using an on-demand pay-as-you-go method and deployed across different deployment models (Mell & Grance, 2011). Cloud computing systems revolutionize the business life cycle by reducing the capital investment in infrastructure while maintaining additional focus on business services and strategies (Yousafzai, Chang, Gani, & Noor, 2016b). Given these prominent features, and the penetration of battery constrained smart and ubiquitous mobile devices, cloud computing systems expands into the business model of MCC. Cloud computing offers it services (i.e. infrastructure, platform,

or software) to the mobile device such that the mobile devices consume these services to migrate a mobile application for battery saving and improvement in execution time.

The server subsystem as introduced in the previous section 1.1.3 also illustrated in Figure 1.4 is provisioned through OaaS service delivery model. The OaaS computational offloading service on the cloud side executes the migrated application straight away after receiving the migrated application and returns the execution results or execution state to the client sub-system so that the application can be resumed on the mobile device.

1.1.5 Process migration

Process migration is the act of transferring an active running process between two machines and restoring the process from the point it left off on the selected destination node (Vasudevan & Venkatesh, n.d.). In conventional distributed environments such as cluster and grid computing environments process migration is utilized to load balance, fault recovery, resource sharing. The underlying technique behind process migration is checkpointing. Checkpointing consists of saving a snapshot of the application's state so that it can restore/restart from that point in case future. This is particularly important for the long running application that is executed in a vulnerable computing system. The most basic way to implement checkpointing, is to stop the application, copy all the required data from the memory to reliable storage (e.g., Parallel file system) and then continue with the execution (Plank, Beck, Kingsley, & Li, n.d.). In this thesis, we exploit this concept for an MCE and migrate an application from resource-constrained mobile device to resource-rich remote computing platforms.

1.2 Research motivation

Local execution of compute intensive mobile applications on mobile devices via native resources either is impossible or leads to fast battery drainage due to native resource incapacitation. Thus, empowering computing capabilities of mobile devices become vital

necessity to realize uninterrupted execution of compute intensive computations on mobile devices without quick battery drainage, which is possible by exploiting the computing power of remote computing infrastructure.

Compute intensive mobile applications that require intensive computational resources, particularly CPU, RAM, storage, and battery to complete the expected resource intensive computing operation. For instance, image processing applications, 3-D rendering applications, and video editing applications are exemplary compute intensive applications. Functionality and operations of compute intensive applications are currently limited due to resource scarceness of mobile devices. Execution of existing compute intensive applications immediately drains the mobile battery that significantly degrades the quality of user interaction. Hence, mobile resource augmentation becomes necessary.

In recent years, MCC paradigm has been introduced to augment (extend) the capabilities of mobile devices, by taking advantage of high-speed wireless communications and high-performance cloud platforms (Chao & Sun, 2013) and (Lloret, Garcia, Tomas, & Rodrigues, 2014) to help gather, store and process data for the mobile devices (Rahimi, Ren, Liu, Vasilakos, & Venkatasubramanian, 2014) and (Kumar, Liu, Lu, & Bhargava, 2013). In this new paradigm, Android-based smartphones have opened real-world venues for cloud-based mobile applications mainly because of the open source nature of Android. However, most of the computational augmentations mechanism empowering Android-based smartphones depends upon the application runtime environment in the Android stack.

For a mobile device, the simplest means of augmentation is using mobile cloud through service-oriented or client/server patterns. Such augmentation is feasible assuming that the requested service, or application source code, or application binaries are available on the server. Despite its simplicity, this approach may result in computational losses and increasing waiting times in case of network disconnection or service disruption. The

most common computational offloading mechanism is code migration, which migrates intermediate level instructions between a mobile device and a server. These intermediate level instructions must be executed on the same type of Application Level Virtual Machine (ALVM) (i.e., DVM) on both the mobile device and server. The literature also reveals offloading mechanisms that consider thread-state migration or thread-state synchronization. Both code migration and thread-state synchronization are highly dependent on ALVMs. That is, the computational offloading mechanisms of Android-based smartphones depend on DVM. This dependency invalidates the offloading mechanisms for the newly launched ART. Apart from being a discontinued product, DVM consumes extra time and energy because of the JIT compilation of DEX bytecode into machine instructions upon every invocation. With regard to this problem, Google has introduced ART featuring AHOT compilation to native instructions in place of DVM. ART has obvious benefits regarding execution time and battery consumption. ART uses AHOT compilation to transform device-independent DEX code into device-specific machine binaries (F., 2014).

1.3 Statement of the problem

In MCC, empowering computing capabilities of mobile devices, especially smartphones and fulfilling required computational resources of compute intensive mobile applications typically undertaken by offloading the compute intensive parts of the application to remote computing infrastructures. Most of these offloading techniques leverage the underlying application environments which provides the abstractions of machine-independent intermediate codes which can be migrated to and forth between the mobile device and remote computing devices. However, mobile device vendors are dropping these application execution environments to improve application performance and device battery life. Apple iOS is already following the native application execution mode and Google

recently introduced Android Runtime (ART) in favor of its the Dalvik runtime (DVM) environment. ART features ahead of time (AHOT) compilation to device specific binaries (Google, 2013), to improve application execution performance and battery consumption (Buckley, 2013). The obsolescence of DVM creates a gap as all migration primitives (e.g. Method or Thread Based) or any other, based on DVM are not compatible with Android ART. Therefore, to enable application migration for current and future Android releases a process level migration (which is platform dependent) is required. The process migration based offloading mechanism should transparently migrate a running application (process) from resource-constrained mobile device to resource-rich computing infrastructure where the application source code is neither required and can be annotated or not.

1.4 Statement of objectives

In this research, we aim to propose a computational offloading mechanism for the Android Runtime Environment for efficient execution of compute-intensive mobile applications in the resource-scarce mobile environment. We define following objectives that are to be achieved to attain the aim of this research.

1. Review the computational offloading frameworks in MCC for acquiring the insight on the state-of-the-art concerning migration mechanism during the execution and migration of mobile application on remote infrastructure.
2. Investigate the computational offloading in general and then investigate the major existing computational offloading mechanisms that can be used to migrate a mobile application to the remote infrastructure.
3. Design and develop a lightweight process migration based computational offloading solution to minimize the execution time,energy consumption and improve the compute power of the mobile device.

4. Evaluate the execution performance of the proposed lightweight process migration based computational offloading framework from two views of application execution time, energy consumption and compute power via experimental testbed on an Android-based smartphone.
5. Validate the results of performance evaluation of the proposal framework based on execution time and energy consumption using statistical analysis.

1.5 Proposed research methodology

We review the credible research efforts in Chapter 2 to gain insight into the computational offloading mechanism domain and determine the significant weaknesses and shortcomings of the recent mobile empowerment approaches. We review recent literature collected from online scholarly databases, particularly IEEE, ACM, Elsevier, and Web of Science to identify inefficiencies of computational offloading mechanism domain and identify the most critical inefficiency to address in this research.

In Chapter 3, we analytically analyze the identified inefficiency in the existing computational mechanisms to demonstrate its significance on efficient computational offloading mechanisms. For this purpose, experimental applications are designed to demonstrate the research gap, and the impact of existing computational offloading mechanism on application execution response time, application energy consumption time and data transfer.

In next phase of the research, Chapter 4 proposes a transparent, lightweight process migration based computational mechanism that enables the native mobile application to migrate transparently from mobile device to remote computing infrastructure and vice versa. The aim of the proposed solution is to minimize the execution time and energy consumption of an application and increase the compute power of the mobile device in an MCE. The execution time is minimized by exchanging the checkpointed process from the mobile device to remote device, where the execution is resumed on the server side.

The process is then re-checkpointed on the server side and are re-synchronized back with the mobile device and restarted. Furthermore, the lightweightness is measured as the overhead energy consumption and application execution time added up by the proposed framework in total energy consumption and total application execution time of the experimental applications.

We implement and evaluate the proposed framework in a real MCC testbed the details are listed in Chapter 5 and Chapter 6. A set of standard computation benchmarks along with a prototype matrix multiplication application are utilized to evaluate the proposed framework. We synthesize the time and energy results of execution using our framework with the results of local execution on the mobile device. Moreover, we build a statistical model to validate the results of performance evaluation. The statistical model is generated using linear regression model which is a predominant observation-based modeling method. The statistical model is validated using split-sample validation approach. The statistical regression model is also validated against empirical results.

1.6 Layout of thesis

This thesis is a detailed study on "A Lightweight Process Migration based Computational Offloading Framework for Mobile Device Augmentation" therefore the thesis has been organized into chapters for a clear understanding of the matter. This thesis is composed of seven chapters that are organized as follows:

- Chapter 2 presents a review of the state-of-the-art computational offloading frameworks proposed for MCC and investigates the critical aspects of the frameworks with respect to migration mechanism issues due to user mobility. We also classify the frameworks and devise a taxonomy. The frameworks are compared on the basis of the parameters derived from the thematic taxonomy. The open research issues to computational offloading frameworks are also identified and discussed in

the chapter.

- Chapter 3 analyzes the computational offloading in general in a MCE. The computational offloading mechanisms based on migration mechanisms is also studied. The analysis shows that the state-of-the-art frameworks lack features to handle the to migrate an ART-based mobile application on a cloud server. The research gap is being experimentally demonstrated, and a proof-of-concept process migration mechanism is presented to show the impact of process migration on application execution time in MCE.
- Chapter 4 presents a lightweight process migration based computational offloading mechanism that aims to solve the issue of computational offloading for native mobile applications during the execution of the application in MCC. It explains the architecture and algorithms of the proposed solution. The distinct features of the proposed solution are also highlighted and discussed.
- Chapter 5 reports on the data collection method for the evaluation of the proposed solution. We explain the tools used for evaluating the proposed solution, data collection technique and the statistical method used for the data processing.
- Chapter 6 presents the effectiveness of the proposed solution by analyzing the experimental results reported in Chapter 5. It analyses the different aspects of the proposed solution regarding application execution time, energy consumption, compute power, and contributing factors. The performance of the proposed solution is also compared with the state-of-the-art solutions in various scenarios.
- Chapter 7 concludes the thesis by reporting on the re-examination of the research objectives. It summarizes the findings of the research work, highlights the signifi-

cance of the proposed solution, discusses the limitations of the research work and proposes future directions for this research.

University of Malaya

CHAPTER 2: LITERATURE REVIEW

Computational offloading in MCE is a hard problem, due to the complexity of modern mobile applications and their inter-dependencies; the heterogeneity between the mobile devices and the cloud infrastructure; the unpredictability and variability of the wireless connectivity; as well as the range of objectives of the actors in a mobile cloud ecosystem. Consequently, the problem has received a considerable amount of attention from the research community. This chapter presents an overview of computational offloading and its principal requirements using the taxonomy of issues on the client(mobile) device subsystem along with a taxonomy of computational offloading functions. Furthermore, we surveyed the recent literature and highlighted the key insights. Based on our analysis, we identify eight challenges for future investigation. These relate to energy efficiency; providing a better migration mechanism; mobility assisted server to server computational offloading; computation security on the server; understanding economic and motivational behavior for cooperation based mobile clouds; automatic application partitioning; partition scheduling for scalable resource management; and improved ARM emulation on the cloud side.

The remainder of this chapter is as follows. Section 2.1 presents computational offloading issues on the mobile device using a taxonomy. Section 2.2 present a taxonomy of computational offloading framework functions. The state-of-the-art computational offloading frameworks based on varied resources and the comparison is presented in section 2.3 and section 2.4, respectively. Major open research challenges are presented in section 2.5 and finally the chapter is concluded in section 2.6

2.1 Taxonomy of computational offloading issues in the client subsystem

Figure 2.1 presents the technical issues on the client subsystem using a taxonomy and provide a brief description about the issues and their underlying effects on the computational

offloading process.

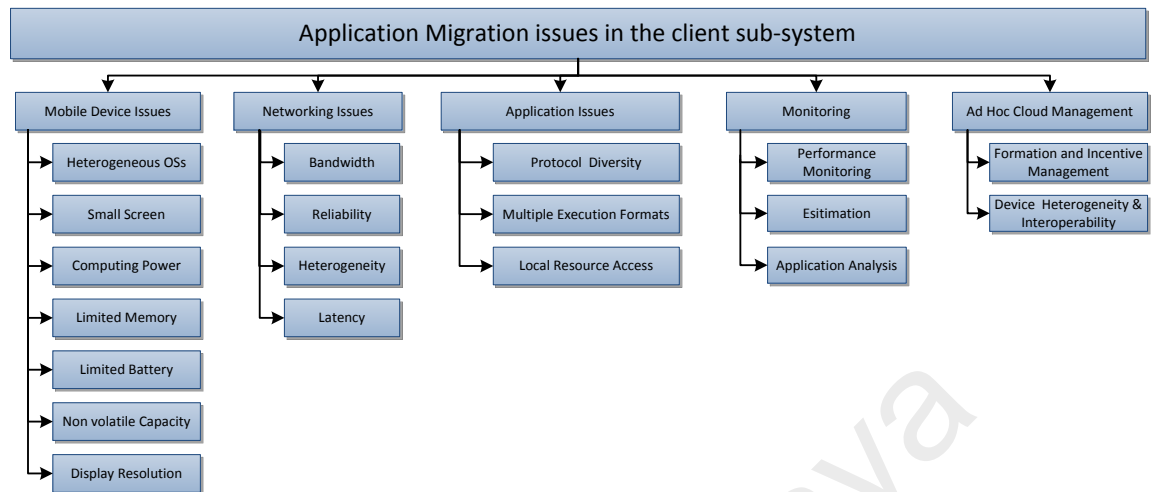


Figure 2.1: Taxonomy of computational offloading issues on the client subsystem in MCE.

2.1.1 Mobile device issues

This mobile device issues presented in Figure 2.1 are actually the reasons and motivations for the computational offloading concept in mobile computing. The limitations of the mobile device are due to the small form factor of mobile devices; the number and the potential of mobile device resources are limited, such as low battery life, storage, input methods, and screen sizes. The existing battery technology is not developing quickly (or accordingly) compared with the rapid evolution of the memory and CPU power. The processing of compute-intensive mobile application depletes battery quickly. Consequently, the limited battery necessitates the design of energy-efficient hardware, protocols, algorithms, and architecture for mobile environments. Another issue is device heterogeneity due to different software (operating systems) and hardware architectures. Thus, the same mobile application should be processed/tailored transparently and optimally for the users according to the functional and systematic characteristics of mobile devices (Luo & Shyu, 2011).

2.1.2 Networking issues

The dynamics of the wireless networks shown in 2.1, which affect the behavior of a computational offloading mechanism. For instance, the synchronization interval between applications execution on two end points can be effected by this issue cause a decrease in Quality of Service (QoS)/Quality of Experience (QoE) and cloud system throughput also.

2.1.2.1 Low bandwidth

Low bandwidth is one of the challenging problems, which may deteriorate the computational offloading ecosystem. The computational offloading decision is a trades off communication cost for computation gain. However, bandwidth in wireless radio networks is significantly scarcer than in wired networks. Thus, the communication cost may be higher, while the computation gain will be lower. Moreover, the network and execution prediction may be inaccurate, causing the performance of these systems to be degraded.

2.1.2.2 Network reliability

Reliability of the wireless network is an important issue given that the QoE of migrating applications strongly depends on the long-lasting network connectivity with an adequate bandwidth, packet loss, jitter, and delay. A mobile application depends on the access network and may be prevented from connecting to the cloud provider because of flow congestion and failures, which are habitual with wireless radio networks. This disconnection is recurrent in places, such as subways and tunnels, and providing a reliable wireless connectivity that is scalable and cost efficient while the learner is commuting is a challenge.

2.1.2.3 Heterogeneity

The users behind the smart handheld devices can migrate the application via a range of radio interfaces available on the device, such as General Packet Radio Service (GPRS),

Long Term Evolution (LTE), WiFi, and WiMax. In such heterogeneous network availability, deploying an efficient connectivity algorithm on the computational offloading client subsystem and server subsystem becomes imperative, which allows the users to permanently stay connected as well as seamlessly and transparently switch established network connections from one interface to another while staying connected/attached to the server subsystem.

2.1.2.4 High access latency

Longer access latencies in wireless radio networks are a major barrier faced by computational offloading in MCE. Longer delays and connection interruption significantly deteriorate the QoS and QoE of the mobile application. Humans are sensitive to this quality deterioration caused by jitter and longer delays that are tough to regulate in the wireless and Wide Area Network (WAN). Conversely, bandwidth is significantly enhanced in modern access networks; latency is unlikely to improve (Bourguiba, Agha, & Haddadou, 2012).

2.1.3 Application issues

Besides the structural and architectural computational offloading issues discussed in the above subsections. The application itself is a matter to be considered due to its complexity such as protocol diversity, packaging mechanism, access to local mobile device hardware, and linkage with local libraries. In addition to these, the most important issues with respect to the application are whether the application is available as a compiled intermediate code to be executed on an ALVM or the application is compiled in a binary executable packaged as Executable and Linking Format (ELF).

2.1.3.1 Protocol diversity

On the application layer Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) are used to transport packets between connected devices. In a MCE, mobile applications may utilize different transport protocols depending upon their requirement. Further from a communication point of view the computational offloading transaction can be either performed using an RPC or SOAP based mechanism. This diversity in protocols poses an issue for a comprehensive computational offloading system for mobile devices.

2.1.3.2 Multiple execution formats

The applications on mobile devices are available in multiple execution formats. Such as applications which run in an ALVM. These applications are relatively easy for migration. Another format is the native compiled mobile application. Natively compiled application are platform dependent and requires homogeneous hardware and software platform on the server side. The last format is about interpreted applications, this may involve bash scripts, Perl or any other system shell.

2.1.3.3 Local resource access

Local resources can be of two type i) hardware resources like(sensors, camera, microphone) and, soft resources such as shared libraries. The application calling for this resources may hinder and deteriorate the computational offloading process. Careful consideration is required to tackle this aspect of applications while designing a computational offloading system.

2.1.4 Monitoring

Monitoring and analyzing the mobile application, system utilization, and network performance is a critical issue which needs to be addressed while designing a computational offloading system. The monitoring data is used to decide dynamic partition sizes, offload-

ing decisions and long-term planning of the computational offloading system. However, monitoring also comes with an overhead energy consumption and compute capacity sharing due to the execution of the monitoring application on the computing infrastructure. If exhaustive monitoring is performed it might provide best offloading decisions but the net benefit may be outnumbered due to the energy consumed by the monitoring process itself. Careful consideration is required while addressing this issue.

2.1.5 Ad-hoc mobile cloud management

When the computational offloading system opts for utilizing the ad-hoc mobile cloud resources, it becomes extremely important to manage the ad-hoc cloud resources. There are two key management issues

2.1.5.1 Formation and incentive management

Mobile devices are becoming powerful day by day and can form a self-organizing mobile ad-hoc network of nearby devices and offer their resources as on-demand services to available nodes in the network. In the ad-hoc mobile cloud, devices can move after consuming or providing services to one another. During this process, the problem of incentives arises for a node to provide service to another device (or other devices) in the network, which ultimately decreases the motivation of the mobile device to form an ad-hoc mobile cloud.

2.1.5.2 Heterogeneity and interoperability

The mobile device differs in their underlying hardware and software platforms. This difference emerges the issue of heterogeneity and interoperability. In case, when a mobile device is consuming and providing service in an ad-hoc MCE, then it is necessary to understand and model the behaviors such that the computational offloading system is interoperable (Yousafzai, Gani, et al., 2016; Yousafzai, Chang, Gani, & Noor, 2016a).

2.2 Taxonomy of computational offloading Functions in MCE

In this section, we outline the main functionalities embodied in computational offloading systems for MCE using a thematic taxonomy illustrated in Figure 2.2. The taxonomy represents the decomposition of steps involved in a computational offloading system depicting logical functional elements which are required by any computational offloading mechanism. These functional elements should coordinate to provide a complete application solution in line with migration objectives. A brief discussion on each of these computational offloading function is presented in the subsequent subsections.

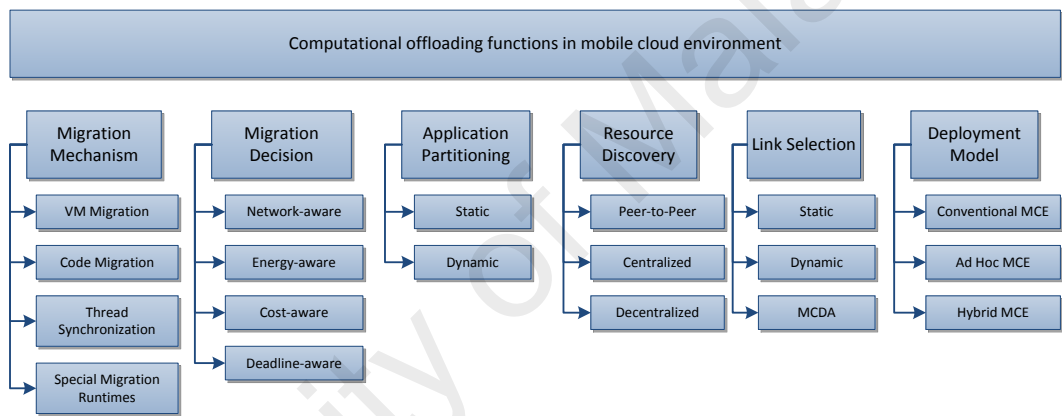


Figure 2.2: Taxonomy of computational offloading functions in MCE.

2.2.1 Migration mechanism

A migration mechanism is the techniques which define the underlying granularity or abstraction through which the application data and application code is bundled and transferred from one device to another device for remote execution of the application. Current state-of-the-art computational offloading mechanism's achieves the remote execution of application either using Virtual Machine (VM) Migration, Code Migration, or Thread Migration.

2.2.1.1 VM/Phone clone migration

VM/Phone clone migration use virtualization technology to keep a synchronized mirror for each connected smartphone on a computing infrastructure allowing some operations to be performed directly on the mirror. A high-level diagram of components and mechanisms involved in VM based augmentation of the mobile device is presented in Figure 2.3. In essence as presented in Figure 2.3, cloud-based smartphones augmentation using VMs/phone clones either delivers and obtain an overlay image (the difference between two consecutive VM images) or a replay trace to and from the computing infrastructure. Via the overlay, smartphones synchronize part of execution on the remote VMs and vice versa. However, this mechanism is being hindered by the amount of data transfer (Hung, Shih, Shieh, Lee, & Huang, 2012).

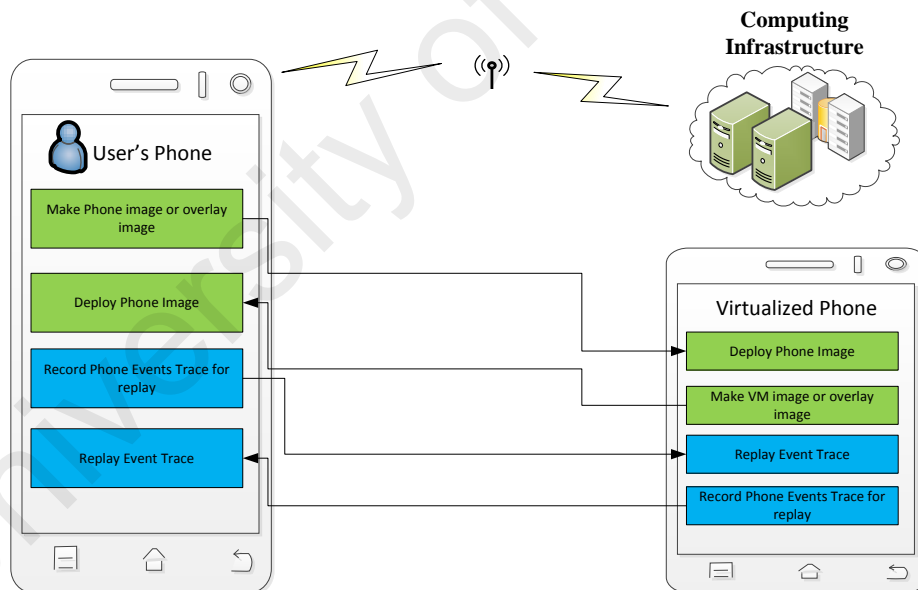


Figure 2.3: VM based augmentation of mobile devices.

2.2.1.2 Code migration and delegation

The most popular and common technique to leverage the computational power of cloud from mobile devices is delegating code execution to remote cloud servers either by migrating the platform independent intermediate code or use a service oriented client/server

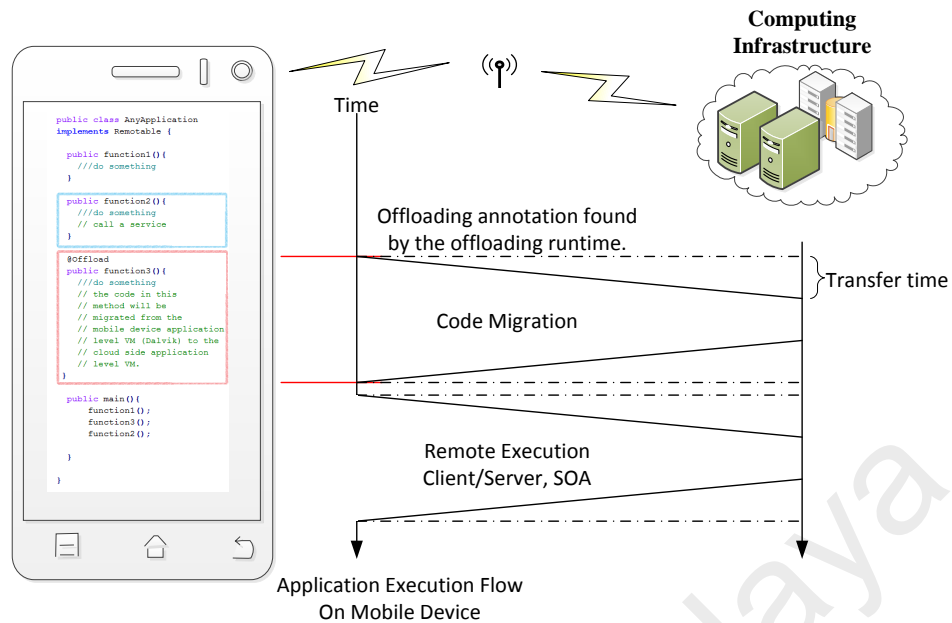


Figure 2.4: Remote execution using code migration and delegation.

setup. An illustration of code migration and remote execution is presented in Figure 2.4. In this figure the difference between the code migration and remote execution via plain client/server or SOA fashion is clearly presented. In code migration, in case the server is disconnected or not available the mobile application can be re-executed locally on the mobile device while in the SOA fashion the execution halts until the service become available, in both these fashion upon interruption the remote computation performed is lost.

2.2.1.3 Thread state migration

In MCE, thread migration is the migration of low-level thread state (heap contents, stack, descriptors, register values) from one ALVM (e.g. DVM, Java Virtual Machine (JVM), .Net Runtime) to an ALVM residing on a remote computing devices. A generalized view of thread migration is presented in Figure 2.5. The thread synchronization can cause a lot of compute and energy overhead in the mobile devices as the ALVMs need to heavily modified in order to synchronize threads between the two connected endpoints.

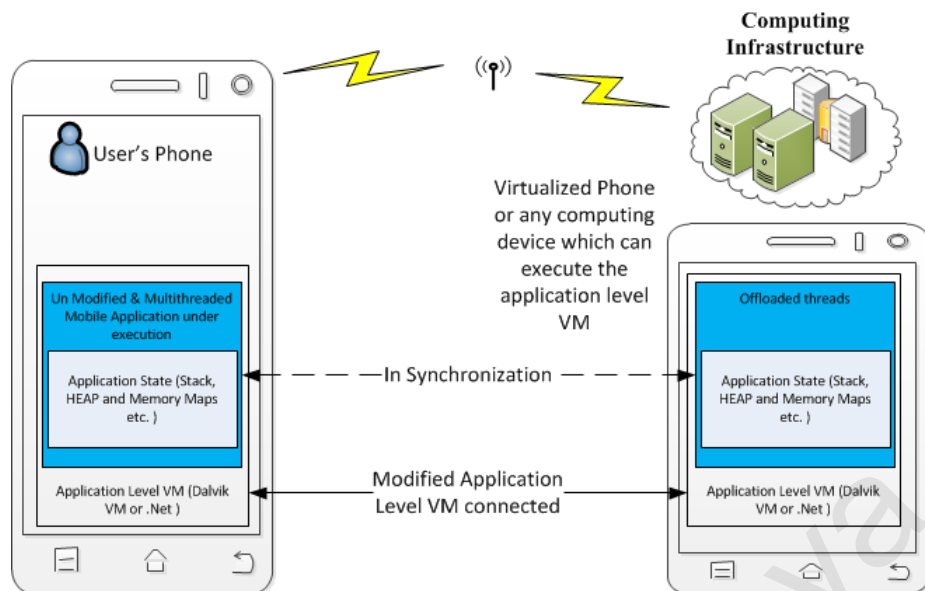


Figure 2.5: Generalized representation of thread synchronization used in MCE.

2.2.2 Migration decision

This is the objective function which actually triggers the process of computational offloading. Migration decisions are dependent upon the resources discovered, the link selected, partition size and the objective of the migration. Current state-of-the-art computational offloading mechanism either employs a network aware, energy aware, cost aware, or deadline aware computational offloading decision mechanism (energy efficiency, or response time minimization). Migration decision is agnostic to the migration primitive.

2.2.3 Application partitioning

Application partitioning is a technique of splitting up the application into separate components, while preserving the semantics of the original application (Liu et al., 2015). Computational offloading may use partitioning of the mobile application to divide a mobile application into separate discrete partitions, which can be executed independently in a distributed MCE. The original mobile application source may or may not be designed, implemented to be partitioned; this depends upon the migration mechanism, which is being used. Partitioning of the mobile application is a pre-phase of computational offloading in the contemporary computational offloading systems for MCC. The current

state-of-the-art computational offloading mechanism requires either a static application partition scheme or a dynamic application partition scheme. The static application partition scheme is defined by the programmer through annotations and other programming skeletons; in addition, for already developed applications, the application needs to be modified and recompiled for partitioning. On the other hand, in the dynamic application partitioning scheme the application partitions are determined on run-time and the already developed application does not need modification.

2.2.4 Resource discovery

In the context of computational offloading in MCE, resource discovery activity involves searching for the appropriate cloud resource/service, or ad hoc mobile node, that match the requirements of applications which is being migrated. The resource discovery methods can be engineered based on various network models including centralized, decentralized, and peer-to-peer with varying degree of cost, scalability, fault-tolerance, and performance (Ranjan & Buyya, 2009).

2.2.5 Link selection

Modern mobile devices are equipped with multiple radio interfaces, e.g. WiFi, Bluetooth, and 3G/4G network interfaces. In order to migrate an application from the mobile device to the remote computing the device can employ any of these interfaces. The link decision can be either static (locked to a single interface) or dynamic where the selection of interface is based on an objective or based on a multi-criteria decision analysis (Drissi & Oumsis, 2015).

2.2.6 Deployment model

As previously described in Section 1.1.1 MCC systems can be deployed in three different deployment models. The computational offloading mechanism strictly depends upon the

deployment model as with the change in deployment model may force the administrators to change the computational offloading mechanisms. For instance in ad-hoc MCE, the VM migration based computational offloading is not practical due to the fact that current virtualization technologies for mobile devices are in infancy (Gu & Zhao, 2012). Furthermore, the data transmission overhead of VM migration(due to VM image sizes) restricts its practicality and can not be beneficial in highly mobile ad-hoc mobile clouds or low bandwidth mobile clouds.

2.3 State-of-the-art computational offloading mechanisms in MCE

In this section, we review the state-of-the-art computational offloading mechanisms in MCE.

2.3.1 AlfredO

AlfredO (Rellermeyer, Riva, & Alonso, 2008) depends on the R-OSGi (Rellermeyer, Alonso, & Roscoe, 2007; Alliance, 2009) a device agnostic, distributed, and loosely coupled Java based middleware. From architectural point of view, AlfredO provides a multi-tier service architecture, consisting of the presentation tier, logic tier, and data tier. AlfredO envision to utilize mobile phones as generic interfaces to remote computing devices, while preserving security to some extent. AlfredO distributes Java applications using Service Oriented Architecture (SOA) to be executed by the R-OSGi framework. The application components are dynamically distributed multiple remote computing resources within the boundaries of the defined tiers. As the framework is based on R-OSGi and OSGi, it requires the mobile device to be equipped with JVM, and support the concepts of R-OSGi. Furthermore, AlfredO restricts its applicability as it requires the application developers to provide a static application partitioning scheme for local and remote executions. Therefore, a fine-grained dynamic partitioning is not possible with AlfredO.

2.3.2 Misco

Misco (Dou, Kalogeraki, Gunopulos, Mielikainen, & Tuulos, 2010) employs a MapReduce (Dean & Ghemawat, 2008) programming model to execution tasks. MapReduce splits tasks into similar sub-tasks and executes them in a parallel fashion. The MapReduce task execution takes place in two phases i.e. a map phase, and a reduce phase. Map phase yields in intermediate results while, the reduce phase, aggregates the intermediate results for the final solutions. The Misco server component, enables the end mobile users to submit new tasks through a web interface, while multiple mobile worker nodes process these tasks. At a particular time, an individual worker node executes either a single map or reduces task. From a computational offloading perspective, the underlying difference between Misco and other described frameworks is the way how tasks are submitted and managed. Due to the central Misco server, the framework is prone to single point of failure. Misco is developed in Python while ensuring a compatibility interface with existing computational offloading systems. Regardless the fact that Python is lightweight and does not have a drastic resource consumption impact on mobile devices, it is still a challenging and costly task to port existing mobile applications to this framework.

2.3.3 MAUI

The main goal of MAUI (Cuervo et al., 2010) is to be as energy efficient as possible. MAUI prototype introduced by the author is for instance limited to Microsoft .Net runtime environment (an ALVM). MAUI employs a method-level granularity to migrate the .Net Common Language Runtime (CLR) machine independent code. Methods that might need to be offloaded are annotated as remote. The offloading decision is made through an optimization framework considering current network dynamics, or the cost (regarding energy and time) required to migrate a method annotated as remote. MAUI main focus is on the migration mechanism part (similar to our research objective) and not on

details on how to manage server resources and network issues. If the connection to remote server is lost, MAUI restart the computation on local device, resulting in a delay and computation lost. MAUI utilized a profiling and solver model, where all executions are monitored and are recorded by the profiler. The historical data is then utilized as input for the optimization solver to take the offloading decisions. The most distinguished feature of MAUI is its simplicity and does not require any apriori knowledge to make use of MAUI. However, a limited few internals need to be known by the application developers to utilize the computational offloading framework. Albeit MAUI is prototyped in the .Net programming language, but the concept can be easily adapted to other programming environments. However, it is time consuming and costly and may not be of interest for platform and application vendors. Lastly, if MAUI is ported to other programming environments, compatibility issues needs to be addressed for cross-platform communications.

2.3.4 CloneCloud

CloneClouds (Chun et al., 2011) aims to offload a compute intensive task to platform clones residing in the remote computing infrastructure to minimize the application execution time and energy consumption on the mobile device. The cloned VM in the cloud need to be as similar to the mobile device as possible both from architectural point of view and contents. CloneCloud can operate on and is able to offload unmodified applications, by utilizing a mixture of static and dynamic application partitioning algorithms. The static analysis of application intermediate code is to store migration and reintegration points based on three properties. First, the partition must not contain device specific feature calls. Second, the partitions methods which access native resources needs to be executed only on the local device. Lastly, nested partitions are not permitted. The dynamic analysis is performed for cost estimation of the application partitions using various execution characteristics. On the basis of the collected data, the optimization solver exer-

cise the offloading decisions. The CloneCloud prototype is based on DVM and Android. Therefore, theoretically CloneCloud is able to operate on every DVM based Android mobile application. The migration and reintegration markers are incited using special DVM commands, introduced into DVM. Therefore, for CloneCloud, it is necessary to provide a modified DVM version, which brings in computational overhead to the DVM in terms of energy consumption and application execution time overhead.

2.3.5 ThinkAir

ThinkAir, introduced by (Kosta, Aucinas, Hui, Mortier, & Zhang, 2012), amalgamate the concepts of CloneCloud and MAUI and, while introducing the notion of scalability, in the amalgamate model. ThinkAir dynamically allocate cloud resources by dynamic initiation and termination of VMs. ThinkAir objective was to dynamically adapt according to workload variations such that the the performance of applications is increased and energy consumption is minimized. ThinkAir provides a static partitioning mechanism like MAUI. (Kosta et al., 2012) prototype has been developed for Dalvik and is based on Android. It eliminates the limitations induced from CloneCloud by combining the approaches from MAUI and CloneCloud. However, it incurs the VM provisioning overhead.

2.3.6 Cuckoo

Cuckoo (Kemp, Palmer, Kielmann, & Bal, 2012), a computational offloading framework with a slightly different goal compared to CloneCloud or ThinkAir. The main focus of Cuckoo is to achieve a transparent integration in the application development life-cycle and in Integrated Development Environment (IDE)s to enhance the ease in application development. Cuckoo does not utilizes a hardware level VM. Instead, it works with JVM, which can be provisioned low server resources. Cuckoo does not requires the same partitions to be executed locally and remotely. The IDE produce interfaces for the remote

partitions of the application. The application developers can either choose the same implementation or provide different implementations for the partitions depending on the locality of partitions where it would be executed. As a matter of fact, during the process of compilation, the resulting application will contain a library embodying the remote parts. In general, Cuckoo seems more efficient than executing the same implementations on all platforms without the possibility to optimize it for certain platforms. On the other hand, Cuckoo does not fulfill the requirement of an easy migration of applications to the cloud.

2.3.7 COSMOS

COSMOS (Shi et al., 2014) a VM-based computational offloading approach for Android aims to provide OaaS for minimizing offloading costs. COSMOS unlike other systems such as CloneCloud consolidate the workload to as few VMs as possible. COSMOS handle this allocation and scheduling problem by introducing a Master server, which manages all available resources and can prevent workload spikes. The migration mechanism has not been described in detail. However, the COSMOS concepts eliminate the delay for starting new resources due to the Master server and focus on cost minimization due to a smart control of the resources. Lastly, due to the centralized server COSMOS are prone to single point of failure.

2.3.8 Hyrax

Hyrax (Marinelli, 2009) leverages local nearby mobile devices to form a cluster of mobile devices to run compute intensive tasks. Hyrax utilizes the fault tolerance principles of Hadoop to tackle the intermittent disconnections of highly mobile proximate devices. In the case, where proximate mobile cluster resources are not sufficient, Hyrax provides accessibility functions to augment the computational capabilities of the cluster through the remote clouds. The Hyrax server employs two client side processes of MapReduce, to

coordinate the execution of process on a mobile device cluster. The mobile devices connect to the server and other mobile devices via WiFi technology over an adhoc settings. The Hyrax seamlessly uses distributed resources and provides interoperability across the heterogeneous proximate mobile cloud platform. However, the Hyrax has high computational overhead because of the complexity of Hadoop system.

2.3.9 VM-based cloudlet

VM-based cloudlet (Satyanarayanan et al., 2009), address the limitations of computational offloading caused by network dynamics (WAN latency, jitter, packet losses) using a resource rich computing infrastructure in geographical proximity. VM-based cloudlet reduces latency and provides a single hop, high bandwidth wireless access to the nearby mobile devices. Cloudlets provides before use modifications and after use restoration to initial state, ensuring the infrastructure stability. Cloudlets utilizes the dynamic VM synthesis approach to synchronize the mobile device and VM residing in the cloud. Dynamic synthesis approach sends a small VM overlay to the cloudlet that already has the base VM from which the overlay VM was derived. The cloudlet infrastructure then derives the launching VM by applying the overlay to the base VM. The approach reduces the VM migration overhead caused by large VM image sizes. Moreover, the performance of dynamic synthesis is further improved by employing parallel processing and by employing caching and pre-fetching techniques. However, privacy and access control are issues with the migration of entire application execution environment that needs to be addressed.

2.3.10 COMET

COMET (Gordon, Jamshidi, Mahlke, Mao, & Chen, 2012) aims on the transparent migration of the multi-threaded DVM-based Android applications to the remote computing platforms. The framework takes a runtime offloading decision while considering the workload of machines and the characteristics of the application. COMET leverage the

benefit of Distributed Shared Memory (DSM) techniques to share the resources and maintain the consistency among the endpoints. Multiple threads of application in COMET can simultaneously coordinate on a field granularity level in order to handle the reader and writer consistency (Courtois, Heymans, & Parnas, 1971). COMET constitute a throughput optimized scheduler which migrates the threads between the endpoints. The scheduler utilizes the historical behavior of thread execution in offloading decision process. The historical behavior is captured by constant monitoring about how long a thread executes on local mobile device without calling a native method. COMET provides transparent and seamless migration of partial and multiple threads of an application. However, the framework does not consider the security concerns and generates huge volume of network traffic.

2.3.11 Virtual smartphone

Virtual Smartphone over IP(Chen & Itoh, 2010) provision a user to initialize an Android VM, which we call Virtual Smartphone (VSmart), in the cloud. The VM act as a dedicated remote execution environment of the user's physical smartphone. The end user can offload an entire mobile application to the VSmart VM and manage the application through remote desktop sharing such as VNC (Richardson, Stafford-Fraser, Wood, & Hopper, 1998). VSmart synchronizes sensor readings, such as GPS, accelerometer, orientation, magnetic field and temperature from the physical to virtual instance. The application installed in the remote VM generates the same sensory results as of the physical smartphone. Virtual Smartphone can be utilized to improve processing power, to preserve smartphone's battery life, and to avoid untrusted applications from accessing local data on the physical device, and to prevent data leakage. However, this approach is being hindered by the heterogeneity of mobile device hardware.

2.3.12 AIOLOS

AIOLOS (Verbelen, Simoens, Turck, & Dhoedt, 2012), a mobile middleware framework, which has an adaptive application migration decision engine that considers dynamic available resources of the server and varying network conditions. The framework predicts the execution time for each method call at both local and external execution considering argument size. Subsequently, on the basis of the estimated results, AIOLOS selects the optimum execution location. The migration decision algorithm objective is to minimize the application execution time and battery consumption. Similar to many other computational offloading systems AIOLOS uses a history-based profile to aid in predicting the local execution time and network dynamics. The predication mechanism adds up the computational complexity of the proposed solution while the central middle-ware leads to portability issues as implementation is dependent on the platforms and protocols.

2.4 Discussion on computational offloading frameworks

The classification discussed in Section 2.2.1 is summarized in Table 2.1. The table also outlines the dependencies and limitations in current state-of-the-art offloading mechanisms.

Table 2.1: Comparison of Computational Offloading Mechanism used in MCE

Computational Offloading Mechanisms	Application Modification	Dependency	Major Drawback
VM/Phone Clone Migration	No	Hardware Virtualization	High communication overhead caused by large VM image sizes and the computational overhead it takes to generate migration overlay state.
Code Migration and Delegation	Yes	ALVMs	Static Partitioning using annotations for high-end mobile devices with intermittent connectivity is not useful. Further, in the case of SOA or client-server setup disconnection/server failure computation is lost and restarted.
Thread Synchronization	No/ Auto	ALVMs	The computational overhead (of profiling and analyzing) caused by the modification of the ALVM for executing an application which does not need offloading.

Cloudlets (Satyanarayanan et al., 2009) , Paranoid Android (Portokalidis, Homburg,

Anagnostakis, & Bos, 2010) , Virtual Smartphone (Chen & Itoh, 2010), and Phone Mirroring (Zhao, Xu, Chi, Zhu, & Cao, 2012) utilizes VM/phone clone migration class of computational offloading from mobile device to a remote computing infrastructure. Cloudlets(Satyanarayanan et al., 2009) proposes the use of nearby resource-rich computers, to which a smartphone connects to single hop wireless LAN, and migrates a current system image. Paranoid Android (Portokalidis et al., 2010) uses QEMU to run replica Android images in the cloud to enable multiple exploit and attack detection techniques to run simultaneously with minimal impact on phone performance and battery life. Virtual Smartphone (Chen & Itoh, 2010) uses Android x86 port to execute Android images in the cloud efficiently on VMWare ESXi virtualization platform. Phone Mirroring (Zhao et al., 2012) framework which keeps a synchronized mirror for each connected smartphone on a computing infrastructure allowing some operations to be performed on the mirror directly.

Similarly (Cuervo et al., 2010; Kosta et al., 2012; Simanta, Ha, Lewis, Morris, & Satyanarayanan, 2013; Kovachev, Cao, & Klamma, 2012; Kovachev, Yu, & Klamma, 2012; Lee, 2012; Verbelen, Stevens, Simoens, Turck, & Dhoedt, 2011; Verbelen et al., 2012; Kemp et al., 2012; Flores, Srirama, & Buyya, 2014) use code offloading to improve performance and energy consumption. Most of these migration and delegation based offloading attempts rely, on programmers – to specify program partitions using code annotation and skeletons to adapt the program for specification of local and remote application partitions. Whereas some of the code offloading based mechanisms (Saab, Saab, Kayssi, Chehab, & Elhajj, 2015) replicates the application binary, intermediate representation or source-code at the server and perform remote execution in a client-server or service oriented fashion. Besides the significant improvement in the execution time, code migration approach do not support already developed applications which does not consider the notion of MCE. In addition, static partitioning by the programmers for high-end mo-

mobile devices with intermittent connectivity might not be useful. Further, in the case of SOA or client-server setup disconnection, or server failure remote computation may be lost and the execution on the phone might need to be restarted from scratch. Finally, code migration in the case where there is no application ALVM is not an easy task due to the underlying heterogeneity of hardware and software platform across the connected systems.

Likewise, CloneCloud (Chun et al., 2011) and its variants (Yang et al., 2014) and COMET (Gordon et al., 2012) exploit the concept of thread migration to improve the overall mobile device performance. The thread state migration mechanism is strictly dependent on ALVM, as these ALVMs provides the abstraction and interoperability of threads across different hardware platforms. These ALVMs needs extensive modifications to enable thread state synchronization mechanisms. CloneCloud (Chun et al., 2011) uses a combination of static analysis and dynamic profiling to partition applications at runtime, the application partition is migrated as a thread from the mobile device at a chosen point to the clone in the cloud, executing there for the remainder of the partition, and re-integrating the migrated thread back to the mobile device. On the other hand, COMET (Gordon et al., 2012) leverages the underlying memory model of Dalvik runtime and modifies the Dalvik running on both phone and server to implement DSM for enabling thread synchronization between the mobile device and the server. The modification of Dalvik on smartphone side has a very worse impact on an application which cannot be offloaded due to reasons such as not designed for offloading, or the server is disconnected, or because of any reason the application is running locally on the mobile device.

2.5 Research challenges

Through analyzing the literature we found that some research challenges are about computational offloading systems. These challenges are presented in the forthcoming subsec-

tions.

2.5.1 Energy efficiency

As the mobile device has increasing processing capability, the energy consumption becomes the major issue for mobile applications. Most device vendors look for approaches to increase the battery life. Besides inventing new battery technologies, there are many methods to save the energy consumption at the system and application layer. Computational offloading is considered as a critical approach to saving energy consumption on mobile devices (Kumar & Lu, 2010). By using the approach, the components of the application that consume a lot of energy, e.g., compute-intensive algorithms, are offloaded onto remote computing devices. However, the difficulty in this approach is to design effective mechanisms to monitor and profile the energy consumption for the applications on mobile devices. Designing the models for the estimation of energy consumption in data transmission is not easy as well. Both the profiled information and models are critical to partitioning the application for energy saving. We need to design the lightweight and energy efficient supporting computational offloading mechanisms. The costly function in a computational offloading is the computation partitioning optimization, offloading decisions based on network profile and device context, and diversity of application execution formats and runtimes. The issues of energy consumption in data transmission also need to be addressed in computational offloading.

2.5.2 Process migration based computational offloading

As illustrated in Table 2.1, the existing computation offloading mechanisms depends upon virtualization technology (i.e. application level or system level). However in the absence of the virtualization middleware software's these computational offloading solution becomes invalid. Further, mobile device OS vendors are now more focused toward integrated machine dependent native compiled mobile applications such as Google intro-

duced Android Runtime (ART) to reduce the application execution overhead in term of energy consumption and application mainly caused due to the repetitive process of JIT. Now with the introduction of such new runtime environments which executes native compiled applications, computational offloading mechanisms should be revisited. Compared with virtualization based computational offloading, process migration based offloading mechanisms consumes significantly fewer network and computing resources. Efficient decision making and replication of applications would render process migration a cost-effective choice over virtualization based computational offloading mechanisms. Furthermore, in volunteer and self-organizing mobile clouds, end users have limited bandwidth, so process migration ultimately becomes a feasible solution. Current process migration solutions issues are related to inter-process communication, open file descriptors, hardware and software platform heterogeneity, network socket states, and interoperability of network sockets.

2.5.3 Mobility-assisted server-to-server computational offloading

In MCE, due to intermittent connectivity and non-seamless wireless coverage, and mobility of the end user remote computing infrastructure (such as Cloud, Cloudlet, Ad-Hoc Mobile Nodes) may not be available or disconnected once an application is migrated to any of the mentioned remote computing infrastructure (Wang, Li, & Jin, 2014). However, this disconnection may results in loss of computation and deteriorate the end user QoS/QoE requirements. Furthermore, due to the mobility of the user a device cannot re-establish the connection with one-hop augmentation devices such as cloudlet or ad-hoc mobile nodes. To save the computation once performed mobility-assisted server-to-server computational offloading mechanism is the need of the day for MCEs. In addition a push based mechanism should be employed in the mobile devices to inform them about the migration of their computation from one server to another server based upon the device

mobility profile.

2.5.4 Secure computations on server side

Most of the identified security vulnerabilities can be fulfilled using available technologies. If the remote computing infrastructure is only used as a storage provider then, reliability can be guaranteed by using multiple providers and by replicating content (AlZain, Pardede, Soh, & Thom, 2012). Integrity can be assured by additionally storing hash values or digital signatures (Wohlmacher, 2000). Privacy and confidentiality can be preserved by applying strong encryption and by limiting the storage of data to trusted entities. The requirement for non-repudiation does not apply to storage providers but is comparable with applying digital signatures to preserve integrity. The requirement for a high isolation level finally maps to the use of strong passwords and two-factor authentication. In case the remote computing service does not only store data but does also process it, the situation gets more complex. One promising approach is the use of fully homomorphic encryption, which enables operations on encrypted data. Other solutions require the provision of the used encryption key to the remote computing infrastructure, which has an equal protection level as applying no encryption at all (Gentry et al., 2009).

2.5.5 Incentive management and resource discovery

The actors in MCE are often different parties having their benefits (Yousafzai, Chang, et al., 2016a). The service providers goal is to attain as much monetary benefit as possible with as low investment as possible. Toward their goal, they might squeeze their computational resources or reduce the service timing. In other words, service wishes to maximize resource utilization, but allocating too many VMs on a single physical machine will result in performance degradation and unpredictability due to the interference of collocated VMs with each other, which in turn impact the mobile application QoS/QoE. On the other hand, these resources are most of the time not free and sometimes not available. However,

modern mobile devices have much more resources than before. As a result, researchers have begun to consider the possibility of mobile devices themselves sharing resources using an ad-hoc mobile cloud model. Now the problem with the devices in the ad-hoc MCE is their mobility, which put questions about how to provide incentives to devices sharing their resources. This incentive problem also decreases the motivation for devices to join the self-organizing proximate MCE, provide, and consume services to and from other devices. So in this response, to solve the retribution and reward problem for devices in the proximate MCE, we proposed a directory based framework which will mitigate the issue of incentives to the nodes in the ad-hoc environment, even after their movement from one environment to another. The assurance of the incentives to devices will help in motivation for devices to join the self-organizing proximate MCE.

2.5.6 Automatic application partitioning

All the previous DVM based partitioning mechanism will not work for ART. To allow the already developed application to take advantage of the mobile cloud using computational offloading those application must be partitioned so that the migration/synchronization points are identified. These partitioning should be automatic such as employed in CloneCloud (Chun et al., 2011) and its variants to transform any mobile applications to a mobile cloud application. The partitioning mechanism should work in such a way when an application is installed on a mobile device using ART, the compilation process of Android runtime should be modified to annotate and add special migration primitives to the code. The schematic of the original ART process and the envisioned modified ART process is presented in Figure 2.6 (a) and (b) respectively. An illustration through pseudo-code of how the DEX code will look like before and after migration markers are inserted, is presented in Table 2.2.

By putting migration markers, the application will be migrated with much ease as

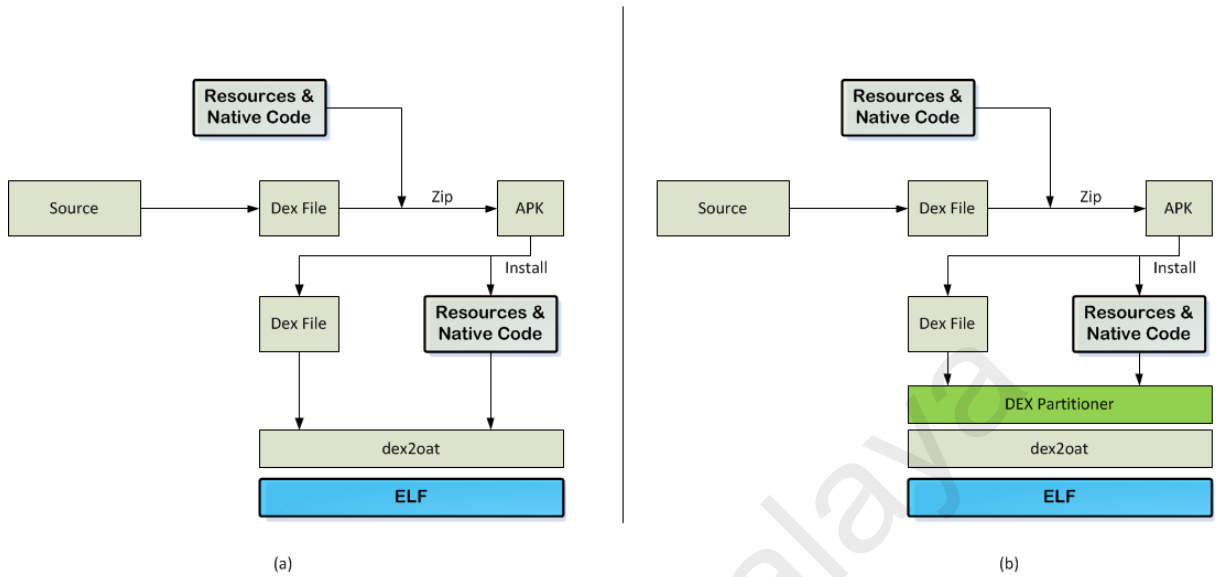


Figure 2.6: (a) The original ART schematics (b) Modified ART schematics.

Table 2.2: Pseudo-codic Illustration of Unmodified Original Code with no synchronization points and the modified code with synchronization points.

Original Code	Modified Code
Function Main()	Function Main()
do something...	Migration_Marker_Send do something...
if some_condition	if some_condition
return;	Migration_Marker_Receive return
end if	end if
File access	Migration_Marker_Receive File access
do something...	Migration_Marker_Send do something...
return	Migration_Marker_Receive return
End Function	End Function

the synchronization points will be defined the migration can be performed in a controlled fashion by a user level migration mechanism surpassing the kernel level state acquisition. That modified ART compilation process should be and must be same on both the mobile side and cloud side to generate a similarly partitioned application.

2.5.7 Scheduling partitions

In case the mobile cloud the user is connected is a complex mobile cloud scenario containing multiple types of resources as illustrated in Figure 1.3. The partitioning of the application in the above-discussed manner (section 2.5.6) will help us to make a graph representation of the application which will be useful in formulating the scheduling problem (Mahmoodi, Uma, & Subbalakshmi, 2016a). Once the partitioning of application is done, now two problems arise from the partitioned code i) to identify the dependencies between the partitions, (ii) schedule the offloading decisions/partitions over the resources available in the mobile cloud taking into account the partition dependencies (such that independent partitions can be executed in parallel) and other factors such as device context. A scheduling decision would of whether to offload a particular partition to CSP/ cloudlet / ad-hoc cloud node or not will be based on an objective function considering the dependencies between the partitions and the cost model. An illustration of such a schedule of an application with fourteen partitions on different types of resources available in the mobile cloud is presented in Figure 2.7.

2.5.8 Better ARM emulation in cloud

ARM is the most common processing architecture found in the current smartphone market (Forbes, 2013; Bent, 2012) and most of the time the cloud infrastructure is based on x86 architecture so to utilizes the current existing cloud infrastructure to truly envision the MCC ecosystem such as: phone clone mirroring and process migration, emulation of ARM is contemporary. However, the current emulators such as (Bellard, 2005) are

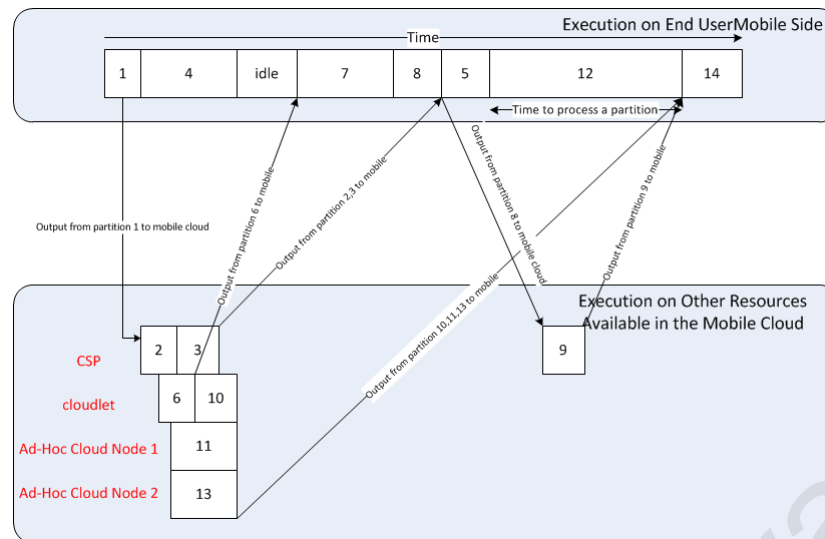


Figure 2.7: Schedule of automatically generated application partitions on mobile cloud resources.

designed to support generalized emulation and can be extended to any new platform with little efforts. This generalization comes at a cost of a performance degradation due to factors such as translation of guest instruction into intermediate code. This intermediate code generation can be avoided and an ARM specific emulators (such as Bochs (Lawton, 1996) PC-x86 emulator) with direct mappings from guest ARM instructions to the host x86, which will improve the performance of ARM emulation. A generalized schematic of the translation process of the current emulator (QEMU (Bellard, 2005)) used to emulate ARM instruction set is presented in Figure 2.5.8 (a), and the schematic of envisioned translation process for ARM emulator is presented in Figure 2.5.8 (b). As seen in the figure the removal of stage 1 of intermediate representation can substantially improve the ARM emulation performance on the cloud side enabling the cloud infrastructure to better utilized for mobile cloud services.

2.6 Conclusion

In this chapter, an overview of computational offloading in MCE is presented, and the most credible computational offloading efforts in MCC are reviewed. We also identified several research problems and challenges that hinder the success are of computational of-

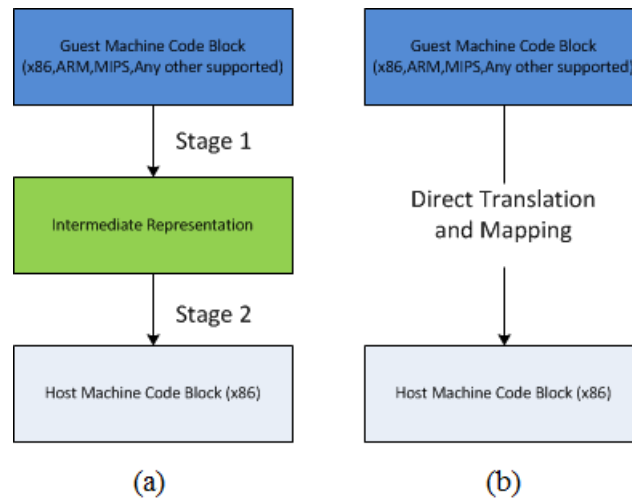


Figure 2.8: (a) Schematics of QEMU translation process from guest to host code blocks (b) Schematics of translation process from guest to host code blocks of envisioned emulator.

floading in MCE. Our analysis of the state-of-art computational offloading system unveils that augmenting computing capabilities of mobile devices are feasible via different types of remote computing resources, including public cloud service providers, one-hop clouds, and proximate resources. However, employing these resources and migrating application to them from mobile devices is not a trivial task and originates several complexities. One of the most significant inefficiencies considering the importance of energy efficiency, application execution time and alleviating the impact of application execution latency is a significant challenging problem of the mobile cloud application that is considered to be addressed in this research.

CHAPTER 3: PROBLEM ANALYSIS

In this chapter, we investigate and analyze based on numerical data the benefit of computational offloading concept regarding application execution time and energy savings. Furthermore, in this chapter, we analyze the existing computational offloading classes based upon the migration mechanisms discussed in Chapter 2. Lastly, by presenting the benefits and limitation of the existing methods based on empirical data we draw the scenario for our proposed research and provide with a preliminary investigation into the possibility and feasibility of this research.

The rest of the chapter is organized into five sections. Section 3.1, provides a detailed investigation on the benefit of computational offloading and the factors affecting the benefit of offloading computation to remote computing devices. Section 3.2, provides empirically experiments of the existing computational offloading mechanisms classified based on the broader concept of migration mechanism. After the discussion on existing methods Section 3.3, provides the details for the motivation towards the idea of process migration based computational offloading mechanism. The feasibility of the process migration based computational is verified using an experimental application in Section 3.4. Lastly, Section 3.5 concludes the chapter.

3.1 Computation offloading benefit analysis

Various benefit studies have been previously focused on whether to offload computation from a resource constraint mobile device to a remote computing device (Wang & Li, 2004; Wolski, Gurus, Krintz, & Nurmi, 2008; Kumar & Lu, 2010) or not. We have utilized a modified formulation of (Kumar & Lu, 2010) for analyzing the benefit analysis of computational offloading in term of energy saving.

Suppose the computation which needs to be offloaded has I instructions. Let S_c and S_m be the speeds, in instructions per second, of the remote computing device (server) and

the mobile device, respectively. The same task thus takes I/S_c seconds on the remote computing device and I/S_m seconds on the mobile system. The mobile device sends α bytes of data to the server and the server in response sends γ bytes of data to the mobile device. The network uplink and downlink bandwidth is β_u and β_d , respectively. The mobile devices take α/β_u seconds to transmit and γ/β_d receive data. The mobile system consumes, in watts, E_c for computing, E_i while being idle, E_t for sending, and E_r for receiving data.

If the mobile device performs the computation, the energy consumption is $E_c \times (I/S_m)$. If the server performs the computation, the energy consumption is:

$$[E_i \times \frac{I}{S_c}] + [E_t \times \frac{\alpha}{\beta_u}] + [E_r \times \frac{\gamma}{\beta_d}] \quad (3.1)$$

The amount of energy saved is:

$$[E_c \times \frac{I}{S_m}] - [E_i \times \frac{I}{S_c}] - [E_t \times \frac{\alpha}{\beta_u}] - [E_r \times \frac{\gamma}{\beta_d}] \quad (3.2)$$

Energy is saved when equation (3.2) produces a positive number. The equation is positive if $\alpha/\beta_u + \gamma/\beta_d$ is sufficiently small compared with I/S_m is sufficiently large. The values of S_m , E_i , E_c , E_t , and E_r are parameters specific to the mobile device.

To empirically analyze the energy saving from the computational offloading according to the basic equation (3.2). The values for the simulation is gathered from (Mahmoodi, Uma, & Subbalakshmi, 2016b), for quick reference also presented in Table 3.1. The average transmission and reception power levels of the mobile device for WiFi service were 257.83 and 123.74mW, respectively. The active and idle power levels of the phone were 644.9 and 22mW, respectively. The average wireless service rates for WiFi obtained using, were 0.80Mbps for the uplink transmission and 1.76Mbps for the downlink trans-

Table 3.1: Simulation Settings

Symbol	Value
E_t	257.83 mW
E_r	123.74 mW
E_c	644.90 mW
E_i	22 mW
β_u	0.80 Mbps (100000 bytes/s)
β_d	1.76 Mbps (220000 bytes/s)

mission, respectively. These uplink and downlink rates are estimated using TCPdump¹ by (Mahmoodi et al., 2016b).

Furthermore, the compute power for a mobile device is considered as 2750 MIPS, and the compute speed for the cloud server is assumed to be 7000 these ratings are assumed based on the BogoMips² calculated from real devices. Now we will present two scenarios to show energy gain and no energy gain of an application task having 7,689,434,795 instructions calculated through Valgrind (Nethercote & Seward, 2007). In the first instance, the application transferred and received 250000 bytes data, while the second scenario transferred 8000000 bytes of received 4000000 bytes of data from the remote server. Now if expand the equation (3.2) and put the value in equation for the both scenarios.

$$\left[644.9 \times \frac{7,689,434,795}{2750 \times 1,000,000}\right] - \left[22 \times \frac{7,689,434,795}{7000 \times 1,000,000}\right] - \left[257.83 \times \frac{250000}{100000}\right] - \left[123.74 \times \frac{250000}{220000}\right] \quad (3.3)$$

$$[644.9 \times 2.79] - [22 \times 1.01] - [257.83 \times 2.5] - [123.74 \times 1.13] \quad (3.4)$$

$$1799.27 - 22.22 - 644.575 - 139.82 = 992 \quad (3.5)$$

¹<http://www.tcpdump.org/>

²<https://en.wikipedia.org/wiki/BogoMips>

$$[644.9 \times \frac{7,689,434,795}{2750 \times 1,000,000}] - [22 \times \frac{7,689,434,795}{7000 \times 1,000,000}] - [257.83 \times \frac{8000000}{100000}] - [123.74 \times \frac{4000000}{220000}] \quad (3.6)$$

$$[644.9 \times 2.79] - [22 \times 1.01] - [257.83 \times 80] - [123.74 \times 18.18] \quad (3.7)$$

$$1799.27 - 22.22 - 20626.4 - 2249.59 = -21098.94 \quad (3.8)$$

The results of offloading decision for scenario 1 is presented in 3.1. It is quite obvious that the mobile device has saved a considerable amount of almost 55% energy by offloading the task. However, the behavior in equation 3.2 is quite different, and the energy saving goes to negative and this due to the fact which we already explained that if $\alpha/\beta_u + \gamma/\beta_d$ is large as compared with I/S_m than the offloading does not give us any benefit regarding energy consumption. Simply we can infer that offloading data-intensive task to remote computing infrastructure is not beneficial.

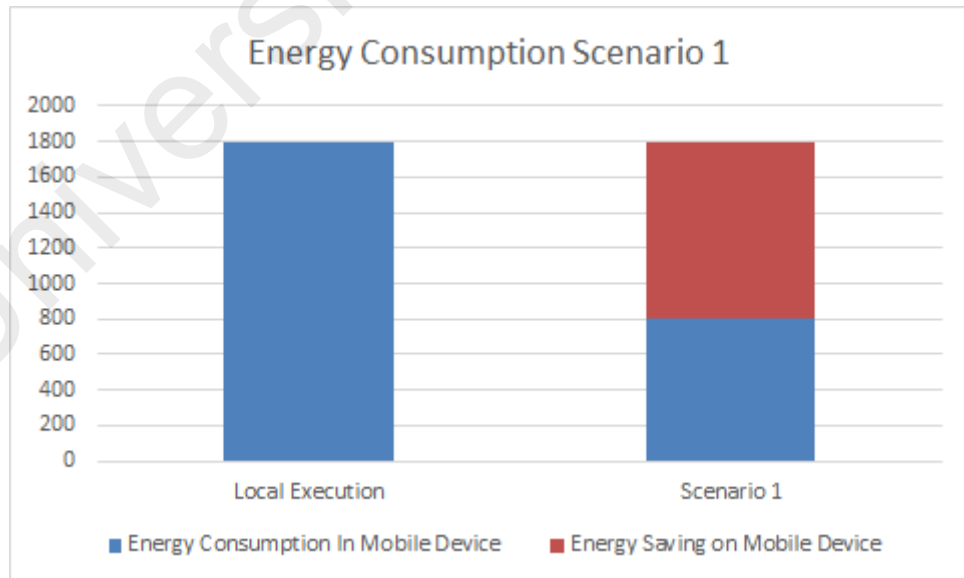


Figure 3.1: Energy consumption on mobile device in scenario 1.

In addition to the above analysis, we simulated two other scenarios. In scenario 1, we have varied the upload and download data transmission sizes and kept the upload

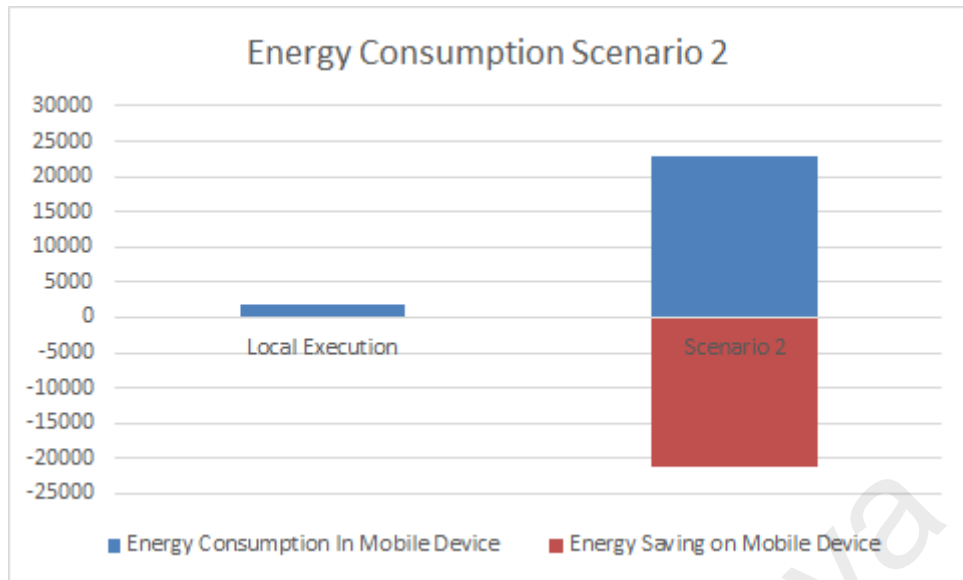


Figure 3.2: Energy consumption on mobile device in scenario 2.

and download bandwidth constant along with other parameters of equation (3.2). The variation range is in between 0...10MB on the interval of 0.5MB for both variables. The result of varying the uplink and downlink data transmission and benefit according to equation (3.2) is presented in Table 3.2 and illustrated graphically in Figure 3.3.

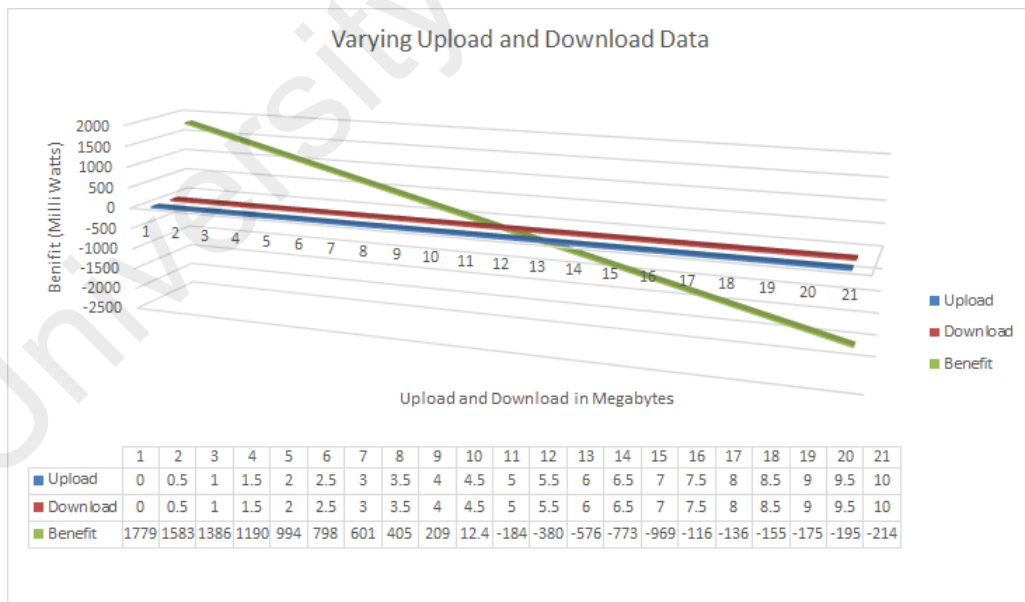


Figure 3.3: Energy benefit indication while varying upload and download transmission data sizes.

According to the data presented in Table 3.2 and in Figure 3.3 it is clear that with the simulation settings we used for our analysis; computational offloading is not beneficial if

Table 3.2: Data Observation while varying Upload and Download Transmission Sizes

Upload	Download	Benefit
0	0	1779.075568
0.5	0.5	1582.778409
1	1	1386.48125
1.5	1.5	1190.184091
2	2	993.8869319
2.5	2.5	797.5897729
3	3	601.2926138
3.5	3.5	404.9954547
4	4	208.6982956
4.5	4.5	12.40113649
5	5	-183.8960226
5.5	5.5	-380.1931817
6	6	-576.4903408
6.5	6.5	-772.7874999
7	7	-969.084659
7.5	7.5	-1165.381818
8	8	-1361.678977
8.5	8.5	-1557.976136
9	9	-1754.273295
9.5	9.5	-1950.570454
10	10	-2146.867614

$\alpha + \beta > 11MB$. A more rigorous analysis is done by the Cartesian product³ of the Upload and Download columns in Table 3.2, and the data is graphically presented in Figure 3.4.

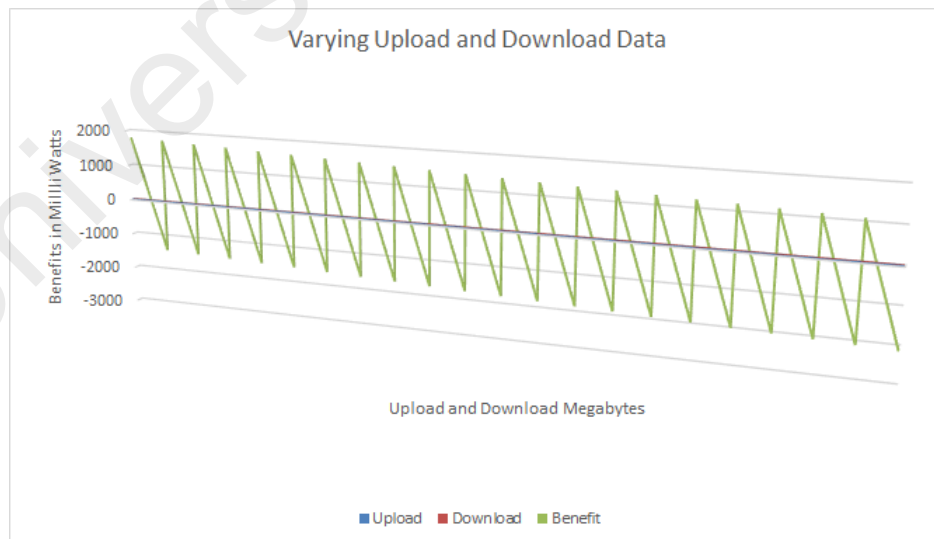


Figure 3.4: Energy benefit indication while varying upload and download transmission data sizes.

Similarly to scenario 1, in scenario 2 we variate the upload and download bandwidth

³https://en.wikipedia.org/wiki/Cartesian_product

Table 3.3: Data Observation while varying Upload and Download Transmission Rates

Upload Rate	Download Rate	Benefit
0.5	0.5	-4326.044432
0.6	0.6	-3308.524432
0.7	0.7	-2581.724432
0.8	0.8	-2036.624432
0.9	0.9	-1612.657765
1	1	-1273.484432
1.1	1.1	-995.9789771
1.2	1.2	-764.7244317
1.3	1.3	-569.0475086
1.4	1.4	-401.3244317
1.5	1.5	-255.9644317
1.6	1.6	-128.7744317
1.7	1.7	-16.5479611
1.8	1.8	83.20890164
1.9	1.9	172.465042
2	2	252.7955683

from 0.5...2MBPS over an interval of 0.1 Mbps and kept $\alpha = 8MB$ and $\beta = 8MB$ so that it is large enough to show the effect of offloading benefit. The data observation from this variation is presented in Table 3.3 and presented graphically in Figure 3.5, the Cartesian product based data observation of upload rate and download rate column (variation range from 0.5...4MBPS) is presented graphically in Figure 3.6.

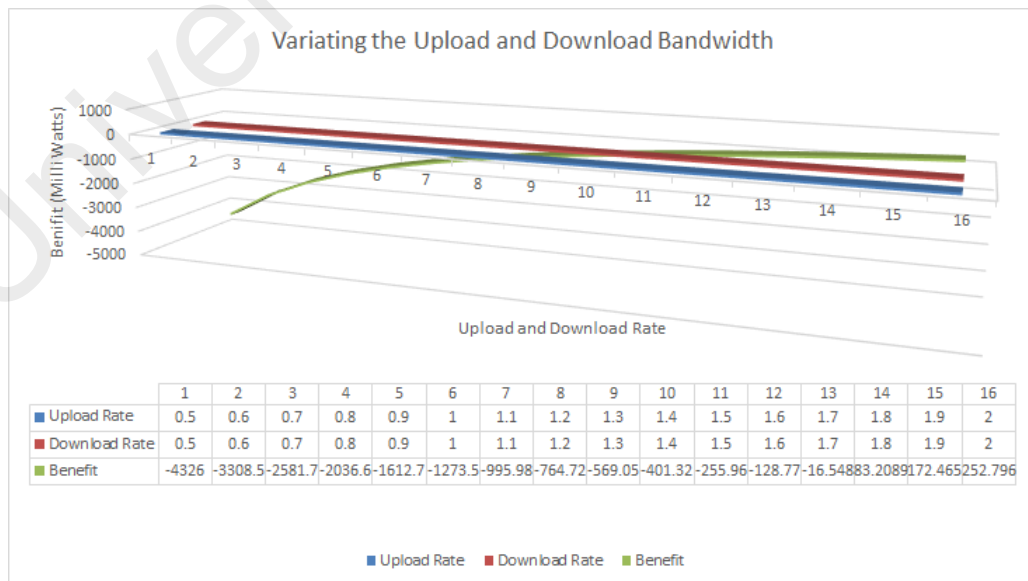


Figure 3.5: Energy benefit indication while varying upload and download transmission data rates.

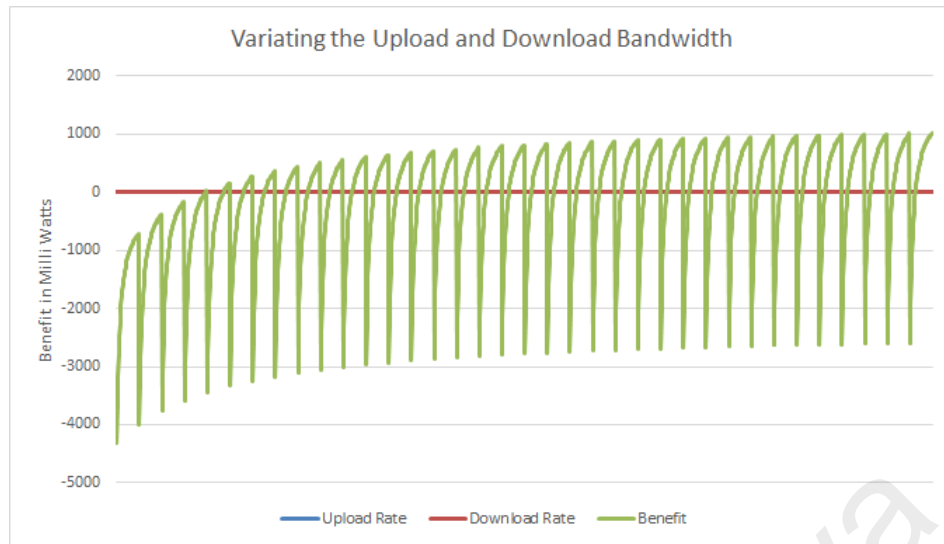


Figure 3.6: Energy benefit indication while varying upload and download transmission data rates.

From the observations presented in the graphs shown in Figure 3.6 and 3.6, we found that for an offloading transaction to be beneficial in which 16 MB is transferred collectively should have the following network bandwidth requirement $\beta_u \geq 1.4MBPS$ if and only if $\beta_d \geq 3.3MBPS$ or $\beta_d \geq 0.8MBPS$ if and only if $\beta_u \geq 3.9MBPS$

3.2 Analyzing the existing computational offloading mechanisms

In this section, we analyze the existing computational offloading mechanisms based upon the broader classification of migration mechanism presented in Section 2.2.1.

3.2.1 VM/Phone clone migration

VM migration based computational offloading solutions uses hardware virtualization technology to synchronize replicas for each connected smartphone on a remote computing infrastructure. As already discussed 2.2.1.1, the VM migration mechanism is being hindered by the amount of data transfer (Hung et al., 2012). An example of the image size (in MB) for a non-live image transfer of Samsung Galaxy SII i9100g with Android Jelly Beans is presented in Figure 3.7.

Additionally, an illustration of the time required to backup and restore these images

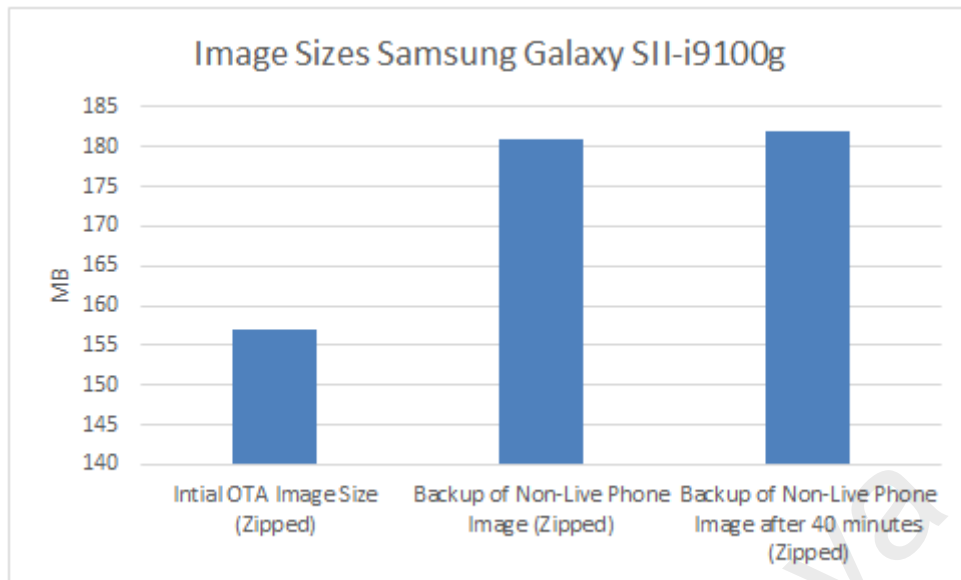


Figure 3.7: Phone image sizes of Samsung Galaxy SII-i9100g.

from and to the phone is illustrated in Figure 3.8.

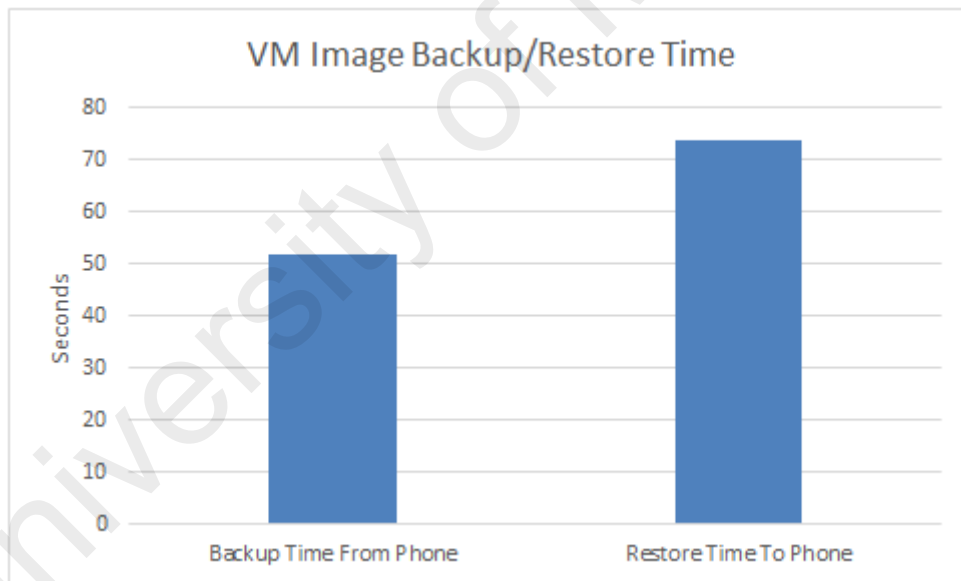


Figure 3.8: Phone image backup and restore time.

Furthermore, with advancement in the Android operating system and the production of new releases with tons of feature further increase the amount of data generated in between successive iterations which subsequently increased the turnaround time. This claim is supported by our another observation of slightly new model smartphone (Xiaomi Mi4i). The observation is presented in Figure 3.9.

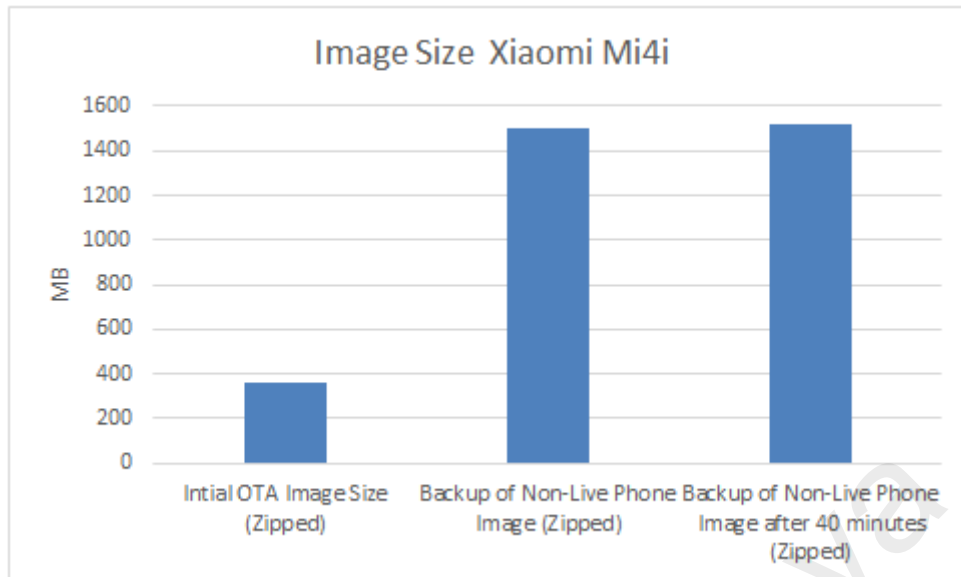


Figure 3.9: Phone image sizes of Xiaomi Mi4i.

To analyze, how such amount of data transfer using different network interfaces impact on the energy depletion of the mobile device, we emulate the scenario using an Android application that performs small data uploads of 10 KB and 100 KB to a remote computing infrastructure using either 3G or WiFi one after another. While the application is executed, the battery status is observed using PowerTutor (Kalic, Bojic, & Kusek, 2012). The experiment employs Samsung Galaxy SII i9100g smartphone. While the remote server uses a network emulator (Hemminger, 2005) that adds a controlled amount of queuing delay to the network interfaces between the smartphone and the server. Similarly like (Cuervo et al., 2010), we then evaluated two scenarios: (1) the smartphone using WiFi to reach the server (adding 25 ms or 50 ms of queuing delay); and (2) the smartphone using 3G with a measured RTT of 200 ms. Figure 3.10 shows the energy consumed by the smartphone during the two uploads. Using 3G, the smartphone consumes almost 2.5 times as much energy as it does while using WiFi with a 50 ms RTT, and nearly five times the energy of WiFi with a 25 ms RTT. By analyzing the drastic discrepancies of energy consumption shown in Figure 3.10 for such small data uploads, one can estimate how drastic would be to perform VM Migration based computational offloading in term

of energy consumption.

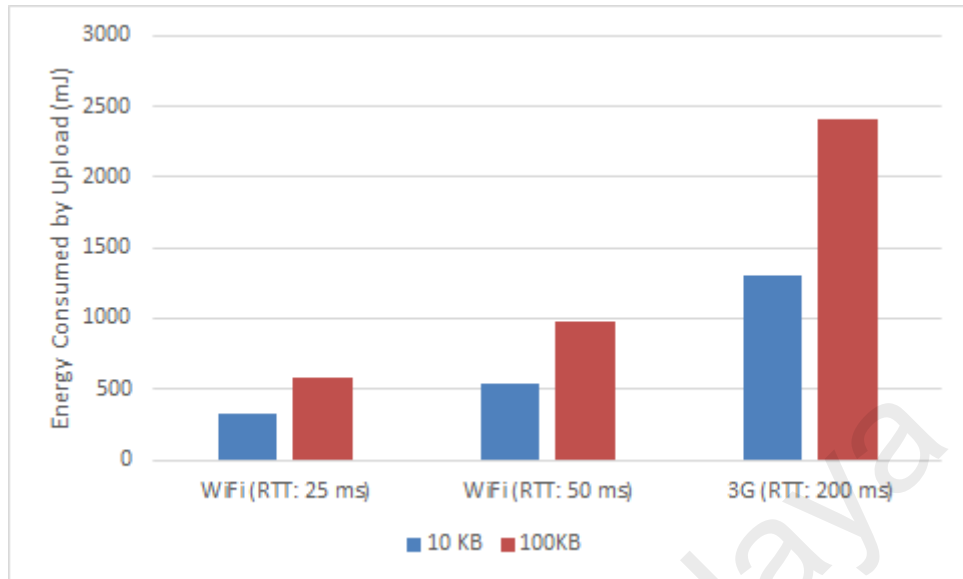


Figure 3.10: The energy consumption of WiFi connectivity vs. 3G connectivity.

Along with the insight presented in Figure 3.10 and according to our analysis in section 3.1, computational offloading could not be beneficial if a significant amount of data is being sent or received as compared with the network metrics. To mitigate this behavior, replay mechanisms (Hung et al., 2012; Flinn & Mao, 2011; Surie, Lagar-Cavilla, de Lara, & Satyanarayanan, 2008; Gomez, Neamtiu, Azim, & Millstein, 2013), in which execution of mobile device instructions are captured as a trace and then executed on remote VMs and vice versa, can be used while synchronizing remote VMs with phone state. However, replay mechanisms generate large trace files and require extensive modifications to the Virtual Machine Monitors (VMM) and phone kernel and remote computing device kernel. From a practical point of view, the phone clones/VMs needs the same hardware platform as the server-side to retain a working synchronized image.

3.2.2 Code migration and delegation

Code Migration mechanisms exploit the platform independence feature of ALVM (such as JVM, .Net Runtime Environment and DVM). Code migration uses source code annotation, reflection, and dynamic class loading features of ALVMs to enable execution

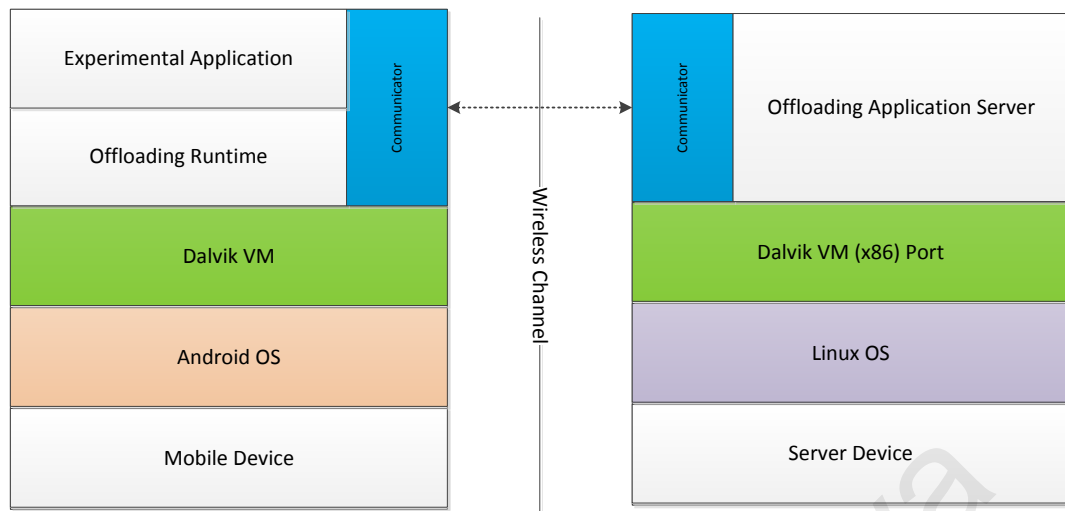


Figure 3.11: Code migration framework for Android devices.

on of run-time migrated platform independent intermediate code. To investigate the code migration based computational offloading mechanism, we have developed a conceptual framework which is presented in Figure 3.11.

The offloading run-time on the mobile device presented in Figure 3.11 utilizes the reflection features to identify potentially annotated methods for offloading. Whereas, the offloading application server on the server side may utilize reflection or dynamic class loading depending upon the offloading request parameters. The service method of the offloading application server is listed in Listing 3.1.

Listing 3.1: The service function of our experimental code offloading mechanism

```
public void service(OffloadingRequest request, OffloadingResponse response) throws Exception {  
    String name = request.getMethodName();  
    Class[] paramTypes = request.getParameters();  
    Object[] paramValues = request.getParametersValues();  
    //Get the class  
    Class classN = request.getClassObject().getClass();  
    //Construct class at server side  
    java.lang.reflect.Constructor constructor = classN.getConstructor();  
    Object instance = constructor.newInstance();  
    //Get the offloading method  
    java.lang.reflect.Method runMethod = classN.getDeclaredMethod(name, paramTypes);  
    runMethod.setAccessible(true);  
    //invoke the remote method with input parameters  
    Object result = runMethod.invoke(classN, paramValues);  
    response.write(result);  
}
```

To analyze the impact of application performance in term of execution time and energy, we have developed an experimental matrix multiplication application. The criminality of matrices used in the experimental application is 1000×1000 . The network topology of the experimental setup is presented in Figure 3.12, in which the mobile device (Samsung Galaxy i9100g) connect to the offloading server over a single hop (access point). The energy consumption values are gathered used PowerTutor.

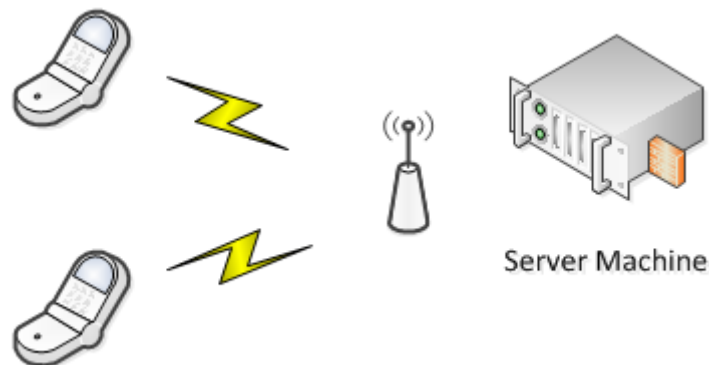


Figure 3.12: Network topology of code migration experimental setup.

Figure 3.13 and 3.14 provides the result of an average of three runs of an experiment

performed using the developed framework for android devices.

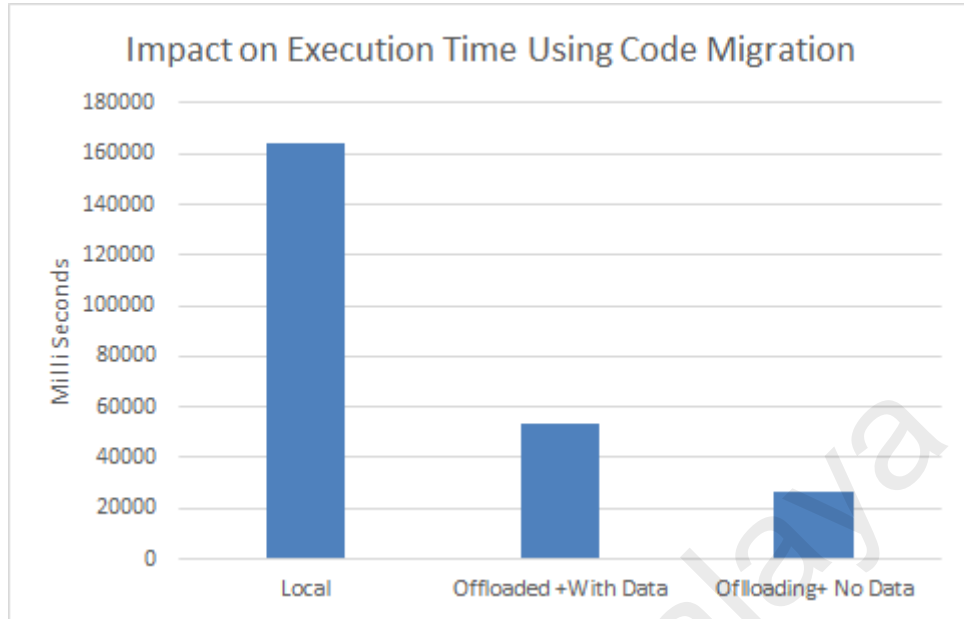


Figure 3.13: Impact on the execution time of experimental application using code migration.

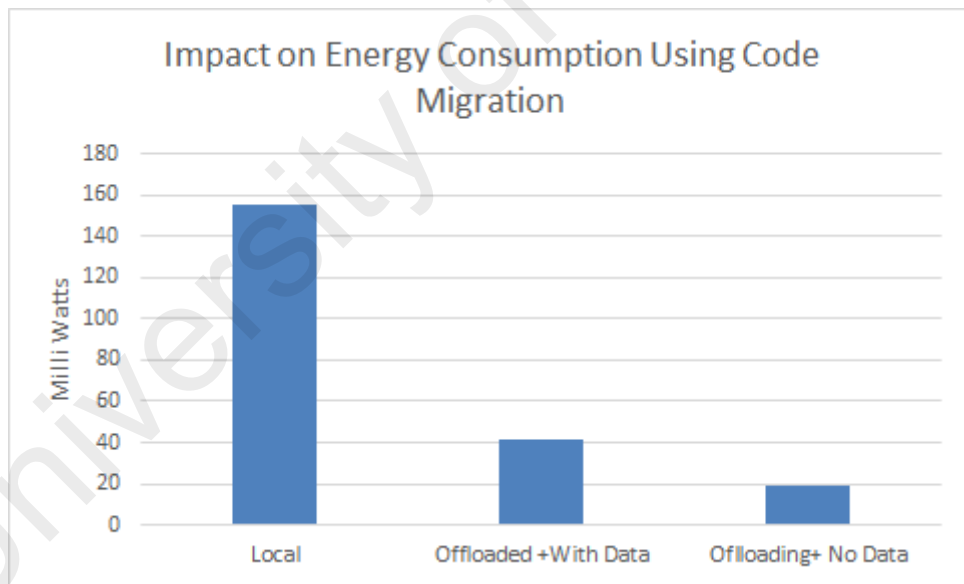


Figure 3.14: Impact on the energy consumption of experimental application using code migration.

The results presented in 3.13 and 3.14 are showing significant improvement in application execution time and energy consumption. Now the question arises why we are introducing a new computational offloading method while code migration provides exquisite results. The details answer is explained in Section 3.3, but the brief answer is the strict

dependency of code migration on ALVMs and its platform independence features which now even Google dropped the Android DVM in favor of ART.

3.2.3 Thread state migration

Similarly like Code Migration (discussed in 3.2.2), Thread synchronization based mechanisms also exploits the platform independence feature of ALVM (such as DVM, JVM, and .Net Runtime Environment). However, instead of migrating code components thread migration primitive migrate a complete thread from one machine to another. Furthermore, thread state synchronization mechanisms can exploit the parallel compute power available on the remote computing infrastructure thriving up to the magnitude of performance improvement. To experimentally evaluate the thread synchronization primitive we once again utilized the same experimental setup as presented in Figure 3.12. The thread state migration mechanism utilized for experimentation is COMET (Gordon et al., 2012) which uses DSM to synchronize threads memory across DVM. In order to analyse the behavior of COMET, we have selected some standard benchmarks (such as Linpack, and SciMark). However, COMET seems is not beneficial in every type application. This phenomenon is presented in Figure using experimentation of the popular Linpack Benchmark developed by two different developers (rs.pedjapps.Linpack.apk⁴ and roylongbottom.LinpackJava.apk⁵).

Moreover as discussed in section 2.2.1.3, the thread state migration mechanism requires extensive modification of ALVMs to synchronize computation between the mobile device and the remote server. Here, we analyse the effect of the modification of ALVMs on applications running locally on the mobile device which either cannot be executed remotely, or cannot be offloaded, or the server is disconnected. In order to analyze this

⁴<https://play.google.com/store/apps/details?id=rs.pedjaapps.Linpack>

⁵<http://www.roylongbottom.org.uk/android%20benchmarks32.htm>

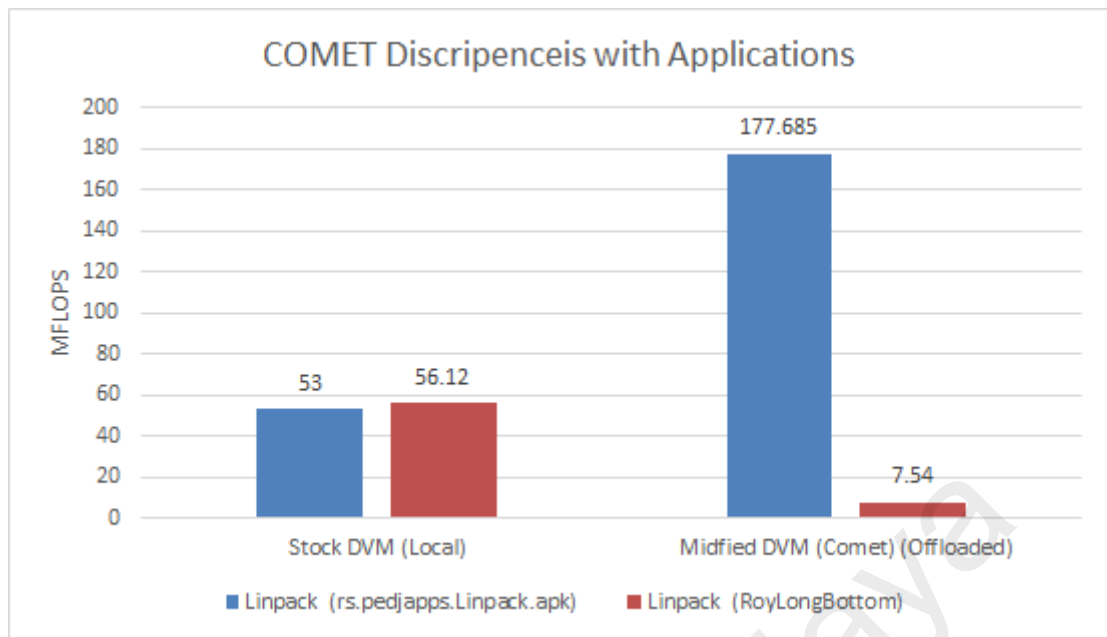


Figure 3.15: COMET discrepancies with same applications from different developers.

behavior we gather two similar devices (Samsung Galaxy SII i9100g) with the same specifications in terms of hardware and software configurations to show its impact. One of the phones has a modified Dalvik (DVM) from COMET (Gordon et al., 2012), whereas the other has the original stock Android DVM. The benchmark results in Figure 3.16 and 3.15 measures the mega floating-point operations per second (MFLOPs), which clearly demonstrate the effect on performance caused by the modification of DVM. The modification adds some extra steps in original lifecycle of the ALVMs which brings in extra energy consumption and also shares the compute capacity which reduces the compute power.

Lastly, when we have analyzed the network performance of the COMET we found a large number of packets are transferred between connected end points. The analysis is done using Wireshark ⁶, and it was done overnight, while the phone was running no applications except the defaults. The analysis was started on 8:15 PM 4th September 2015 and was ended on 12:15 PM 5th September 2015. The overnight analysis reveals that almost 17 million (17473214) packets are exchanged between the two connected end point

⁶<https://www.wireshark.org/>

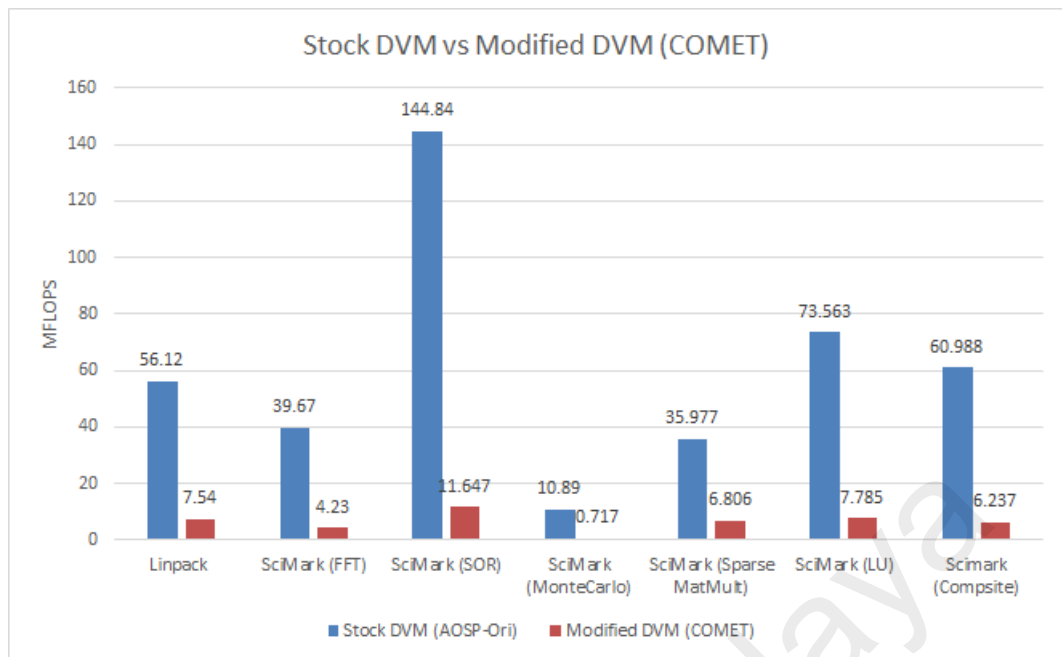


Figure 3.16: Performance comparison of modified DVM vs. stock DVM using Linpack benchmarks (in Mega FLOPS).

(i.e. mobile device and a local server). Each packet size was investigated and was found around 90 bytes which in total leads to approximately 1.572 GB. Hence, in the case where the user is on a cellular data plan COMET could be disastrously expensive. In addition to this, it would drain the battery out rigorously without any substantial benefit. Conclusively, thread migration also suffer from strict dependency on ALVMs and which mobile device OS vendors are dropping for performance issues (detailed in coming Section 3.3).

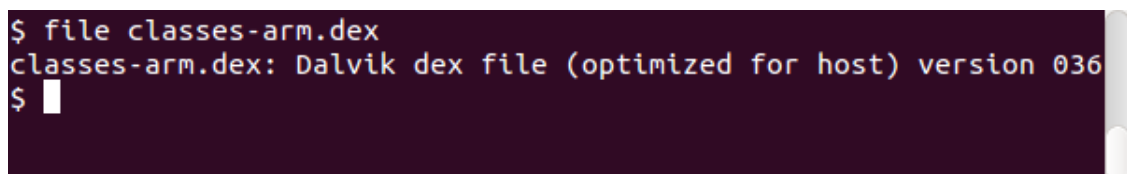
3.3 Why process migration based computational offloading?

These code migration and thread-state migration mechanisms besides their benefits become automatically invalid in environments where the application execution runtime is based upon platform specific native binary machine code instead of platform independent intermediate code. Apple iOS was already using such environment to provide better application performance and lower energy depletion. From the Android Open Source Project (AOSP) point of view, Google introduces ART environment (Google, 2013), in place of the Dalvik runtime environment. ART features AHOT (ahead of Time) compila-

tion to device-specific native binaries; this feature expedites the application execution by reducing the overhead energy consumption and application execution time caused mainly by the repetitive just-in-time (JIT) compilation in DVM (Buckley, 2013).

Now as already discussed in Chapter 2 and Section 3.2, current computational offloading based solution are based on primitives such as VM/phone clone migration, or migration of intermediate language codes executing inside an ALVM (such as DVM), or ALVM thread synchronization or the classical client/server SOA. Each of these augmentation techniques is deficient in its own respective ways such as communication overhead caused due to transmission of large VM image sizes, dependency on ALVMs and loss of execution in case of disconnection. Apart from this, the existing computational offloading solution for android based smartphones are strictly dependent on the Dalvik VM. However, the introduced Android Runtime (ART) environment featuring ahead-of-time (AHOT) compilation of mobile applications into native machine-dependent binaries upon installation; and these state-of-the-art offloading solutions do not consider the native code of ART-based mobile applications.

To verify the native code behavior of ART, we gathered the DEX files of an installed application from two Android devices. One device is running Dalvik, whereas the other is running ART. The gathered files are checked with the `file`⁷ tool available in most Linux distributions (Figures 3.17 and 3.18). The DEX files gathered from Dalvik- and ART-based phones are in bytecode format and Executable and Linking Format (ELF), respectively.



```
$ file classes-arm.dex
classes-arm.dex: Dalvik dex file (optimized for host) version 036
$
```

Figure 3.17: File type of Dalvik compiled DEX file.

⁷<http://linux.die.net/man/1/file>

```
$ file classes-arm64.dex
classes-arm64.dex: ELF 32-bit LSB shared object, ARM aarch64, version 1
(GNU/Linux), dynamically linked, stripped
$
```

Figure 3.18: File type of ART compiled DEX file.

The ELF file is platform dependent (Committee et al., 2001). For instance, the ELF file compiled for an ARM platform cannot be executed on an x86-based platform. Most current state-of-the-art computational offloading mechanisms are exploiting Dalvik bytecode, which is platform independent. The same Dalvik bytecode can be executed without any modification on any platform where DVM is running.

Furthermore recently, 1,208,476 mobile applications from the Google Play Store have been statistically analyzed to investigate the number of mobile applications utilizing native libraries (Afonso et al., 2016). The authors reported, a total of 446,562 mobile applications (37.0%) used at least one natively compiled library. Considering the result of this recent study which also emphasizes the growth of applications toward native codes, computational offloading mechanisms should be reevaluated to overcome the issue of native code. Therefore, we are interested in a process migration-based computational offloading mechanism utilizing checkpoint/restore method.

3.4 Feasibility analysis of process migration based computational offloading

In order to analyze the process migration based computational offloading system, we exploit the concept of context switching in an operating system process. The process state of any computer program, as well as the ART-generated ELF file, is architecture dependent. A process state from an ARM-based machine cannot be used to restart the process on an x86-based machine. This condition is due to the difference in assembly, hardware components, instruction sizes, application binary interface, and other related factors. In most cases (almost 90%–95%), mobile devices use an ARM-based machine (Forbes, 2013;

Bent, 2012), whereas the physical or virtual machines in the cloud are normally based on x86 architectures. Thus, we have three options for the checkpoint/restore or process migration-based computational offloading for native and ART-based mobile applications. First, we can use a certain type of manual transformation of a process state from one architecture to another (Chanchio & Sun, 2002). We can disregard this option because it requires extensive modification of the compiler or source code for both endpoints. Second, we can emulate the ARM instruction set in the cloud using binary translators, such as QEMU (Bellard, 2005), which is not feasible for deadline and performance sensitive applications. This hypothesis is being verified by a simple experiment (presented in Fig. 3.19) where we executed several times a 1000x1000 matrix multiplication program (natively compiled) on a real device (Samsung Galaxy SII) and the QEMU emulator. The difference of performance cannot compensate the data transfer time and provide no performance improvement to mobile applications, as emulator are developed for behavioral testing, not performance sensitive fulfillment (Yousafzai, Gani, et al., 2016). Further, in the case of high-end mobile devices, the hypothesis would be even stronger. That's why we outline the emulator performance problem as a research challenge in section 2.5.8.

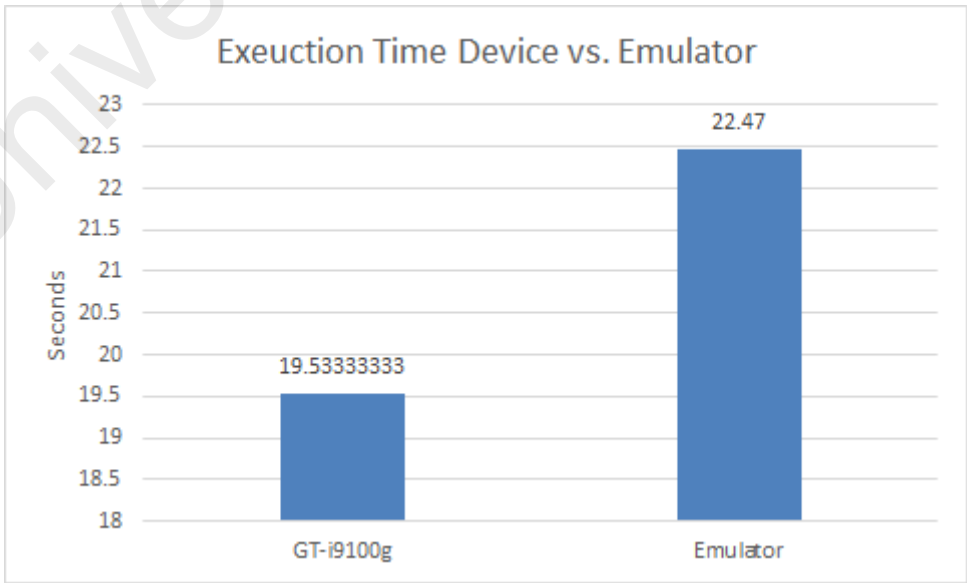


Figure 3.19: Comparison of emulator performance with real device.

Finally, the third case we are left with is to utilize a compatible infrastructure (e.g., ARM) on the server. The availability of ARM infrastructure in the remote cloud or local cloudlet is crucial to actually envision the MCC ecosystem and enable phone clones and VMs in the remote servers.

For the sake of analysis, we assume that the network conditions are stable, that no disconnection occurs during an offloading transaction, and that an offloading transaction is atomic. These assumptions are made to retain the focus on analyzing migration mechanism rather than network analysis and disconnection management, although these options should be explored in the future.

For analyzing the process migration based computational offloading, we have modified the original Android kernel to enable the insertion of custom kernel modules. With this additional kernel module, which exploits the concept of context switching. Primarily, the module is configured to connect to the remote offloading server via an associated user space communicator, which also resides and is running on the mobile device.

For the feasibility analysis, we deployed an ARM infrastructure box (Compulab Utilite⁸) at the server. We also deployed an x86 machine to facilitate and administer the ARM Box. The topology of the experimental setup is presented in Figure 3.20

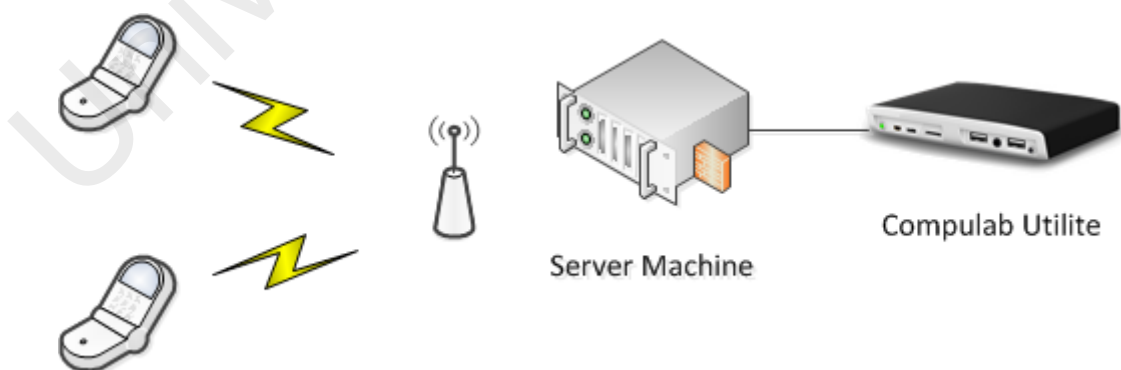


Figure 3.20: Experimental setup.

The server PC configuration comprises Ubuntu 14.04, 4 GB Ram, and Core i7 CPU

⁸<http://www.compulab.co.il/utilite-computer/web/utilite-overview>

3.4 GHz. A Samsung GT-I9100G smartphone is used as the client device. The experimental application is a 1000×1000 matrix multiplication program written in C with no migration or annotations and interfaces. The experimental application is compiled with the standard Android toolchain downloaded along with the AOSP/CyanogenMod source code.

3.4.1 Experiment details

To migrate an application state from the client device (Samsung GT-I9100G) to the ARM box, both devices are flushed with the custom kernel to enable the insertion of kernel modules. Both devices (client device and ARM box) execute a user-space program to communicate with each other and transfer the process state over the socket. For the proof-of-concept experiment, the experimental applications are installed and executed on both devices. Once the communicators residing on the devices set up the link with each other, the modules on both sides detect the setup and start synchronizing the process state. However, the kernel modules periodically synchronize the applications executing on the other device because of the lack of synchronization markers in the experimental application. Thus, the migration control is unsuitable. We have performed ten runs of the experiment and then used the average results for presentations. Our first parameter of interest is the improvement in the execution time of the offloaded matrix multiplication application. Figure 3.21 presents the results of this parameter.

The second significant parameter is the number of bytes transferred between successive synchronizations of application states from the client device to the server ARM box. The application declares and stores three 1000×1000 matrices (two are operands, one is the resultant). Thus, the memory acquired by the process for these three matrices are 12 MB ($1,000,000 \times 3 \times 4bytes$). The bookkeeping of process state supplements an additional 200 KB to the process snapshot size.

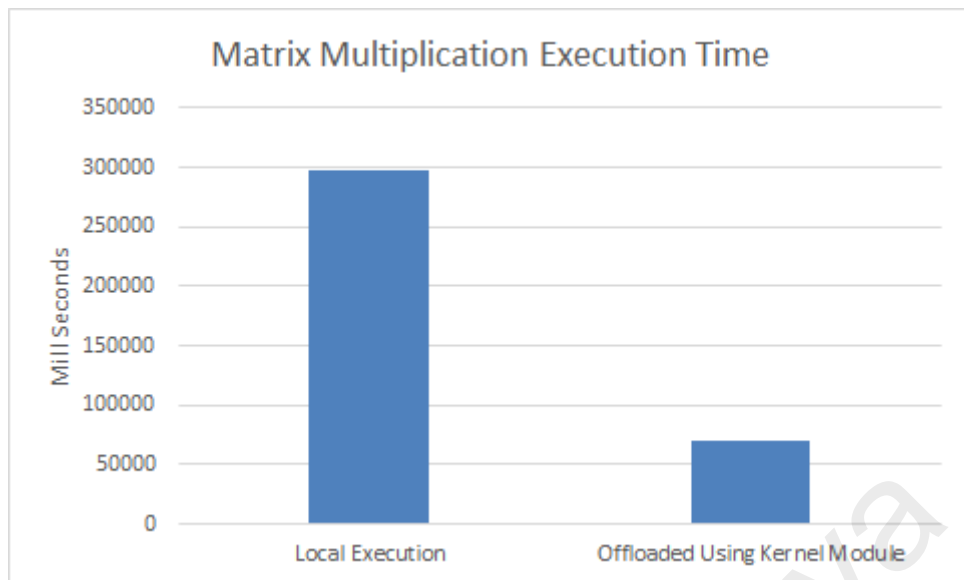


Figure 3.21: Execution time impact (in milliseconds).

The experimental analysis, reveals that using kernel module is not suitable for the migration for a number of reasons. They include code complexity, the difference in kernel version between devices, disabled kernel modules by default, needs a rooted and custom ROM on the phone to exploit this method, and the user curiosity on the security and privacy exploitation, which can be made in the kernel space. Above all, the kernel module based process migration mechanism is hard to port to other applications. Lastly, the analysis also reveals that the application binaries are required on both ends, and applications on both should be in running state and synchronized on an interval which effectively leads to miss utilization of computational resources.

3.5 Conclusion

In this chapter, we empirically analyze the benefits of computational offloading and the factors affecting the decision of computational offloading algorithms. Using empirical analysis of varying the offloading data transmission sizes and bandwidth data rates, we identified the benefits range according to some static settings of energy consumption and computational powers of the devices involved. Our finding complement the work by other researchers in this domain (Kumar & Lu, 2010) who emphasize on the trade-off between

computation and communication before offloading.

In addition to this benefit analysis, we empirically evaluate the current computational offloading solutions using a broader classification based upon migration mechanism. Using the empirical analysis and the research gap we provide a feasibility experiment on the checkpoint restore based computational offloading mechanism using a matrix multiplication program. In this study, we found that performance gain of utilizing remote resource is highly influenced by the number of synchronizations along with the amount of data transmission between mobile and remote computing devices.

Our investigation advocate that kernel module based process migration computational is not suitable for the migration in MCE. The reason behind unsuitability includes code complexity, security, privacy, and kernel version mismatch between devices and modification of kernel because that most of the mobile device are stuck with kernel version 3.4.x to 3.10.x. Concluding, the kernel module based process migration mechanism in mobile devices in current is difficult to port to support all applications. Finally, the analysis also reveals that the application binaries are required on both ends, and both should be in running state and synchronized on an interval which effectively leads to miss utilization of computational and network resources.

CHAPTER 4: LIGHTWEIGHT COMPUTATIONAL OFFLOADING FRAMEWORK FOR MOBILE DEVICE AUGMENTATION

This chapter presents the framework proposed in this research. With the help of few schematic diagrams, we present the main building blocks and components of the proposed framework and describe their functionality. Moreover, the interaction among major elements of the framework is illustrated using flow diagrams and described in detail. The design of data to evaluate the performance of our work is also presented, to analyze and synthesize the finding.

The remainder of this chapter is as follows. Section 4.1 presents the comprehensive description of the proposed framework. Section 4.2 describes the major building blocks and algorithms in the proposed framework. Section 4.3 identifies data to be generated from the framework and explains how to generate required data. The chapter is concluded in section 4.4.

4.1 A Lightweight process migration based computational offloading framework for mobile device augmentation

We proposed a lightweight PMCO framework that is capable of augmenting resource-constraint mobile devices to improve native applications performance and energy consumption. As already discussed in Section 1.1.5, the basic idea behind process migration is of checkpointing which in turns stems out from the process context switching in operating systems. As obvious from the name, process checkpoint/restart has two phases. The first phase is to save the running state of a process. This usually includes register set, address space, allocated resources, and other related process private data. The second phase is to re-construct the original running process from the saved image and resume the execution from exactly the interrupted point.

Designing our framework based on process migration based computational offloading framework for mobile device augmentations has the following benefits:

- Generalized: Process migration based upon checkpoint/restart provide the ability to handle both migration-aware and non migration-aware applications unlike the many migration mechanism discussed in literature review in Chapter 2, which only consider offloading migration-aware applications. Migration-aware applications have been coded to explicitly take advantage of process migration. Dynamic process migration can automatically migrate these applications to save mobile device energy and improve its computational power. While in non migration-aware applications, the application developers does not design them to take benefits of offloading.
- Ability offloading application without needing application binary on the server side similarly, like COMET (Gordon et al., 2012). The checkpoint/restore methods in user space package the application binary and state into one package which can be transferred to the remote device and executed directly. Eliminating the requirement of application binaries availability, in the remote computing platform.
- Portability: Process migration can be performed almost on every type of computing infrastructure with minimal modification to the operating system kernel. So the proposed computational offloading system can be ported to any of the supported hardware and software platforms which has the checkpoint features.
- Loose Coupling: Process migration based computational is loosely coupled with the underlying mobile device environment, unlike the state-of-the art which depends on ALVMs or modification of these ALVMs.
- Saving Computations: Unlike any other computational offloading mechanism the checkpoint/restart in mobile devices can also help mobile applications to checkpointed in critical battery conditions and then later restarted either locally or remotely.

- **Resource localization:** Mobile applications which are distant from the device which has the data that the application is using tend to spend most of their time in performing communication between the mobile device and remote computing device for the sake of accessing the data. The process migration framework can be used to migrate the process closer to the data that it is processing, thereby ensuring it spends most of its time doing useful work.
- **User Preference:** Our framework, is based upon preference based system defined by the user to control the behavior of the computational offloading system. Further, most of the existing offloading mechanism uses HTTP-based communication while we focused on using single TCP socket to get some improvement.

Additionally, our framework based on Checkpoint/Restore in user space has a shortcoming also, but it applies to any process migration based mechanism and also applies to VM Migration based computational offloading:

- **Platform Dependent:** Our proposed checkpoint restore based computational mechanism is platform dependent meaning that if the mobile device hardware platform is ARM, then one needs an ARM hardware platform on the server side. This shortcoming also applies to VM Migration based computational offloading systems.

4.1.1 System requirements

To support process migration based computational offloading effectively in MCE, a mobile device should be able to provide the following functionality:

- **Import/Export of Process soft state:** The mobile device must provide some import/-export interfaces that allow the process migration mechanism to extract a process's soft state and import this state on the destination remote computing device. These interfaces should be provided by the underlying operating system, supported by the

programming language, or other elements of the programming environment of the process. Process soft state includes processor registers, process address space and communication state, such as open message channels in the case of message based systems, or open files and signal masks.

- Naming/accessing the process and its resources: After migration, the migrated process must be accessible by the same name and mechanisms as it was before migration as though migration never occurred. The same applies to its resources, namely open files, threads, etc.

Our proposed framework consists of five major building blocks, namely Migration Preference Manager, Migration Manager, Migration Coordinator and Admission Control illustrated in Figure 4.1. the high-level components of the proposed framework.

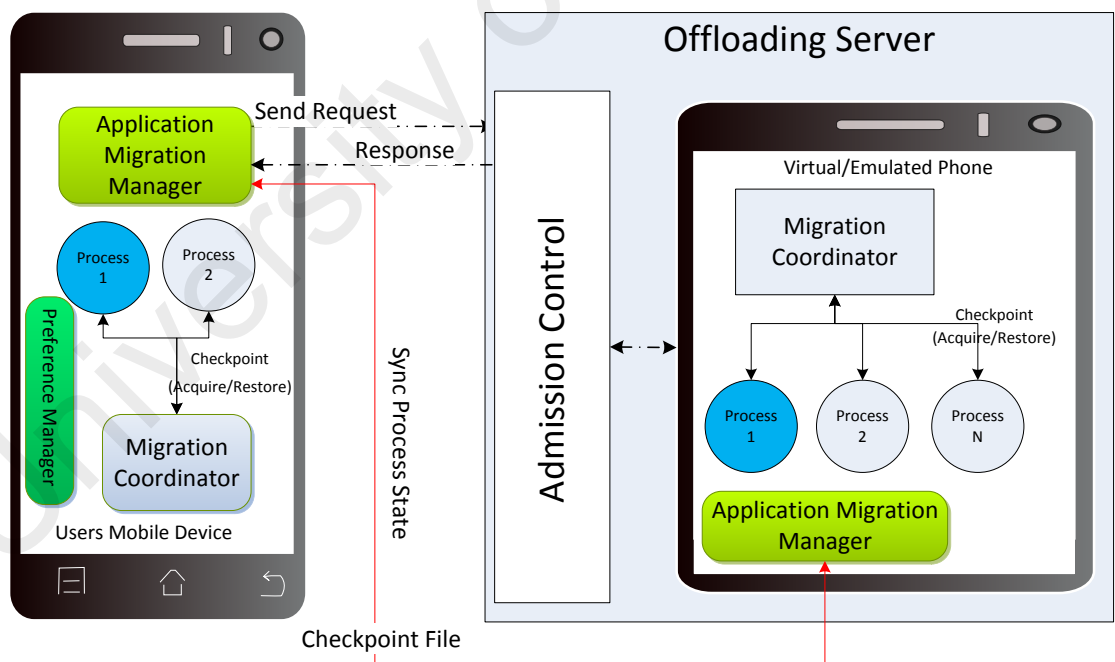


Figure 4.1: Bird eye view of proposed PMCO for mobile device augmentation.

An abstract description of the system and its operational tasks are briefly presented in Figure 4.2 and described as follows, and the in-depth description is presented in section

4.2.

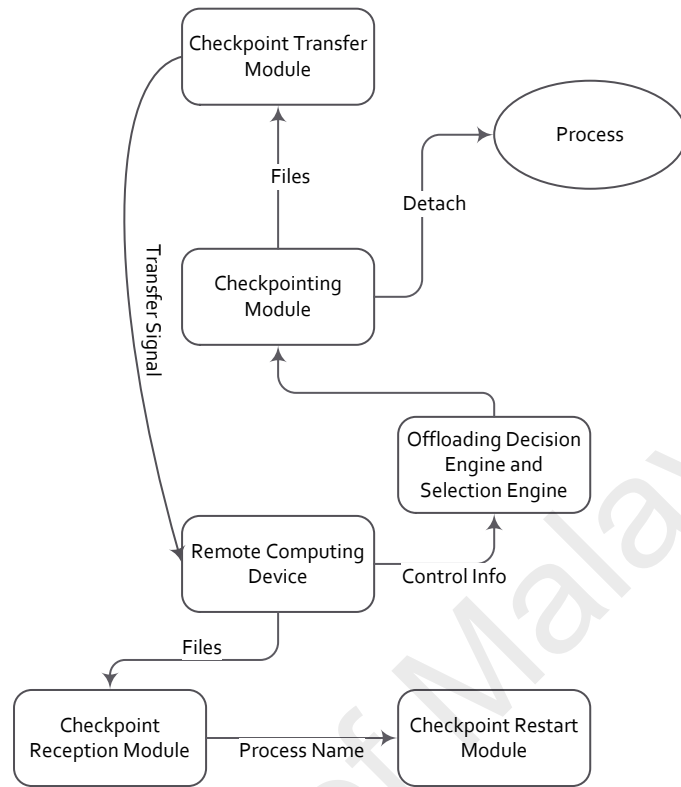


Figure 4.2: Interactive flow of the proposed PMCO based mobile device augmentation framework.

Initially, on the mobile side, the user defines migration preferences of installed application by defining application absolute path along with a binary migration indicator. After setting up the application migration preferences, application migration manager should be started which is responsible to select, and communicate with offloading service providers and also coordinate with application migration coordinator. Afterward, the migration manager will initiate the application migration coordinator (a checkpoint coordinator) which is used to assist the checkpointing process. The migration coordinator receives commands from application migration manager to checkpoint an application which is not migration-aware. The applications which are migration-aware checkpoint itself and signal the application migration manager after checkpointing itself. Once the application is checkpointed the migration manager will transfer the checkpoint file to the

remote computing infrastructure.

On the server side, there are three modules the first one is the Admission control. Admission control modules enforce the fair-usage policy of server resources. Application migration manager on the server receives the checkpoint file and restart the process on the server from the checkpoint file. The application migration manager will wait for the completion of execution of the restarted process. In the case of non migration-aware applications the server side application migration coordinator will periodically checkpoint the restarted applications. For the case of migration-aware application, it will automatically checkpoint once the re-checkpoint marker is reached. Once the application execution is complete, the application migration manager will fetch the checkpoint file and transfer it to the mobile device where it can be restarted to finish the final process exit point.

4.1.2 Process execution

Application binaries are required to started under the checkpoint launcher, causing them to connect to the application migration coordinator upon startup. As a results threads are spawned, child processes are forked, libraries are dynamically loaded, the checkpointing mechanism will automatically and transparently tracks accordingly. Figure 4.3, shows the relationship between the application migration coordinator and the process which is executed using the checkpoint launcher.

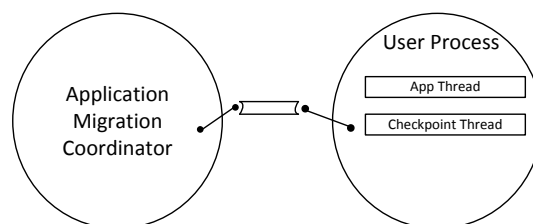


Figure 4.3: Application execution in our proposed framework.

The checkpoint launcher will preload the checkpoint library as a checkpoint thread before calling the main function of the original process which needs to be migrated. The checkpoint thread is connected to the application coordinator using a local socket.

At any given moment either the application threads are active, or the checkpoint thread is active at the process signal handler. Due to the checkpoint thread, the process is being able to be checkpointed for both migration-aware applications (using self-checkpointing) and non migration-aware applications using a signal from the application migration coordinator to the checkpoint thread in the process to checkpoint itself and respond, this behavior is presented in Figure 4.4.

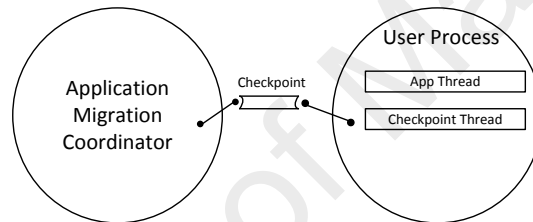


Figure 4.4: Checkpoint message from coordinator to the user process.

4.2 Proposed Components in PMCO framework

4.2.1 Migration Preference Manager (MPM)

Migration Preference Manager is a conceptual user space non-migratable registry that will allow the user to manage (enable and disable) the migration preference for all the user space installed native mobile applications on the mobile device. This preference manager is added keeping in view of deficiency of existing computational frameworks which are being burdened by the static analysis (using at least one pre-execution) of application which is counter productive in the offloading process. The preferences are stored in a key/value based fashion in the registry. The preferences defined in this registry are of two types global preferences and application specific preferences. Global preferences

define the preferences for parameters which are device specific and can apply to all the mobile applications this includes $E_c, E_i, E_t, E_r, S_m, B_t$ and server selection criteria such as cost per service. Where E_c stands for active energy consumption of the device CPU, E_i stands for idle energy consumption of device CPU, E_t and E_r correspond to the energy consumption while the wireless interface is in transmission and reception mode. S_m represent the compute power of the device in MIPS, and B_t is the benefit threshold. The B_t value indicates the amount of energy saving of the computational offloading process that is required by offloading decision function (Equation (4.1)) to trigger the offloading process.

$$\left[\left[E_c \times \frac{I}{S_m} \right] - \left[E_i \times \frac{I}{S_c} \right] - \left[E_t \times \frac{\alpha}{\beta_u} \right] - \left[E_r \times \frac{\gamma}{\beta_d} \right] \right] > B_t, \text{ where } B_t \geq 0 \quad (4.1)$$

On the other hand, application specific parameters preference value will be a 4-tuple $\langle I, \alpha, \gamma, P_f \rangle$. Where I is the number of instruction in the application, α represent the amount of data which will be at least transferred in a single offloading transaction; this is the only value which the offloading engine can get accurately with minimal compute overhead as it can calculated at the time of offloading. γ is the amount of data which will be received in a single offloading transaction. Intuitively, this cannot be known at the time of offloading decision without prediction, estimation or pre-execution, which becomes expensive and counter-productive as it increases the offloading decision time. Due to these reasons, the preference manager will simplify the offloading process with user assistance. P_f is a binary variable which can set to enable forced migration or disable migration for a specific application.

4.2.2 Application Migration Manager (AMM)

Application migration manager is responsible for discovering, selecting and provisioning the communication channel to OSP (they could be either ad-hoc mobile nodes, cloudlets

or CSP depending upon the MCC deployment model). An OSP is selected based upon the preferences (cost and QoS metrics) defined by the mobile user and the offloading decision function. Once an OSP is selected, and a connection is successful it will initiate the offloading workflow on the mobile device. The first step would be to start/initiate the application migration coordinator. The second step would be to iterate over MPM application list and if there is an application which needs to be executed in an offloadable fashion such that the offloading decision equation also results positive, it will launch the application using the application checkpointing mechanism (ACM). AMM will wait for a signal from ACC that a checkpoint package is ready to be shipped to a remote computing device on the OSP's infrastructure. The AMM communicator will fetch the checkpoint package from the local temporary storage designated by the coordinator for the process and will first deliver a meta package to the remote computing device containing checkpoint file name and size. The meta package will also define the type of application which is being migrated such migration-aware application and applications which do not have any semantical and syntactical checkpointing information. This meta package is required by the remote computing device checkpoint manager. Once the package is transferred to the remote device AMM will wait for the response from the remote computing device. Once the response package (also a checkpoint image) is received from the OSP, the AMM is also responsible for restarting the received checkpoint file in the client device. Furthermore, AMM will store the network performance metrics which are gathered from the communicator modules results. The network performance metrics will be used in the next subsequent offloading decision (offload or not to offload).

4.2.3 Application Migration Coordinator (AMC)

The application migration coordinator is a modified over the counter checkpointing tool which can checkpoint a running process and later restart on remote computing infrastruc-

ture and back on mobile devices ¹. An application executed through AMC will cause a coordinator process to be launched on the local client device to assist the checkpointing process (Ansel, Arya, & Cooperman, 2009). The coordinator is stateless, explicitly, if the coordinator is killed or crashed the process needs to be restarted. Furthermore, there are two approaches for the coordinator to issue checkpoint signal to the running applications. The first approach is applied to applications which do not have checkpointing/migration markers (i.e. non migration-aware applications). Meaning that they cannot self-checkpoint themselves. In MCC such applications can be termed as a unmodified application which does not have programmer defined migration/checkpointing markers. On the other hand, the second approach applies to the application which can be modified (either by the developer or automatically during the compilation process of application) to put migration markers which can readily checkpoint themselves. The first approach is resource hungry because it can checkpoints the application may be more than one time as it requires a manual checkpoint signals forwarded through AMM or checkpointing based upon time intervals. While the second approach is lightweight as it has the ability to checkpoint upon reaching a checkpoint marker.

The checkpoint mechanism interrogates kernel state (open file descriptors, file descriptor offset, etc.) and saves register values using available kernel data structures. Lastly, all user-space memory is stored to checkpoint image (/proc/self/maps). The checkpoint mechanism was configured to uses gzip to compress the checkpoint images. The compression is performed to reduce the effect on network traffic unless the data in memory is incompressible. The checkpointing process includes any libraries that the process was using. This strategy improves the portability of the checkpoint images and it even in some cases allows migration of the process to hosts with different Linux distributions, different Linux kernels, etc.

¹<https://github.com/dmtcp/dmtcp>

4.2.4 Server side admission control

Admission control modules enforce the fair-usage policy of server resources. In the case of a cloudlet with a weak pricing model based admission control, ensuring the efficacy of the system is vital (Whaiduzzaman, Gani, & Naveed, 2014b, 2014a). Admission control can regulate the number of active server users depending on the system utilization or a manual policy defined by the system administrator. To share the resources among different devices, requests will be queued and processed using any available scheduling policy (e.g., FIFO, round-robin) defined by the system administrator or dynamically selected based on the system load and other metrics. Furthermore, whenever a mobile device sends requests to the offloading server, the admission control will communicate with the instance manager to verify whether the device can be emulated/virtualized or not. Afterward, the admission control verifies the availability of the software platform is available. If both conditions are satisfied, the request is transferred and the ID of the emulated/virtualized instance received from the instance manager is sent back to the requesting mobile device. Once the request is granted, the subsequent process state synchronization packets from that device are directly transmitted/forwarded to the instance manager.

4.2.5 Server side application migration manager

The server side application migration manager will reside in a virtualized/emulated device or physical device instance depending upon the availability. This manager will serve as a server to the AMM residing on the user's mobile device. To maintain the policies implemented by the admission control, this coordinator will track the service time and active time that user uses the server resource and if there is a violation of that policy or the requested/paid time is over the will stop servicing the mobile device. Once the server-side migration manager starts receiving packets (meta information and checkpoint file) from the mobile device. The manager will then signal the checkpoint manager to along

with the location of the checkpoint which needs to be restarted. Once the application is restarted then either, it would be re-checkpointed based upon regular intervals for unmodified applications or applications which do not have checkpointing/migration markers. On the other hand, if the application is migration-aware then it will re-checkpoint itself once it reaches a checkpoint marker. The re-checkpointing is done to transfer the computation and pending execution back to the mobile device.

4.2.6 Proposed PMCO Algorithms

In this section, we present our proposed algorithms which originally correspond to the interaction between the components of the drafted framework detailed in section 4.2. The implementations of process migration based computational offloading algorithm employing checkpoint can vary depending upon the checkpointing methodology, hardware and software architecture. However, the primary steps in generic process migration according to (Vasudevan & Venkatesh, n.d.) can be summarized in the following steps:

1. A process migration request is issued to a remote computing device. After negotiation with server side Admission control, migration has been accepted.
2. The process is detached by suspending its execution on the mobile device, declaring it to be in a migrating state.
3. The process soft state is extracted, including memory contents, processor state (register contents), communication state (e.g., opened files and message channels) and relevant kernel context. The extracted process soft state is stored in a checkpoint file and is hardware & OS dependent.
4. Checkpoint file is transferred and restarted as a new process on the remote computing device.

Based on these generic step and the components of our proposed solution we present our process migration based computational offloading algorithm and a server side service algorithm to serve the end users as Algorithm 1 and Algorithm 2, respectively.

Algorithm 1 Offloading Algorithm

```

1:  $E_c, E_i, E_t, E_r, S_m, B_t \leftarrow \text{MPMGetGlobalPreferences}()$ 
2:  $S \leftarrow \text{AMMGetFindServer}()$ 
3:  $\beta_u, E_t \leftarrow \text{TestUpload}()$ 
4:  $\beta_d, E_r \leftarrow \text{TestDownload}()$ 
5: while isConnected(S) do
6:    $P, I, \alpha, \gamma, P_f, P_t \leftarrow \text{MPMGetProcessInfo}()$ 
7:   if  $P \neq \text{NULL}$  then
8:      $\text{AMC} \leftarrow \text{LaunchApplicationMigrationCoordinator}(E_c, E_i, E_t, E_r, \beta_u, \beta_d, \alpha, \gamma)$ 
9:      $\text{LaunchProcess}(P)$ 
10:    if  $P_t = \text{MigrationAware}$  then
11:       $O_t \leftarrow \text{AMMInfoPolling}()$ 
12:      if  $O_t \neq \text{NULL}$  then
13:         $\beta_u \leftarrow \text{TransferCheckpoint}(O_t, P_t, S)$ 
14:         $\beta_d \leftarrow \text{ReceiveCheckpoint}(O_r, S)$ 
15:         $\text{RestartCheckpoint}(O_r)$ 
16:      end if
17:    else
18:      if  $\text{Benefit}(E_c, E_i, E_t, E_r, \beta_u, \beta_d, \alpha, \gamma) > 0$  or  $P_f = 1$  then
19:         $\text{AMMSignalCheckpoint}(P)$ 
20:         $O_t \leftarrow \text{AMMInfoPolling}()$ 
21:         $\text{AMMSignalKill}(P)$ 
22:        if  $O_t \neq \text{NULL}$  then
23:           $\beta_u \leftarrow \text{TransferCheckpoint}(O_t, P_t, S)$ 
24:           $\beta_d \leftarrow \text{ReceiveCheckpoint}(O_r, S)$ 
25:           $\text{RestartCheckpoint}(O_r)$ 
26:        end if
27:      else
28:         $\text{ExecuteProcessLocally}(P)$ 
29:      end if
30:    end if
31:  end if
32: end while
33: while isNotConnected(S) do
34:    $P \leftarrow \text{MPMGetProcessInfo}()$ 
35:   if  $P \neq \text{NULL}$  then
36:      $\text{ExecuteProcessLocally}(P)$ 
37:   end if
38: end while

```

In the Algorithm 1, Line 1, initialize the algorithm with the global parameter values defined by the user in the MPM registry. Line 2, calls the AMM to find and negotiate

a suitable server according to the criteria defined by the user. Once the mobile device is connected to the server, a small test upload and download transmission is performed in Line (3-4). The test transmission is performed to gather the upload bandwidth β_u and download bandwidth β_d along with E_t and E_r if not set by the user in the global setting. Afterward, Line (5-32) is the main offloading loop conditioned by the server connectivity. Line 6, will query the MPM registry and get a process if listed along with the preferences defined for the process. Once an application listing has been discovered, Line 8, will execute the AMC along with, the parameters required for an offloading decision. Once the AMC starts running the application needs to be offloaded is Launched in Line 9. After that, Line 10, checks whether the application is migration-aware or not if the application is migration-aware the offloading decision benefit function equation (4.2) will be called inside the application execution life cycle on the point of offloading. Once that function is called and returns true, the application will be checkpointed, and AMC will be acknowledged about the checkpoint. This acknowledgment will be received on Line 11 (AMMInfoPolling()) of the offloading algorithm. The acknowledgment means that a checkpoint is ready and needs to be offloaded. Line 13, will transfer the checkpoint to the remote device. While line 14, will wait for receiving the updated checkpoint from the remote device. On the other hand, if the application is not migration-aware, the control will then jumped to Line 18, where the offloading algorithm will call the benefit function listed in equation (4.2). If the offloading benefit function returns true then in Line 19, AMM will signal the AMC to send a checkpoint signal to the process checkpoint thread. Once the process receive that signal, it will checkpoint itself and acknowledge AMC, which will be handled by the polling function in Line 20. Once the acknowledgment is received, the process would be then killed, and the checkpoint would be transferred to the remote device in Line 23. While Line 24, will wait for a response. If the benefit function listed in Line 18 returns false, then the application would be executed locally.

Line 33-37 will execute the application locally on the mobile device if and only if an MPM entry for the mobile application is available and is set to true, but the mobile device is not connected to an OSP.

The offloading decision function Equation (1), present a modified offloading decision function from Equation (4.1) which is tuned to consider process migration based computational offloading.

$$[E_m^p - E_m'^p - E_m''^p - E_t^p - E_r^p] > B_t, \text{ where } B_t \geq 0 \quad (4.2)$$

Where E_m^p , presents the energy consumption of process p on source mobile device m . $E_m'^p$ is the energy consumption of the check-pointing of process p on source mobile device m , while $E_m''^p$ represents the energy consumption of restarting the process from a checkpoint state (received from a remote computing device) on the on source mobile device m . While E_t^p and E_r^p is the energy consumption to transfer a checkpoint and receive a checkpoint of process p . The modification of the decision function is done due to the nature of process migration. Where the process is killed after the checkpointing and then transferred to the remote device, and then again received and restarted on the source device from the updated state.

Now from the prescriptive of the server side, there should be a service which should be published to enable its consumption by the process migration based computational offloading listed in Algorithm 1. The server side service function is listed as Algorithm 2.

The server side algorithm is straightforward, Line 2-3 accept a client connection and then a checkpoint file. Once the checkpoint file is received Line 4, will check if the application for which the checkpoint file is received is migration-aware, if so then will restart it with the process with its own coordinator on Line 5. Line 6, the AMM will

Algorithm 2 Offloading Service

```
1: while 1 do
2:    $C \leftarrow \text{AcceptConnection}()$ 
3:    $\text{ReceiveCheckpoint}(O_r, P_t, C)$ 
4:   if  $P_t = \text{MigrationAware}$  then
5:      $P \leftarrow \text{RestartCheckpoint}(O_r)$ 
6:      $O_t \leftarrow \text{AMMInfoPolling}()$ 
7:     if  $O_t \neq \text{NULL}$  then
8:        $\text{TransferCheckpoint}(O_t, S)$ 
9:     end if
10:  else
11:     $P \leftarrow \text{RestartCheckpoint}(O_r, \text{Interval})$ 
12:    if  $\text{isProcessExecutionFinished}(P)$  then
13:       $O_t \leftarrow \text{AMMInfoPolling}()$ 
14:      if  $O_t \neq \text{NULL}$  then
15:         $\text{TransferCheckpoint}(O_t, S)$ 
16:      end if
17:    end if
18:  end if
19: end while
```

start polling for readiness of the new checkpoint image to be sent to the client device.

On the other hand, if the process is not migration-aware than control will be jumped to Line 11, where the process will be restarted with its own coordinator on an interval based checkpointing. Once the process is restarted Line 12, will wait until the process execution is not finished. Once the execution is finished Line 13, will poll for the latest checkpoint image (which is generated based on interval timeout during program execution). The checkpoint image will be then sent back to client device where it is restarted.

Besides the algorithms presented above, to simplify and better the depict the process migration based computational offloading we have presented the flow diagrams of the process. The flow diagram represents use cases of (i) migration of a process from the mobile device to remote computing device; (ii) restarting of the process at both of the endpoints; and (iii) migration of the process back from the remote computing device to the mobile device.

Figure 4.5, presents the use case (i) where the application is under execution and needs to be offloaded will take an offloading decision based upon assistance from other

modules of the framework. Once the offloading decision is successful the process is paused and checkpointed, and the checkpoint file is then transferred to the remote computing device.

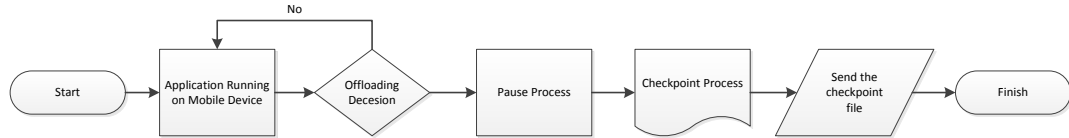


Figure 4.5: Flow diagram of process migration from mobile device to remote computing device.

Figure 4.6, present the use case (ii) where a process is restarted from a checkpoint file on both the remote computing device and mobile device (once returned from the remote computing device).



Figure 4.6: Restarting a process from the checkpoint file.

Figure 4.7, present the use case (iii) where once an application is executing on the remote computing infrastructure and finished after a checkpoint marker is reached or execution of the process finished completely. The checkpoint file generated on the remote computing device is then sent back to the requesting mobile device.

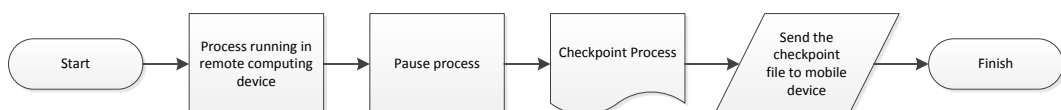


Figure 4.7: Flow diagram of process migration from remote computing device to mobile device.

4.2.7 PMCO execution flow using sequence diagram

To represent the interaction and execution flow of the components of the proposed framework according to algorithmic analogy sequence diagram sound to be the best candidate due to its ability of presentation based upon object lifeline. Figure 4.8 and 4.9 present the sequence diagram computational offloading using the proposed framework. Figure 4.8 present the computational offloading scenario of offloading migration-aware mobile applications. Whereas, Figure 4.9 presents the scenario of offloading non migration-aware mobile applications. The description in the following lines can be easily tracked using the components lifelines.

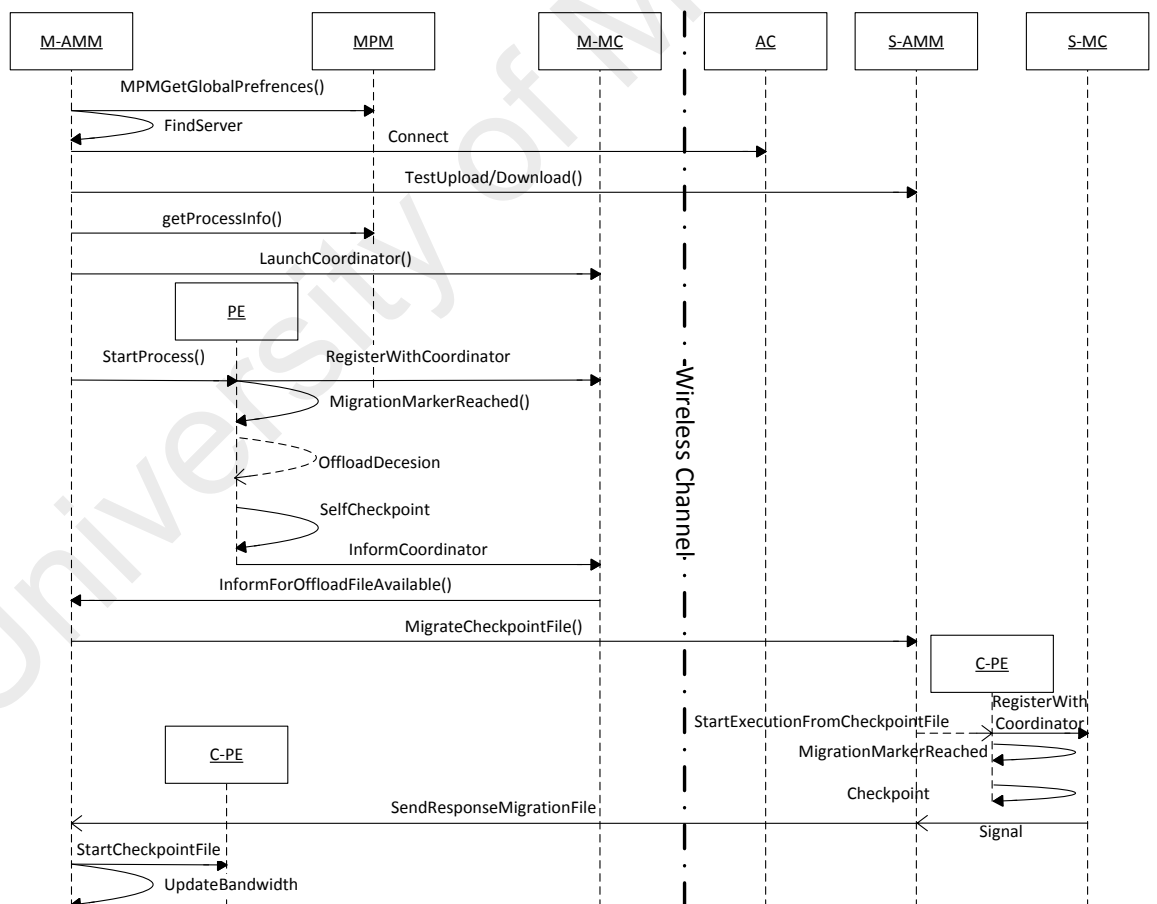


Figure 4.8: Sequence diagram of offloading a migration-aware application using the proposed PMCO based mobile device augmentation framework.

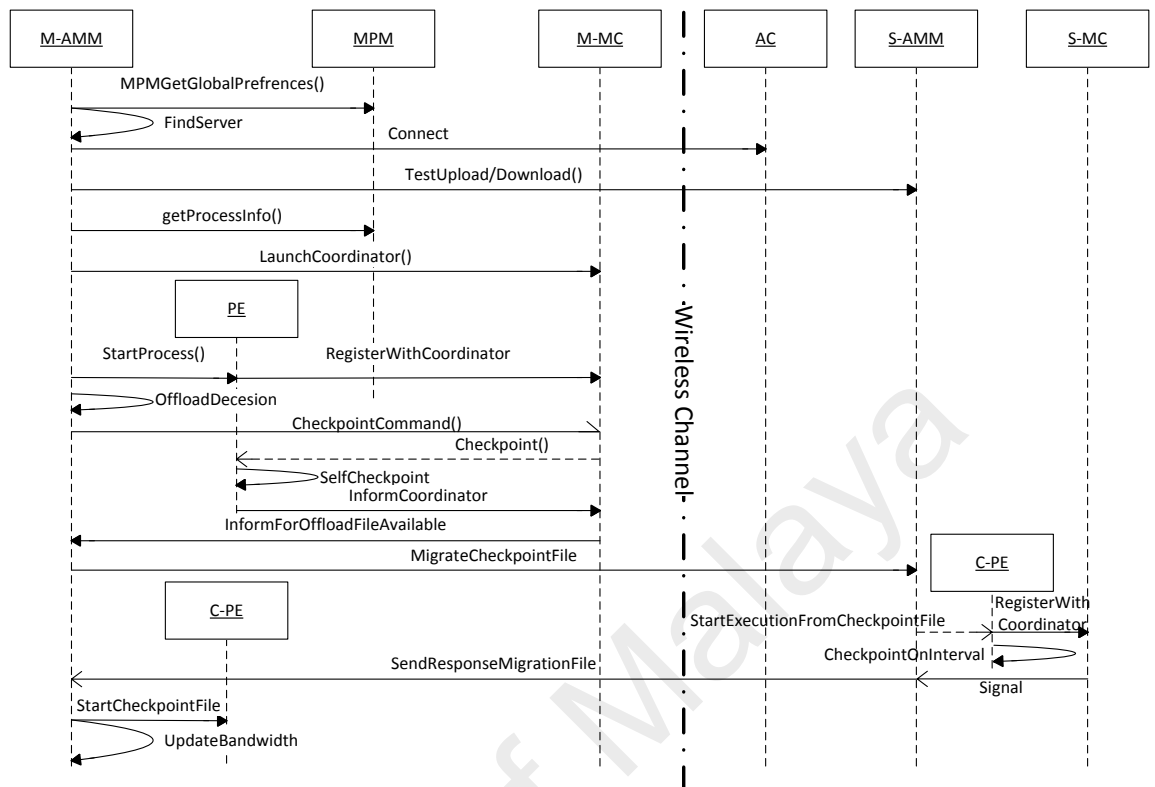


Figure 4.9: Sequence diagram of offloading a non migration-aware application using the proposed PMCO based mobile device augmentation framework.

In the sequence diagram presented in Figure 4.8 and 4.9 M-AMM stands for a mobile side application migration manager. MPM stands for Migration Preference Manager. M-MC stands for a mobile side application migration coordinator. AC stands for Admission control. S-AMM stands for server side application migration coordinator, and S-MC stands for server side application migration coordinator. PE stands for the process execution, while C-PE stands for starting execution from a checkpoint file. The interaction of these components is as follow.

The first seven steps in both the sequence diagrams (Figure 4.8 and 4.9) are same and correspond to: (i) getting global preferences; (ii) connection with server; (iii) test upload and download transmission; (iv) get process from MPM; (v) launch the application migration coordinator; (vi) execute the program which needs to be offloaded using

the checkpoint launcher program discussed in section 4.1.2; and finally register the application with the mobile side M-MC. Afterward, in scenario 1 Figure 4.8 application itself take the offloading decision, and if it yields true, the application will checkpoint itself and inform the coordinator which will in turn signal the mobile side M-AMM that a checkpoint parcel is ready to be shipped to a remote computing device.

On the other hand, in scenario 2 Figure 4.9 the M-AMM will take the offloading decision, and if it yields true, then the signal will be sent to M-MC to forward a checkpoint message to the application under execution. Once the checkpoint thread in the application receives the checkpoint signal from the M-MC, it will self-checkpoint the complete process and acknowledge M-MC and this acknowledgment will be relayed to M-AMM for migrating the checkpoint file to the remote device.

Once the checkpoint file is migrated from the client device, it is received by the S-AMM on the server side and restarted immediately. The sending back of the computation of both the scenarios are little different for scenario 1 Figure 4.8 the application after restarting will again checkpoint itself on the migration marker marked from where the computation needs to be done on the mobile device, and the checkpoint file will be sent to the device for restarting again. However, the non migration-aware applications after restarting at the server side will be checkpointed on an interval basis and only the last checkpoint after once the execution finishes will be sent to requesting mobile device.

On the server side, the AMM will restart the checkpoint file as a new process using a restart tool which will also span its own coordinator process. The application information received by server-side AMM will be used to define setting of sending back the result. If the application is migration-aware, then the application will re-checkpoint itself on the migration exit point while if the application is non migration-aware then the server coordinator will be configured and started in a way to perform an interval based checkpoint until the execution of the process does not finish. In the first case if the appli-

Table 4.1: Metrics for Performance Assessment of the Proposed Framework

Metric	Definition	Unit
Consumed Energy	Total amount of energy consumed on the device to complete one entire life cycle of a single program	Joule
Compute Power	From a benchmarking perspective, the compute power can be measured as the total amount of mega floating point operation or mega instruction per seconds computed for our experimental environment using standardized benchmark methods	MFLOPS, MIPS
Execution Time	Total time takes to complete one entire life cycle of a single workload	Seconds
Data Transfer	The amount of data transferred between a single offloading transaction	Bytes

ation is migration-aware the re-checkpoint file is straight away forwarded to the mobile device where it can be restart from the execution point where the server left it. While for non migration-aware applications once the application execution is finished the last checkpoint which is acquired will be transferred to the mobile device.

4.3 Data design

In this section, we describe the methods used to evaluate the performance of this framework. We also identify the performance evaluation metrics that are identified for evaluation of the framework.

4.3.1 Evaluation metrics

We identify the framework performance and consumed energy as two metrics of evaluation that are presented in Table 4.1 and explained as follows. These two metrics help us to obtain our aim and objectives in this study. Consumed energy and framework performance are the most important established metrics to evaluate the lightweight properties of a typical computing system.

- **Consumed Energy:** Consumed energy is the amount of energy consumed to complete the entire life cycle of one complete execution cycle of a program which

is measured and stated in Joules. The energy data is collected using PowerAPI (Bourdon, Noureddine, Rouvoy, & Seinturier, 2013) which is capable of collecting energy data for the computing devices which is based upon profcs. The energy consumed by the other components such as storage and LCD are disregarded in this study. To avoid man-made mistakes, the energy values are extracted and are processed using specialized results parser written to suit our data requirements.

- **Compute Power:** We measure the effectiveness of our proposed framework in a controlled environment and consider compute power as one of the important performance indicator. The compute power of the any (including the proposed) computing system can be better expressed as the amount of instruction per second or mega flop operations per second which are available to program to complete an entire life cycle of its execution. Theses performance metrics can be generated using different methods. However, we used standardized methods explained in next chapter. Similar to the energy consumption, to avoid man-made mistakes, the values are extracted and are processed using a specialized results parser written to suit our data requirements.
- **Execution Time:** Besides measuring the performance using standardized benchmarks. We measure another performance metric in term of the reduction of the total execution time of a set of the workload when executed locally and when offloading using the proposed method. Likewise, to avoid man-made mistakes, the values are extracted and are processed using specialized results parser written to suit our data requirements.
- **Data Transfer:** The third performance metric which we measured is the amount of data transferred between two endpoints in a successful offloading transaction. The extraction mechanism is same as for the other parameters.

4.4 Conclusion

In this chapter, we illustrated the proposed framework in this research and described its requirements, and characteristics. The schematic presentation of the proposed framework and its major building blocks are depicted. We also described our proposed framework by explaining functional and non-functional characteristics of its main components. Flow diagrams and sequence diagrams are used to outline the functional behaviors of major components of the framework. Moreover, several significant features of the framework are highlighted. Data design is described, and data generation procedures are explained. We have identified consumed energy, compute power, application execution time and the amount of data transfer in a successful offloading transaction as the performance evaluation metrics. Statistical modeling is determined as the performance evaluation and validation techniques. The data analysis methods that are utilized in the evaluation of this framework are presented.

CHAPTER 5: EVALUATION

This chapter presents the performance evaluation methods used to evaluate and validate the performance metrics of our proposed framework. To evaluate the proposed model and its lightweight features, we use standardized synthetic benchmarking experiments, also with a synthetic compute intensive application for measuring mega floating point operation per seconds, mega instruction per seconds application execution time, the amount of data transfer and consumed energy. Using thirty observation of each benchmark application, we collect and analyze the performance metrics. The evaluation results are validated via statistical modeling. To build our statistical model, we used independent replication method and validated the proposed model using split-sample approach. In another set of experiments, we build a separate test-bed for the comparative study of our proposed technique with the other offloading migration mechanisms to demonstrate the lightweight feature of our framework. Lastly, we describe the statistical data analysis methods used to analyze and synthesize the results.

The remainder of this chapter is as follows. Section 5.1 presents the experimental setup along with data generation methods. Section 5.2 presents the evaluation methods and explain how the statistical models are validated. Section 5.3 presents our parametric analysis and describes the statistical data analysis observations to analyze and synthesize the results. The chapter concludes in section 5.4.

5.1 Experimental setup

To benchmark the proposed prototype we selected four standard and different synthetic computing benchmarks, and a synthetic compute intensive application with various granularity of execution inputs. The selected synthetic benchmarks are (i) Dhrystone¹, (ii)

¹<https://en.wikipedia.org/wiki/Dhrystone>

Table 5.1: Benchmark Matrix Multiplication Granularity

Workload#1	Matrix Granularity
1	300x300
2	400x400
3	500x500
4	600x600
5	700x700
6	800x800
7	900x900
8	1000x1000

Whetstone², (iii) Linpack³, and (iv) Scimark2⁴. The Scimark benchmark was configured to execute the large instance; the configuration can be set as command line parameter using the source code provided by NIST. While the compute intensive application is a matrix multiplication application with different matrix granularities presented in Table 5.1. Furthermore, all the applications (benchmarks+matrix multiplication program) have been annotated with migration points enabling their self-checkpointing and compiled using standard GNU Compiler Collection (GCC) with -FPIC (position independent machine code) switch enabled while compiling the applications. Which is essentially required by the checkpointing engine to checkpoint and restart the process transparently in user space.

The primary data of performance evaluation are collected by testing the prototype applications on both the Android and real mobile cloud environment in three different scenarios. In the first scenario, all the components of the mobile application are executed on the local mobile device to analyze performance evaluation metrics of the application on the local mobile device. In the second scenario, the application is again executed on the local mobile device using the proposed framework so that the framework overhead energy consumption and application execution time can be analyzed. In the last scenario,

²[https://en.wikipedia.org/wiki/Whetstone_\(benchmark\)](https://en.wikipedia.org/wiki/Whetstone_(benchmark))

³<https://en.wikipedia.org/wiki/LINPACK>

⁴<http://math.nist.gov/scimark2/>

the components of the mobile application are offloaded at runtime by implementing the proposed computational offloading techniques. The schematic presentation of our benchmarking setup is illustrated in Figure 5.1 and described as follows.

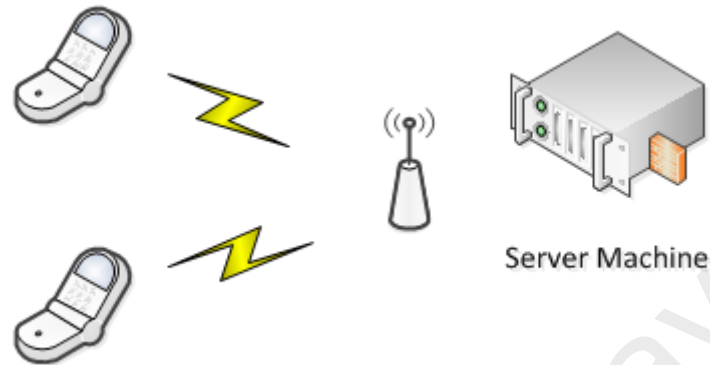


Figure 5.1: Network topology of experimental setup.

Previously discussed in section 3.3 and in the preamble of Chapter 4 the process state is platform dependent. To prepare the real experimentation environment, we need same hardware and operating system architecture on both endpoints of the experimental setup presented in Figure 5.1. In this response to prototype the proposed mechanism we used a Samsung Galaxy S-II i9100g as the client devices. The client device is equipped with a Dual-core 1.2 GHz TI OMAP 4430 ARM Cortex-A9 SoC and 1 GB RAM. The client device is connected to the server device using campus WiFi network. The server device is a high-performance Sony Xperia Z Ultra equipped with Quad-core 2.2 GHz Qualcomm MSM8274 ARM Cortex-A9 SoC. To further make a difference between the client and server device, we down-clocked the client device from 1.2 GHz to 600 MHz to emulate a resource-poor client device and a resource-rich server device.

From the operating system point of view, both devices are by default coming with Android OS. However, for our experimental setup, we utilized the Android kernel but deployed a standard ARM port of Ubuntu 13.10 Saucy Salamander in a chroot environment over both of the devices. This chroot jail over Android allows us to take advantage of

full capabilities of standard GNU Glibc (missing in Android) and Android kernel (Linux kernel), especially in accessing process state from user spaces.

From the component implementation point view, the application migration manager is written in Java providing the client-server communication interface between both endpoints. While for the checkpoint management and coordination, we exploited and modified DMTCP ⁵, a multi-purpose checkpointing mechanism for parallel and distributed computing environment. DMTCP is a community driven having progressive enhancement by which the framework can be ported to newer platforms and environments enabling its portability to a large extent.

We evaluate the performance of each of the benchmark and eight different granularities of the matrix multiplication workload in three modes of Local, Local_PMCO, and the proposed PMCO. Each benchmark and matrix multiplication workloads are executed thirty times whose mean value is considered for analysis. To enhance the reliability of the results the findings are presented with 95% confidence interval in this experiment to ensure data collection is unbiased.

To collect the energy consumption of the complete application under execution, we monitor the application and the framework components using PowerAPI (Bourdon et al., 2013) in console mode, a granular tool for investigating power consumption of running applications based on chipset Thermal Design Power (TDP). The TDP value for the client device (Samsung Galaxy SII i9100g) in our case is set to 0.6W ⁶. PowerAPI also provides a feature to monitor the aggregate energy consumption of a group of processes, which in our scenario of remote execution (i.e. process migration) is configured to monitor the power consumption of the AMM and AMC components of the proposed framework. Energy consumed by other software components are not considered in this data collection

⁵<http://dmtcp.sourceforge.net/>

⁶<http://www.notebookcheck.net/Texas-Instruments-TI-OMAP-4430-SoC.86865.0.html>

phase, and we run the applications in active mode throughout the experiment by preventing from pausing or blocking the application execution. Furthermore, we parse the power profile output generated by PowerAPI using bash scripts to generate CSV files for analysis.

The improvement in the compute power is measured by executing the benchmark applications, which are specifically designed for this purpose and used widely for benchmarking computing systems and hardware. The migration segments of the benchmark applications comprise the start and end of the main function of the application so that the generated benchmark values are not biased and provide with a real increase in the compute power based upon the server device. The benchmark execution results are encapsulated automatically in the checkpoint parcel by the checkpoint manager on the server as we used the output redirection to a file on the server device which enabling the checkpointing process to store that file in the parcel also and once restarted on the client device the file is created then.

The improvement in execution time is measured using the matrix multiplication applications, as benchmarks are not designed for this purpose as they would in most of the cases run for the same time units on all devices to provide an unbiased analysis of the compute capabilities. The execution time of matrix multiplication application is measured by the application itself as we enclosed the migration segment of the source code with timers to provide details about the execution time while executing locally or remotely. This will provide the execution time of the program on the system on which it is executing now. To compute the remote execution time we have added the checkpoint restart overhead time, and added the transmission and reception time as well. Their respective components generated these values and stored them, in a formatted log file which is then processed using bash scripts to generate CSV files for analysis.

Our last parameter is the amount of data transfer; it is the size of the checkpoint file

which is first migrated from the mobile device, and then an updated checkpoint state is transferred by the mobile device from the server. The amount of data transfer is logged by the AMM which to provide the bandwidth consumption details. The log file is processed as the other parameters.

5.2 Evaluation methods

To assess reliability and validity of our research, we perform several statistical analyses on primary data generated via executing workloads and statistical modeling. In the following section, we describe each of the statistical methods being used in this research.

5.2.1 Descriptive statistics

To analyze data, describe improvements, and highlight significance of achievements in consumed energy, compute power, execution time and the amount of data transfer for local and PMCO modes, descriptive statistics including minimum, maximum, and mean are determined. Desired descriptive data are obtained for data collected and are summarized in tabular and graphical presentations to fulfill the desired objectives.

5.2.2 Confidence interval

According to the sample central limit theorem, approximately 95% of the sample means fall within 1.96 standard deviation of the population mean, provided that the sample size is greater than or equal to 30 ($n \geq 30$). Hence, application in the experiments is executed 30 times for the evaluation of each parameter to derive the value and verify that the sample is one of the representative samples. The measurement of central tendency of the data sample of each experiment is calculated by using the sample mean, for the reason that sample mean is ascertained the better point estimate of the population mean as compared to median or mode. Data sampling involves the factor of sampling error; hence the sample mean can differ from the population mean. Hence, to signify the good-

ness of the calculated point estimate; the interval estimate of each sample is determined. The interval estimate for each sample mean of the primary data is calculated with 95% confidence interval by using the following equation. Therefore, we raise the confidence and reliability of results up to 95% when reporting the parametric results.

$$\bar{X} \pm 1.96 \frac{\sigma}{\sqrt{n}} \quad (5.1)$$

Whereas, σ indicates the standard deviation in the sample values and n indicates the size of sample space.

5.2.3 Paired samples t-test

In this research, we use Paired Samples T-Test to ensure that there is a significant difference between the mean values of the identical measurement made in two different execution modes (Local execution vs. PMCO). In our study, local parametric values and PMCO parametric values are paired data of the same workloads executed in two separate modes. We use this test to ensure if execution modes (local and PMCO) have significant impacts on energy, compute power and time or not. In other words, with the help of the results from paired samples t-test we can show that consumed energy, compute power, and execution time values in local and PMCO modes have significant differences.

5.2.4 Linear regression

In this section, we describe our statistical analysis model, using the results of the statistical model we can verify the reliability of the concluded results from the experiments data.

For producing the statistical model of consumed energy, execution time and the amount of data transfer for local and PMCO modes, we employ independent replication method to generate independent dataset consisting of the measured consumed energy, execution time and amount of data transfer for new independent workloads in local and

PMCO modes. The same testbed described in Section 5.1 is used, except the fact, the benchmarks workloads are not used in regression analysis as there is a no correlation and cannot be configured to execute in different intensity levels. However, the prototype matrix multiplication is used for regression analysis due to correlation between the computational intensity of different configurations. This is enough to verify and validate the behavior of the data generated by the experiments.

We train the linear regression model to identify the correlations between the matrix multiplication workloads and execution time as well as multiplication workloads and execution time and amount of data transfer and also between execution time and consumed energy to derive the regression models of time and energy. Using the regression models, the consumed energy, execution time and amount of data transfer can be generated to validate the performance evaluation findings undertaken via experimental analysis.

To validate our regression model, we leverage split-sample approach and perform the calibration-validation exercise. Thus partial dataset is used to build the model and the rest to validate the results of the model. To perform validation, we randomly split the sample into two different size samples and identify the correlations between the dependent and independent variables. If the results are supporting each other, the model is valid.

5.3 Parametric evaluations

The following section presents the data collected in different experiments for the evaluation of the proposed process migration based computational offloading framework. The data are presented from the perspective of performance metrics (i) Compute Power, (ii) Execution Time, (iii) Energy Consumed, and (iv) Data Transfer by the workloads in three different scenarios i.e.: 1) Execution of the application on local mobile device, 2) Execution of the application on local mobile device by the components of the proposed framework referred as Local_PMCO beyond this point 3) Execution of the application

using process migration to the remote device referred as PMCO.

5.3.1 Data transmission

The amount of data transfer and reception is investigated to support the accuracy of execution response time and energy consumed in the proposed PMCO framework. However, when a workload is executed on local mobile devices it does not need to transfer its computations to the remote computing device, so this parameter does not apply to the local computing. However, as stated earlier the execution time and energy consumed when a computation is offloaded depends upon the data transfer. Table 5.2, represents the mean data transfer and reception in bytes of 30 observations with error margin according to 95% confidence interval while executing the workload in PMCO mode of execution (where the application is migrated using the proposed framework when it reaches a migration marker). Table 5.7, presents the mean execution time plus the error rate based on 95% confidence interval of matrix multiplication workloads having different computational intensities determined by the criminality of matrices. Table 5.7, also presents the average amount of time it takes to transfer and receive the workload to the remote computing infrastructure upon migration time and response time, based on the experimental setup detailed in section 5.1. The time values are also based upon the 95% confidence interval. As it is evident, when the workload size increases the amount of data transmission will also increase but on the other hand the amount of time required to transfer and receive does not have a strong correlation with workload size as compared with the network dynamics. To investigate this behavior, we have statistically modeled the data transfer and reception time in an offloading transaction.

The statistical modeling details are already presented in section 5.2.4. Furthermore, the statistical model for the data transfer and reception time will be then used in the model for execution time and consumed energy for corresponding workloads to identify the

Table 5.2: Average amount of data transfer and received, along with the average time it takes, based on 95% confidence interval

	Data Transfer(Bytes)	Data Received(Bytes)	Transfer Time (Seconds)	Data Reception Time (Seconds)
Dhrystone	1873158 ± 406	1907801 ± 365	0.07016 ± 0.00245	0.05074 ± 0.00321
Whetstone	1870997 ± 363	1903545 ± 358	0.0709 ± 0.00348	0.04913 ± 0.00232
Linpack	1878427 ± 448	2067430 ± 316	0.07773 ± 0.00314	0.05792 ± 0.00321
Scimark-FFT	1870491 ± 267	1902899 ± 212	0.0859 ± 0.01911	0.06034 ± 0.00977
Scimark-SOR	1870839 ± 339	2026929 ± 337	0.07825 ± 0.00543	0.06978 ± 0.00581
Scimark-MonteCarlo	1870536 ± 182	1903174 ± 146	0.0774 ± 0.00277	0.08029 ± 0.00639
Scimark-SparseMatMult	1870997 ± 363	1903545 ± 358	0.0709 ± 0.00348	0.04913 ± 0.00232
Scimark-LU	1870734 ± 273	2030351 ± 299	0.07254 ± 0.00157	0.06863 ± 0.01187
Mat. Mult. (300x300)	2189342 ± 384	2201717 ± 136	0.21606 ± 0.02481	0.17327 ± 0.0082
Mat. Mult. (400x400)	2438353 ± 834	2437188 ± 245	0.19705 ± 0.02534	0.17371 ± 0.01027
Mat. Mult. (500x500)	2754353 ± 637	2738264 ± 372	0.19631 ± 0.0207	0.19269 ± 0.01218
Mat. Mult. (600x600)	3144045 ± 633	3109320 ± 556	0.49064 ± 0.12438	0.22865 ± 0.01946
Mat. Mult. (700x700)	3602758 ± 291	3559345 ± 578	0.66713 ± 0.24228	0.3476 ± 0.03194
Mat. Mult. (800x800)	4132230 ± 284	4057875 ± 582	0.42294 ± 0.04018	0.44645 ± 0.03209
Mat. Mult. (900x900)	4732427 ± 238	4636233 ± 552	0.47579 ± 0.0361	0.39975 ± 0.01981
Mat. Mult. (1000x1000)	5403381 ± 280	5283824 ± 581	0.63774 ± 0.02102	0.46386 ± 0.02316

correlations between the workloads and execution time as well as between execution time and consumed energy. For the sake of simplicity, we are using the matrix multiplication workloads to statistical model the amount of data transmission (transfer and receive) in PMCO mode of execution.

Let us assume that to transmit n bytes of data from the mobile device to the remote server it takes $O(n)$ time and the same also applies for reception of n bytes. Then in matrix multiplication offloading transaction the asymptotic upper bound to for the transmission time as:

$$T_{transmission}(W_i) = T_{transfer}(W_i) + T_{receive}(W_i) \quad (5.2)$$

whereas, $T_{transfer}(W_i)$ and $T_{receive}(W_i)$ are:

$$T_{transfer}(W_i) = O(2 \times \chi \times (N \times N)) \quad (5.3)$$

$$T_{receive}(W_i) = O(\chi \times (N \times N)) \quad (5.4)$$

where, N is the granularity of the matrix workload, and χ is the amount of bytes

it takes by the data type of the matrix elements. The multiplication with 2 is because when an offloading transaction a matrix workload is initiated only the input matrices are declared and initialized, so it becomes a factor of two. However, this is not the case on the reception time from the server, as the matrix multiplication process restarted on the server-side while only send back the resultant matrix.

Now the data transfer and reception time complexity is used as an input variable to the regression model as:

$$T_{transmission}(W_i) = O(2 \times \chi \times (N \times N)) + O(\chi \times (N \times N)) \quad (5.5)$$

Now first we will model the data transfer time. For the sake of simplicity and accuracy, we consider $c_i = 2 \times \chi \times (N \times N)$, c_i represents the number of bytes needs to be transferred. Additionally, we consider $d_i = \chi \times (N \times N)$, d_i represents the number of bytes needs to be received in an offloading transaction. Hence, the statistical model of the data transmission time for the matrix multiplication workload is:

$$T_{transmission}(W_i) = ((m \times c_i) + j) + ((l \times d_i) + k) \quad (5.6)$$

Now before regression, the linearity of matrix multiplication data transfer and reception time must be ensured; otherwise, the linear regression can be misleading. The scatter diagram for matrix multiplication workload and corresponding data transfer time and data reception is presented in Figure 5.2 and 5.3, respectively. As the results show in Figure 5.2, the relationship between the workload sizes and data transfer time of matrix multiplication is non-linear verified the R^2 also which is about 0.17. Furthermore, in Figure 5.3, the relationship between the workload sizes and data reception time of matrix multiplication is a better linear fit as compared to the transfer time. This is non-linearity

is due to the dynamics of the underlying wireless network, which fluctuates and depresses the workload size as the predictor by noisy observations.

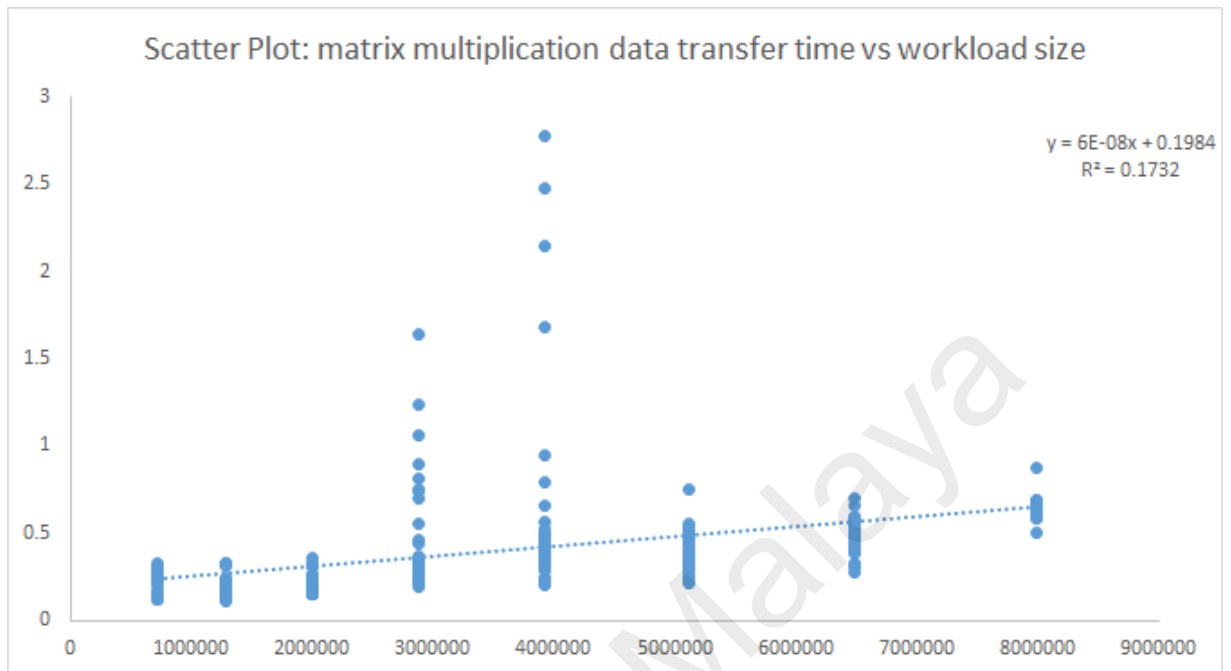


Figure 5.2: Scatter plot of matrix multiplication noisy data transfer time, with the linearity correlation determined by the line of best fit.

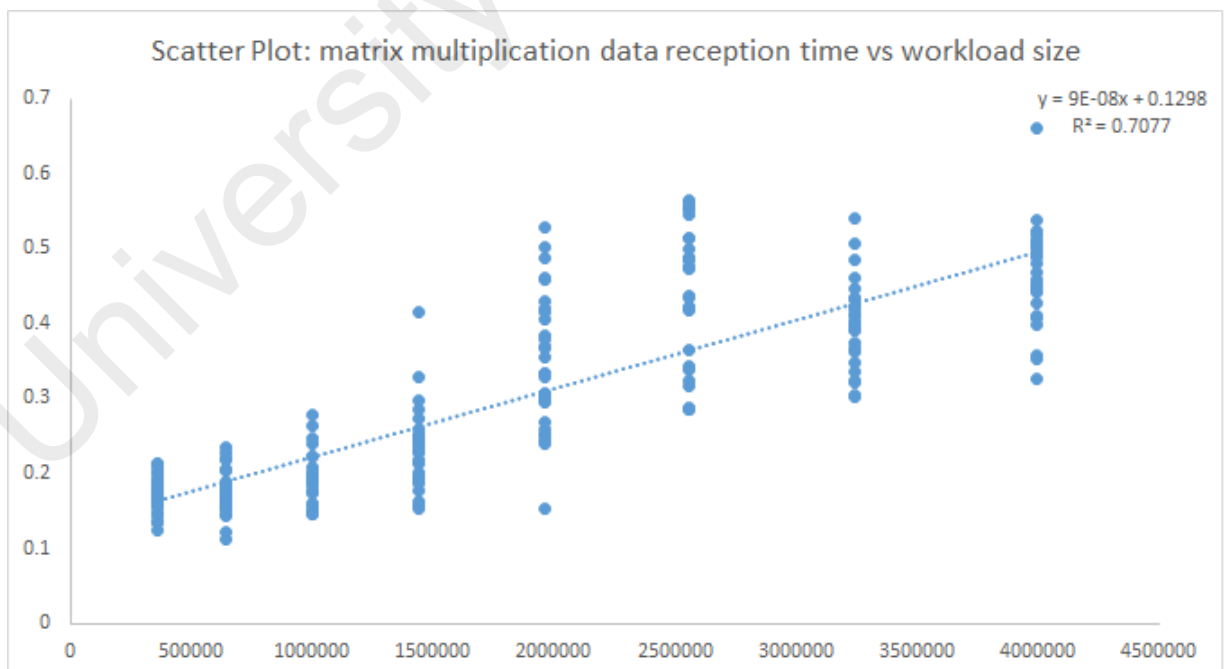


Figure 5.3: Scatter plot of matrix multiplication noisy data reception time, with the linearity correlation determined by the line of best fit.

However, due to the large sample size (240 observations), we can remove some

outliers to move the model towards linearity and increase its validity and usefulness in determining the execution time for remote processing using PMCO. The noisy outliers are removed for corresponding observations of the data transfer and reception time so that the model are consistent. After removing the noisy outliers we have improved the adjusted R^2 of the data transfer time from 0.17 to 0.92, the scatter plot of the removed noised from data transfer time is presented in Figure 5.4. Similarly, after removing the noisy outliers we have improved the adjusted R^2 of the data reception time from 0.70 to 0.82, the scatter plot of the removed noised from data transfer time is presented in Figure 5.5. The noise removal process has improved the model and its linearity to be used in further analysis.

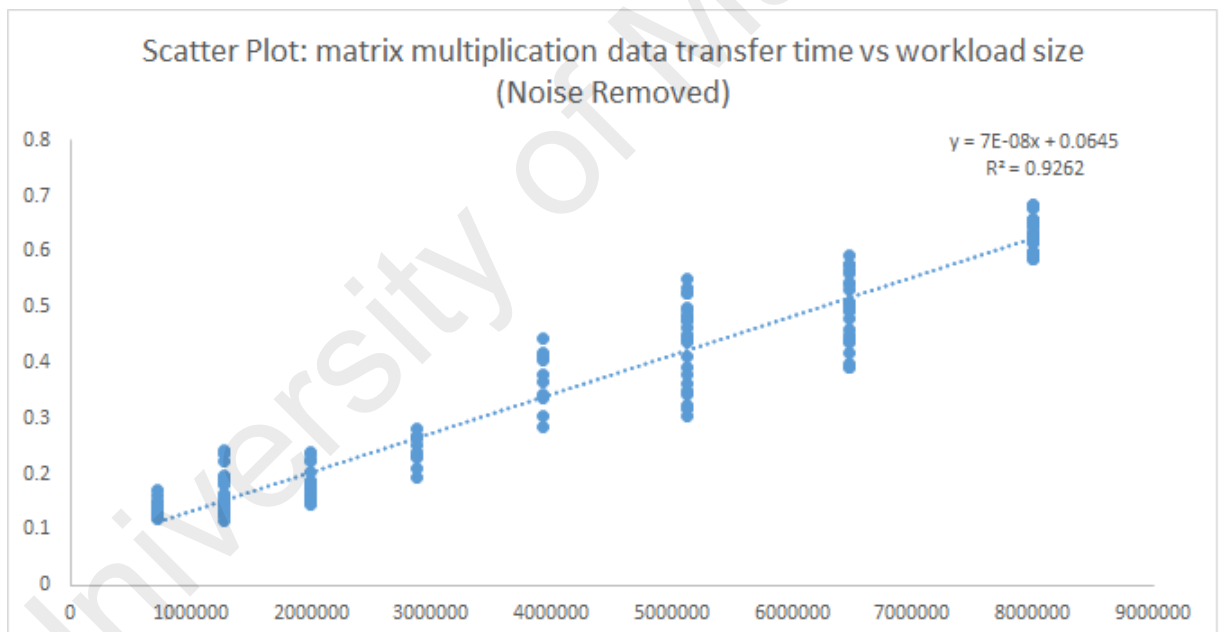


Figure 5.4: Scatter plot of matrix multiplication normalized data transfer time, with the linearity correlation determined by the line of best fit.

Thus the linear regression is feasible to model this relationship. The linear regression results for the normalized data transfer time to determine the m and j are summarized in Table 5.3.

To determine the m and j values, we utilize matrix sizes of the measured data set and

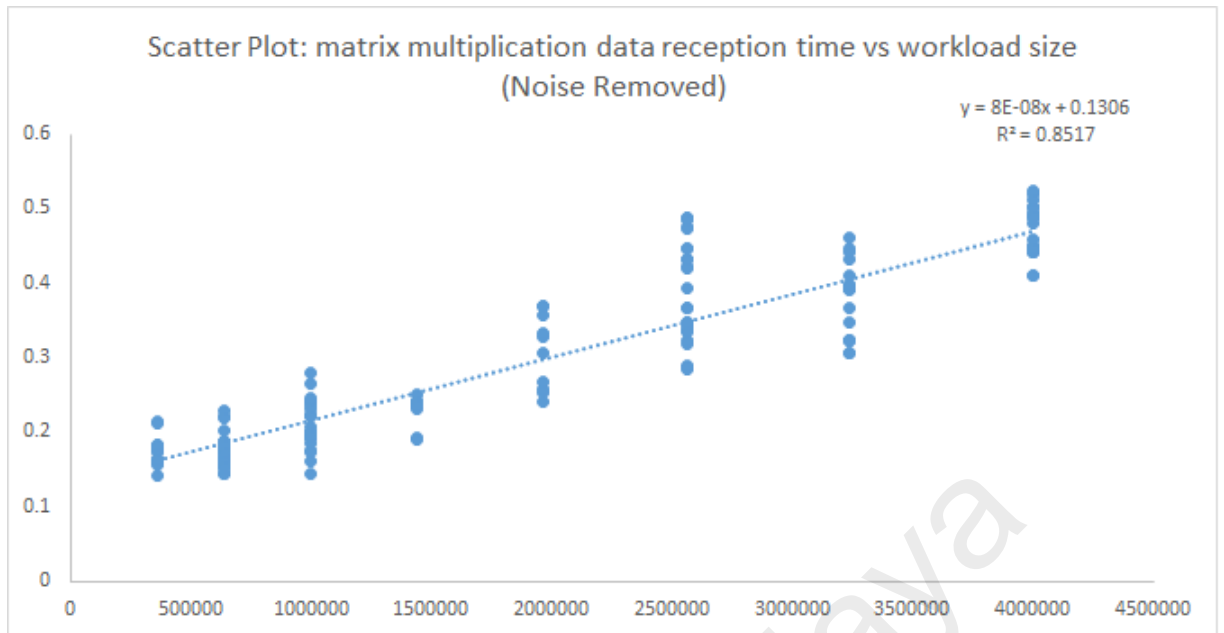


Figure 5.5: Scatter plot of matrix multiplication normalized data reception time, with the linearity correlation determined by the line of best fit.

Table 5.3: Regression statistics summary for data transfer time of matrix multiplication in PMCO mode

Multiple R	0.962390523
R Square	0.926195519
Adjusted R Square	0.925716269
F	1932.594176
Significance F	4.63737×10^{-89}
Intercept	0.064535721
X Variable 1	6.97687×10^{-8}
Observations	156

their corresponding normalized data transfer time to train the linear regression model.

The results of the linear regression analysis for the matrix multiplication are given in Table 5.3. The R value in the Table testifies correlation between the workload size and

the amount of data transfer time. The R^2 in the Table explains that the data transfer time can be 92.61% explained using the given workload dimension. Adjusted R^2 ensures that the predictor (workload sizes as independent variable) is almost an appropriate regressor.

The results of m and j values coefficients are determined by the linear regression line fitting as 6.97687×10^{-8} and 0.064535721. Therefore, the data transfer time model of

Table 5.4: Regression statistics summary for data reception time of matrix multiplication in PMCO mode

Multiple R	0.922889885
R Square	0.851725741
Adjusted R Square	0.850762921
F	884.6158767
Significance F	1.03254×10^{-65}
Intercept	0.130619288
X Variable 1	8.46526×10^{-8}
Observations	156

the matrix multiplication in PMCO execution mode is:

$$T_{transfer}(W_i) = (6.97687 \times 10^{-8} \times c_i) + 0.064535721 \quad (5.7)$$

Now, the linear regression results for the normalized data reception time to determine the l and k are summarized in Table 5.4.

The R value in the Table 5.4, testifies correlation between the workload size and the amount of data reception time. The R^2 in the Table explains that the data transfer time can be 85.17% explained using the given workload dimension. Adjusted R^2 ensures that the predictor (workload sizes as independent variable) is almost an appropriate regressor. The results of l and k values coefficients are determined by the linear regression line fitting as 8.46526×10^{-8} and 0.1306192. Therefore, the data reception time model of the matrix multiplication in PMCO execution mode is

$$T_{receive}(W_i) = (8.46526 \times 10^{-8} \times d_i) + 0.1306192 \quad (5.8)$$

Now, the data transmission time model of the matrix multiplication in PMCO execution mode presented in equation (5.6), can be re-written as:

$$T_{transmission}(W_i) = ((6.97687 \times 10^{-8} \times c_i) + 0.064535721) + ((8.46526 \times 10^{-8} \times d_i) + 0.1306192) \quad (5.9)$$

Table 5.5: Validation of data transfer time statistical model using split-sample approach

Split Sample 1		Split Sample 2		Original Sample	
Multiple R	0.962846183	Multiple R	0.963859585	Multiple R	0.962390523
R Square	0.927072772	R Square	0.9290253	R Square	0.926195519
Adjusted R Square	0.92636474	Adjusted R Square	0.927576837	Adjusted R Square	0.925716269
df	104	df	50	df	155
Observations	105	Observations	51	Observations	156

Table 5.6: Validation of data reception time statistical model using split-sample approach

Split Sample 1		Split Sample 2		Original Sample	
Multiple R	0.932895243	Multiple R	0.901898535	Multiple R	0.922889885
R Square	0.870293535	R Square	0.813420968	R Square	0.851725741
Adjusted R Square	0.869034249	Adjusted R Square	0.809613233	Adjusted R Square	0.850762921
df	104	df	50	df	155
Observations	105	Observations	51	Observations	156

To validate our devised model, we perform split-sample procedure explained earlier. We split the sample data into two randomly selected partitions for PMCO data transmission time. For each partition, we determine the correlation coefficients and compare the results of both partitions with the full sample to ensure validity. The comparison results are summarized in Table 5.5 and 5.6. As the results in Table 5.5 shows, the model produces identical R , R^2 , and adjusted R^2 for all three different samples, whereas in Table 5.6 there is slight deviation of up to 0.4% in the values of R^2 , and adjusted R^2 from the original sample. The degree of freedom (df) column in the tables shows that the size of each sample is unique and random. The adjusted R^2 is alike for all the splits which is an evidence of the validity of our proposed statistical model.

5.3.2 Execution time

To analyze the impact and decrease in the execution time of the matrix multiplication workload based on the experimental setup explained in section 5.1, each of the matrix workloads with different granularity is executed thirty times in all of the execution modes. The reason why the benchmark workload are not included in this analysis is because benchmarks application normally executed for the same amount of time (except Scimark2) over different systems, so that make fair comparisons. Table 5.7, represents the

Table 5.7: Mean execution time of matrix multiplication workloads with 95% confidence interval, Using three execution modes

Workload #	Workload	Local	Local_PMCO	Deterioration %	PMCO	Improvement %
1	Mat. Mult. (300x300)	4.36 ± 0.026	4.46 ± 0.04	2.29	2.32 ± 0.03	46.78899083
2	Mat. Mult. (400x400)	11.04 ± 0.070	11.36 ± 0.08	2.90	5.24 ± 0.04	52.53623188
3	Mat. Mult. (500x500)	22.36 ± 0.173	22.75 ± 0.2	1.74	9.91 ± 0.03	55.67978533
4	Mat. Mult. (600x600)	36.12 ± 0.083	36.31 ± 0.13	0.53	17.02 ± 0.14	52.87929125
5	Mat. Mult. (700x700)	60.94 ± 0.184	61.77 ± 0.38	1.36	27.93 ± 0.26	54.16803413
6	Mat. Mult. (800x800)	107.70 ± 0.271	108.79 ± 0.2	1.01	42.01 ± 0.07	60.99350046
7	Mat. Mult. (900x900)	141.06 ± 0.418	141.8 ± 0.36	0.52	62.49 ± 0.07	55.69970225
8	Mat. Mult. (1000x1000)	199.73 ± 0.664	200.93 ± 0.54	0.60	87.61 ± 0.15	56.13578331

mean execution time of thirty observations of each matrix multiplication workload and Scimark components along with error margin according to 95% confidence interval while executing the workload on the local mobile device, Local_PMCO, and using PMCO to the remote computing devices. Table 5.7, presents the mean execution time plus the error rate based on 95% confidence interval of matrix multiplication workloads having different computational intensities determined by the criminality of matrices. Table 5.7, also presents the deterioration percentage (increase in execution time), when the workload is executed locally through the components of the proposed frameworks. The deterioration is observed to be as less as 0.6% and at max is around 2.9%. Furthermore, the Table 5.7, also represents the percentage of improvement (reduction in execution time) of corresponding matrix multiplication workloads. The improvement is substantially ranging from almost 46% to 61%.

Linear regression method is applied to statistically model the behavior of execution time in local execution mode and PMCO mode. The details of the linear regression setting are described in subsection 5.2.4. We describe the statistical modeling of execution time in local and PMCO execution modes as follows. The devised models are validated using split-sample approach.

5.3.2.1 Statistical modeling of execution time in local execution mode

For the matrix multiplication workload, the application generates two random matrices of size $[N \times N]$ matrices and calculates the product of these two matrices into another $[N \times$

N] matrix. The matrix multiplication algorithm is implemented using the classical tri-loop method with running time of $O(N^3)$, where N represents the dimension of the generated matrices. Therefore, the upper asymptotic runtime bound for the matrix multiplication application on local mobile device is

$$T_{local}(W_i) = O((N_i)^3) \quad (5.10)$$

The multiplication complexity is used as an input variable to the regression model. For the sake of simplicity and accuracy, we consider $c_i = N_i^3$, where c_i represents the number of the coefficient of the matrices involved and will allow use to convert the cubic equation to a linear equation. Hence, the statistical model of execution time for the matrix multiplication is:

$$T_{local}(W_i) = (m \times c_i) + j \quad (5.11)$$

which is a linear equation. Before performing regression modeling, the linearity of matrix multiplication function must be ensured; otherwise, the linear regression can be misleading. The scatter diagram for matrix multiplication workloads and corresponding execution times appears in Figure 5.6. As the results show, the relationship between the workload sizes and the respective execution time of matrix multiplication function as expected is linear. Thus the linear regression is feasible to model this relationship. The linear regression results for determining the m and j are summarized in Table 5.8.

To determine the m and j values, we utilize workload sizes of the measured data set and their corresponding execution time to train the linear regression model. The results of the linear regression analysis for the matrix multiplication are given in Table 5.8. The R value in the Table testifies full correlation between the workload value and its execution

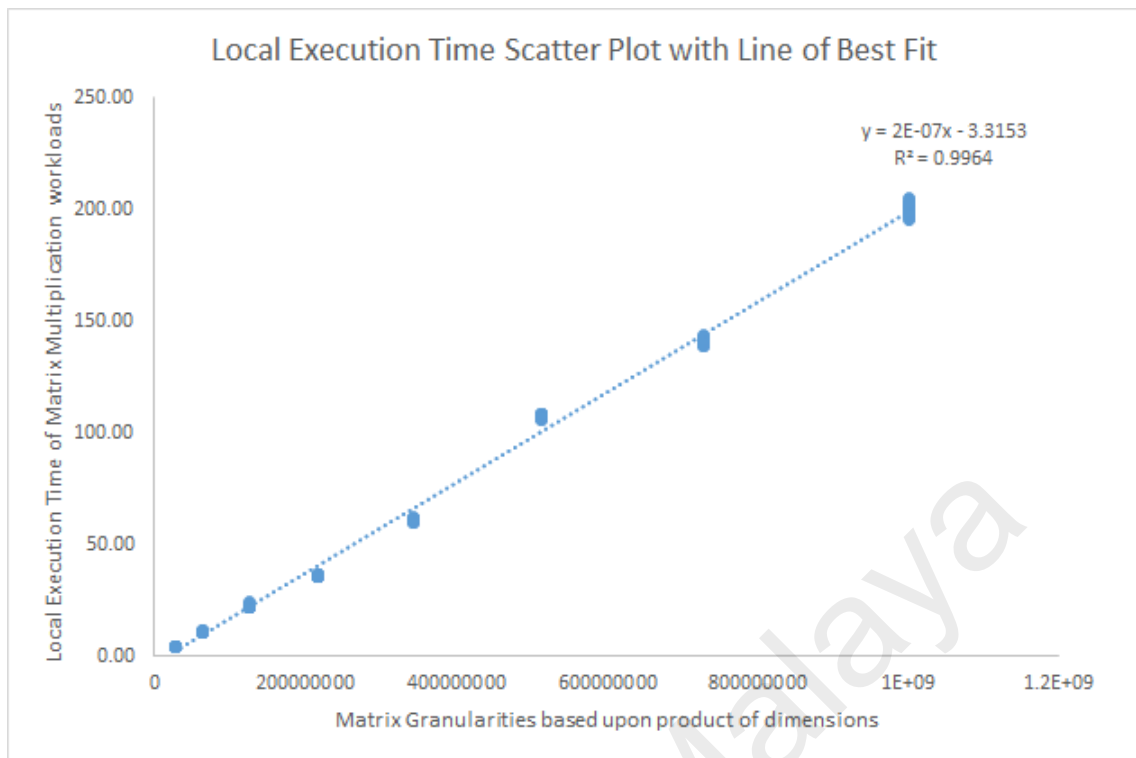


Figure 5.6: Scatter plot of Local matrix multiplication execution time, with the linearity correlation determined by the line of best fit.

Table 5.8: Regression Statistics Summary for Local execution time of matrix multiplication

Multiple R	0.998221727
R Square	0.996446615
Adjusted R Square	0.996431685
F	66740.39731
Significance F	1.7393×10^{-293}
Intercept	-3.315317243
X Variable 1	$2.02212695240863 \times 10^{-7}$
Observations	240

time. The R^2 in the Table explains that the execution time values can be 99.64% explained using the given workload dimension. Adjusted R^2 ensures that the predictor (workload sizes as independent variable) is an appropriate regressor. The results of m and j values coefficients are determined by the linear regression line fitting as $2.02212695240863 \times 10^{-7}$ and -3.315317243 respectively. Therefore, the execution time model of the matrix

Table 5.9: Validation of Local Execution Time Statistical Model using Sample Split Approach

Sample Partition 1		Sample Partition 2		Original Sample	
Multiple R	0.9981812	Multiple R	0.998279439	Multiple R	0.998221727
R Square	0.996365708	R Square	0.996561838	R Square	0.996446615
Adjusted R Square	0.99634256	Adjusted R Square	0.996518317	Adjusted R Square	0.996431685
df	158	df	80	df	239
Observations	159	Observations	81	Observations	240

multiplication in local execution mode is

$$T_{local}(W_i) = (2.02212695240863 \times 10^{-7} \times c_i) - 3.315317243 \quad (5.12)$$

To validate our devised model, we used the split-sample procedure. We split our sample into two randomly selected partitions. For each partition, we determine the correlation coefficients and compare the results of both partitions with the unpartitioned sample to ensure validity. The results of our comparison are presented in Table 5.9. As the results show, the model produces identical R , R^2 , and adjusted R^2 for all three different samples. The degree of freedom (df) and the observation rows in the table shows that the size of each partitioned sample is unique and random. The adjusted R^2 is similar for all the groups which are evidence of the validity of our proposed statistical model of local execution time.

5.3.2.2 Statistical modeling of execution time in PMCO execution mode

PMCO is the second execution mode, is when the local execution is offloaded using the proposed process migration based computational offloading method to the remote device. The execution time in PMCO mode is being influenced by the execution time on the server device, communication time, and overhead in terms of time consumed on checkpointing and restarting the process on both the client device and the server device. $T_{PMCO}^{MatMult}(W_i)$ is the maximum time that the entire cycle of the i^{th} matrix multiplication workload takes

for execution in PMCO mode and is:

$$T_{PMCO}(W_i) = 2 \times (T_{checkpoint}(W_i) + T_{restart}(W_i)) + T_{transmission}(W_i) + T_{remote}(W_i) \quad (5.13)$$

Where $T_{checkpoint}$, is the time it takes to checkpoint the i^{th} workload process on local device and $T_{restart}$ is the time it takes to restart the i^{th} workload from a checkpoint file on the local device. It is multiplied by two because the same process happens on the remote side also. The $T_{checkpoint}$ and $T_{restart}$ time on the server device can be lower than the client devices due to the difference in the compute power. However, we have calculated this turnaround time checkpoint and restore time and fixed the value to 0.5. $T_{transfer}$ and $T_{receive}$ is the transfer and receive time for a checkpoint file from local device to the remote server and vice versa and off-course this time strictly depends upon the network dynamics. Lastly, T_{remote} is the execution of the offloaded i^{th} workload. The execution for the PMCO based execution presented in Table 5.7 and next chapter are calculated as:

$$T_{PMCO}(W_i) = 0.5 + (((6.97687 \times 10^{-8} \times c_i) + 0.064535721) + ((8.46526 \times 10^{-8} \times d_i) + 0.1306192)) + T_{remote}(W_i) \quad (5.14)$$

In contrast with the execution time in local execution mode that depends on the computing power of the host mobile device. The execution time in PMCO execution mode strongly depends on the computing capabilities of the remote servers which are not identical even if the computing specifications are the same (due to existing heterogeneity); in addition with underlying network dynamics. In equation (5.14), the most computational expensive is the server side matrix multiplication process. Hence, the input to the linear regression is the complexity of the matrix multiplication algorithm and their workloads. As described in local execution mode, the complexity of matrix multiply application is

$O(N^3)$. So the size of the matrices that are sent for execution to the remote server using PMCO, called g_i , are used to measure the application complexity.

$$T_{remote}(W_i) = (m \times g_i) + j \quad (5.15)$$

To accurately estimate the m and j values, the server-side measured execution time of the matrix multiplication execution in PMCO mode are used to draw the linear regression.

Before we perform regression modeling, we depict the linearity of matrix multiplication and PMCO execution time. We have depicted the scatter diagram for matrix sizes of workloads and corresponding server side execution times in Figure 5.7. As the results show, the relationship between the workload sizes and respective execution time is linear. Thus the linear regression is feasible to model this relationship.

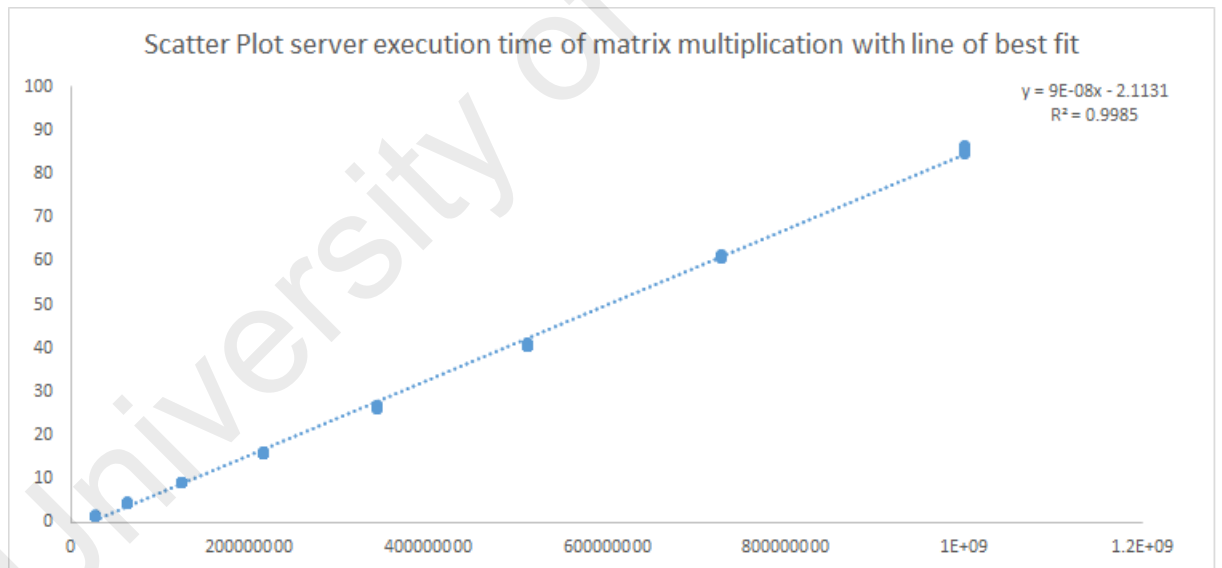


Figure 5.7: Scatter plot of server side matrix multiplication execution time, with the linearity correlation determined by the line of best fit.

The results of the linear regression are given in Table 5.10. The R value in Table testifies 99.99% correlation between the matrix workload size value and its corresponding server side execution time. The R^2 explains that 99.85 of the PMCO execution response time values can be explained using the given matrix workload sizes. The F and $Sig.$

Table 5.10: Regression Statistics Summary for server side execution time of matrix multiplication

Multiple R	0.999249279
R Square	0.998499121
Adjusted R Square	0.998492815
F	158335.7415
Significance F	0
Intercept	-2.113132585
X Variable 1	8.67706×10^{-8}
Observations	240

values in the Table show significant direct correlations between the workload size and the PMCO execution. These results enable us to leverage linear regression to derive our statistical model for the PMCO execution time of the matrix multiplication application. The linear regression analysis determines the coefficients values for the waiting time of the multiply as $m = 8.67706 \times 10^{-8}$ and $j = -2.113132585$. Hence, the statistical model of execution response time for the PMCO execution time is written as:

$$\begin{aligned}
 T_{PMCO}(W_i) = & 0.5 + (((6.97687 \times 10^{-8} \times c_i) + 0.064535721) + \\
 & ((8.46526 \times 10^{-8} \times d_i) + 0.1306192)) \\
 & + ((8.67706 \times 10^{-8} \times g_i) - 2.113132585)
 \end{aligned} \tag{5.16}$$

To validate our devised model, we previously validated our devised model for data transmission model. Now, we will validate the server side execution time to validate the whole PMCO model presented in equation (5.16). Similarly, like the other devised model we used the split-sample procedure. We split our sample into two randomly selected partitions. For each partition, we determine the correlation coefficients and compare the results of both partitions with the un-partitioned original sample to ensure validity. The results of our comparison are presented in Table 5.11. As the results show, the model produces identical R , R^2 , and adjusted R^2 for all three different samples. The degree of freedom (df) and the observation rows in the table shows that the size of each partitioned sample is unique and random. The adjusted R^2 is similar for all the groups which is an

Table 5.11: Validation of server side execution time statistical model using sample split approach

Sample Partition 1		Sample Partition 2		Original Sample	
Multiple R	0.99928393	Multiple R	0.999169677	Multiple R	0.999249279
R Square	0.998568372	R Square	0.998340043	R Square	0.998499121
Adjusted R Square	0.998559253	Adjusted R Square	0.998319031	Adjusted R Square	0.998492815
df	158	df	80	df	239
Observations	159	Observations	81	Observations	240

evidence of the validity of our proposed statistical model of server side execution time, which in turn validate the model presented in equation (5.16).

5.3.3 Energy consumed

To analyze the energy consumption of the experimental workloads, each workload is executed thirty times in all of the execution modes. Table 5.12, represents the mean of energy consumption of thirty observations of each workload along with error margin according to 95% confidence interval while executing the workload on the local mobile device, Local_PMCO, and using PMCO to the remote computing devices. The granularity of the matrix multiplication indicates the computational intensity which varies from 300x300 to 1000x1000. The overhead column in Table 5.12, presents the energy overhead i.e. difference between Local_PMCO and local, when the application is executed locally with the checkpoint thread injected. The difference is ranging from 0.01% to 5.7634% compared to the original energy consumption of the workload. Table 5.12 also presents the benefit percentage in energy consumption from the local execution and execution using PMCO. The energy consumption benefit based upon descriptive statistics is almost about from 53% to 91% on average.

The second statistics method is the regression analysis of the energy consumed by the experimental workloads. As already discussed in Section in 5.2.4, the regression analysis can be done only for the matrix multiplication workload as it has a correlation and has the data for different computational intensities to verify the behavior for unknown data

Table 5.12: Workload Energy Consumption in Joules with 95% confidence interval, Using three execution modes

Workload #	Workload	Local	Local_PMCO	Overhead %	Process Migration	Benefit %
1	Dhrystone	1.46 ± 0.01	1.46 ± 0.01	0.108937057	0.68 ± 0.03	53.12534436
2	Whetstone	6.81 ± 0.13	7.2 ± 0.05	5.763452626	2.37 ± 0.14	65.26221583
3	Linpack	6.24 ± 0.05	6.44 ± 0.06	3.196366752	2.15 ± 0.06	65.55407922
4	Scimark-FFT	4.97 ± 0.03	4.97 ± 0.03	0.069632963	0.35 ± 0.01	92.90256872
5	Scimark-SOR	2.67 ± 0.01	2.8 ± 0.01	4.555613101	0.66 ± 0.06	75.38408935
6	Scimark-MonteCarlo	2.28 ± 0	2.28 ± 0.01	0.309088031	0.78 ± 0.02	65.56103235
7	Scimark-SparseMatMult	2.06 ± 0.01	2.07 ± 0.01	0.301181829	0.67 ± 0.02	67.45402674
8	Scimark-LU	7.83 ± 0.11	8.18 ± 0.11	4.38841816	0.69 ± 0.02	91.20458107
9	Mat. Mult. (300x300)	1.27 ± 0.02	1.33 ± 0.01	4.88273489	0.26 ± 0.01	79.76313523
10	Mat. Mult. (400x400)	3.65 ± 0.03	3.85 ± 0.03	5.418546485	0.65 ± 0.02	82.21512205
11	Mat. Mult. (500x500)	7.68 ± 0.09	7.99 ± 0.09	4.044797353	1.17 ± 0.04	84.74104488
12	Mat. Mult. (600x600)	7.1 ± 0.01	7.11 ± 0.02	0.060987027	1.13 ± 0.01	84.06025846
13	Mat. Mult. (700x700)	12.17 ± 0.03	12.23 ± 0.06	0.460964246	1.95 ± 0.03	84.01468844
14	Mat. Mult. (800x800)	21.68 ± 0.01	21.78 ± 0.02	0.447504338	3.02 ± 0.02	86.0528725
15	Mat. Mult. (900x900)	28.4 ± 0.06	28.4 ± 0.06	0.010991418	4.7 ± 0.04	83.45021109
16	Mat. Mult. (1000x1000)	40.33 ± 0.09	40.48 ± 0.09	0.362594848	6.33 ± 0.03	84.31202949

points.

5.3.3.1 Statistical modeling of energy consumed in local execution mode

The energy consumption of the mobile device running a matrix multiplication workload mainly comprises of the total energy used by the CPU as obviously there will not be any wireless communication. Hence, the only power consuming components is CPU. If E_m^i is the mean energy consumed for the local execution of the i^{th} workload, therefore we have

$$E_m(W_i) = P_m \times (CPU_{W_i}) \quad (5.17)$$

where $E_m(W_i)$ is the total CPU energy consumed to execute the entire i^{th} workload locally. While P_m is power rating of the CPU when active.

CPU power consumption for each computational workload highly depends on the computational intensity of the workload. However, the computational intensity of the workload is also correlated with the execution time. So, the intenser is the workload, the higher will be the execution time, and the more will be CPU power consumption.

Because of significant dependency of the consumed energy to the execution time, we study the consumed energy of a workload as whole regardless of the energy consumption

of each component. Therefore, we consider the workload in whole for presenting energy model of our model.

Similar to the statistical model of execution time, to present a reliable and accurate estimation model of the CPU energy, we perform linear regression using measured real data on the mobile device. We use datasets of workloads including the execution time and energy consumption of each workload and use them for training the regression model to produce the energy model. For validation of our proposed model, we use the split sample approach. Hence, the power model can be presented as a function of execution time written as:

$$E_m(W_i) = (z \times T_{local}(W_i)) + c \quad (5.18)$$

where $T_{local}(W_i)$ is the total execution time for the i^{th} workload and c is a constant value. Both z and c values can be determined using training over linear regression.

Before regression analysis, the type of regression should be identified whether it is linear or non-linear. The scatter diagram for the local energy consumption is plotted in Figure 5.8. As the results show, the relationship between the energy and respective execution time of workloads is linear. Thus, we perform a linear regression to model this relationship and derive the statistical model.

The detail statistics of the statistical model of our linear regression are summarized in Table 5.13. The R value shows a significant correlation between the execution time and consumed energy. The R^2 value in the Table testifies that 99.41% of the energy values can be explained using execution time due to significant direct correlation adjusted R^2 advocates that the predictor (time) is an appropriate regressor to model energy. The F and $SignificanceF$ values in the Table ensure that available dataset is appropriate to be used for linear regression. Hence, performing linear regression is applicable to our model

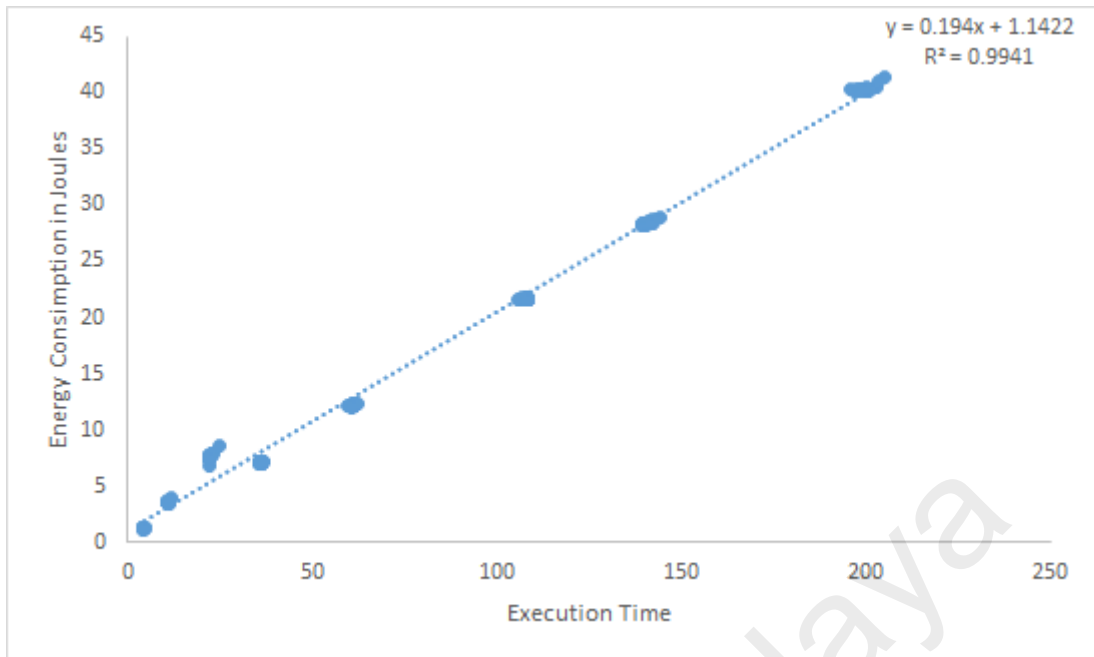


Figure 5.8: Scatter plot showing linearity correlation between local execution time and consumed energy.

Table 5.13: Regression Statistics Summary for energy consumption in local execution mode

Multiple R	0.997052147
R Square	0.994112985
Adjusted R Square	0.994088249
F	40189.95571
Significance F	2.1478^{-267}
Intercept	1.14220578
X Variable 1	0.193971484
Observations	240

and derived model is reliable and beneficial.

The coefficient values for the regression model are $z = 0.193971484$ and $c = 1.14220578$.

Hence, Equation (5.18) is

$$E_m(W_i) = (0.193971484 \times T_{local}(W_i)) + 1.14220578 \quad (5.19)$$

Validation of the devised energy model of local execution mode is carried out using the split-sample procedure. The sample is divided into two randomly selected partitions. For each partition, we determine the correlation coefficients and compare the results of

Table 5.14: Validation of local device energy consumption model using sample split approach

Sample Partition 1		Sample Partition 2		Original Sample	
Multiple R	0.996883776	Multiple R	0.997325672	Multiple R	0.997052147
R Square	0.993777262	R Square	0.994658497	R Square	0.994112985
Adjusted R Square	0.993735217	Adjusted R Square	0.994597798	Adjusted R Square	0.994088249
df	149	df	89	df	239
Observations	150	Observations	90	Observations	240

both partitions with the full sample to demonstrate validity. The results of our comparison are presented in Table 5.14. As the results show, the model produces identical R , R^2 , and adjusted R^2 for all three different samples. The degree of freedom (df) column in the table shows that the size of each sample is uniquely accidental. The adjusted R^2 is identical for all the splits which is an evidence of the validity of our proposed statistical model.

5.3.3.2 Statistical modeling of energy consumed in PMCO execution mode

In the proposed PMCO execution mode CPU and communication energy are two major energy consumers who will be considered for devising the energy model. Hence, if $E_{PMCO}(W_i)$ is the total energy consumed for execution in PMCO of the i^{th} workload, therefore we have

$$E_{PMCO}(W_i) = E_m^{checkpoint}(W_i) + E_m^{restart}(W_i) + E_t(W_i) + E_r(W_i) \quad (5.20)$$

where, $E_m^{checkpoint}(W_i)$, and $E_m^{restart}(W_i)$ is the energy consumption it takes a checkpoint and restarts a process, while $E_t(W_i)$ and $E_r(W_i)$, is the amount of energy spend while transmitting and receiving the checkpoint image. Similarly, as discussed in the energy consumption of the application execution on the local mobile device, the consumption of energy is a function of activity time. The more, the longer an activity is performing, the higher the energy consumption will be. Furthermore, PowerAPI allows us to collect the energy consumption of the all the processes in equation (5.20) collectively, so it would be much more logical to model the energy consumption as a function of total PMCO ac-

tivity time. Hence, the power equation (5.20) can be described as a function of PMCO execution response:

$$E_{PMCO}(W_i) = P_m \times T_{PMCO}(W_i) \quad (5.21)$$

where P_m is power rating, while PMCO activity is ongoing. Hence the linear model can be re-written as:

$$E_{PMCO}(W_i) = (z \times T_{PMCO}(W_i)) + c \quad (5.22)$$

where $T_{PMCO}(W_i)$ is the total execution time for the i^{th} workload and c is a constant value. Both z and c values can be determined using training over linear regression.

Prior to the regression analysis, the type of regression should be identified whether it is linear or non-linear. The scatter diagram for the energy consumption in PMCO execution mode is plotted in Figure 5.9. As the results show, the relationship between the application energy consumption and its respective PMCO execution response time of is linear. Thus, we perform a linear regression to model this relationship and derive the statistical model.

The detail statistics of the statistical model of our linear regression are summarized in Table 5.15. The R value shows a significant correlation between the execution time and consumed energy. The R^2 value in the Table testifies that 99.15% of the energy values can be explained using execution time due to significant direct correlation adjusted R^2 advocates that the predictor (time) is an appropriate regressor to model energy. The F and $SignificanceF$ values in the Table ensure that available dataset is appropriate to be used for linear regression. Hence, performing linear regression applies to our model, and the derived model is reliable and beneficial.

The coefficient values for the regression model are $z = 0.070284574$ and $c = 0.164839255$.

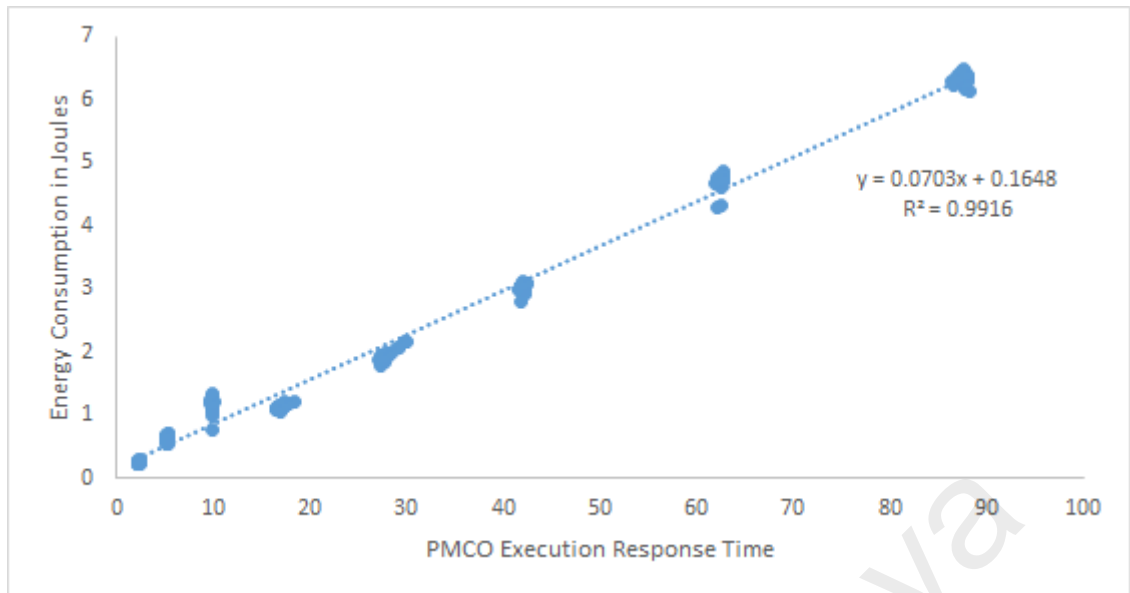


Figure 5.9: Scatter plot showing linearity correlation between PMCO execution response time and consumed energy.

Table 5.15: Regression Statistics Summary for energy consumption in PMCO execution mode

Multiple R	0.995785437
R Square	0.991588636
Adjusted R Square	0.991553294
F	28057.05329
Significance F	5.9446×10^{-249}
Intercept	0.164839255
X Variable 1	0.070284574
Observations	240

Hence, Equation (5.22) is:

$$E_{PMCO}(W_i) = (0.070284574 \times T_{PMCO}(W_i)) + 0.164839255 \quad (5.23)$$

Validation of the devised energy model of local execution mode is carried out using the split-sample procedure. The sample is divided into two randomly selected partitions. For each partition, we determine the correlation coefficients and compare the results of both partitions with the full sample to demonstrate validity. The results of our comparison are presented in Table 5.16. As the results show, the model produces identical R , R^2 , and adjusted R^2 for all three different samples. The degree of freedom (df) column in the table

Table 5.16: Validation of PMCO energy consumption model using sample split approach

Sample Partition 1		Sample Partition 2		Original Sample	
Multiple R	0.995594996	Multiple R	0.996160946	Multiple R	0.995785437
R Square	0.991209395	R Square	0.992336631	R Square	0.991588636
Adjusted R Square	0.991148349	Adjusted R Square	0.992253333	Adjusted R Square	0.991553294
df	145	df	93	df	239
Observations	146	Observations	94	Observations	240

shows that the size of each sample is uniquely accidental. The adjusted R^2 is identical for all the splits which is an evidence of the validity of our proposed statistical model.

5.3.4 Compute power

To analyze the impact and improvement in the compute power of the client device based on the experimental setup explained in section 5.1, we have utilized the standard CPU performance benchmarks also explained in section 5.1. The selected benchmarking applications are executed in three execution modes of local and Local_PMCO, and PMCO for performance evaluation of the proposed framework in terms of MFLOPS, MIPS, execution time, data transfer amount and consumed energy. In the local execution mode, we execute the entire task locally on the mobile device without utilizing remote resources. In the Local_PMCO, we again execute the entire task locally on the mobile device without utilizing remote resources, but this time the program is executed using the checkpoint launcher discussed in 4.1.2, which will in-turn inject the checkpoint thread into it. While the last execution mode is the proposed offloaded execution in which once the application execution reach a migration marker the process is migrated to a nearby remote computing infrastructure available on one hop. The schematic presentation of our benchmarking setup is illustrated in Figure 5.1.

Each of the benchmark workload is executed thirty times in all of the execution modes. Table 5.17, represents the mean of compute power (MIPS, MWIPS or MFLOPS) of thirty observations of each workload along with error margin according to 95% confidence interval while executing the workload on the local mobile device, Local_PMCO,

Table 5.17: Benchmark workloads descriptive statistics depicting the improvement in the compute power

Workload #	Workload	Local	Local_PMCO	Deterioration %	PMCO	Improvement %
1	Dhrystone (MIPS)	372.24 ± 2.63	365.02 ± 2.75	1.94	1326.52 ± 44.7	256.3614872
2	Whetstone (MWIPS)	171.22 ± 0.55	166.71 ± 1.3	2.63	665.63 ± 10.79	288.7571545
3	Linpack (MFLOPS)	36.62 ± 0.33	35.74 ± 0.26	2.40	148.64 ± 0.72	305.8984162
4	Scimark-FFT (MFLOPS)	8.73 ± 0.055	8.72 ± 0.03	0.11	40.86 ± 0.45	368.0412371
5	Scimark-SOR (MFLOPS)	53.24 ± 0.042	53.24 ± 0.04	0.00	174.05 ± 5.88	226.9158527
6	Scimark-MonteCarlo (MFLOPS)	10.73 ± 0.005	10.72 ± 0.01	0.09	38.19 ± 0.79	255.917987
7	Scimark-SparseMatMult (MFLOPS)	23.22 ± 0.018	23.22 ± 0.02	0.00	86.1 ± 2.96	270.8010336
8	Scimark-LU (MFLOPS)	33.95 ± 0.018	33.95 ± 0.01	0.00	110.42 ± 2.02	225.2430044

and using PMCO to the remote computing devices.

The deterioration, column in Table 5.17 represents the percentage of the deterioration in the compute power of the client device when the benchmark is executed locally through the checkpoint launcher of the AMC. The deterioration percentage is calculated using the following formula.

$$D_m^i = (CP_m^i - \bar{CP}_m^i) \times 100 \quad (5.24)$$

where D_m^i is the deterioration in compute power observed by the i^{th} workload, CP_m^i is the compute power observed by the i^{th} workload on the local device, and \bar{CP}_m^i is the compute power observed by the i^{th} workload on the local device while the workload process is also accompanied by the checkpoint thread. The deterioration is observed to be around 0 to 2.63% for the selected benchmark workloads. Similarly, to the deterioration Table 5.17 also reports the improvement in compute power with PMCO when compared with the local compute power. The improvement percentage is the actually the percentage of the difference between the compute power of PMCO and local compute power. It is observed that compute power is increased almost about 200% to 300% for corresponding benchmark workloads; this improvement is directly proportional to the compute power of the server device, the more it is powerful, the more compute power the client can draw from it.

5.4 Conclusion

In this chapter, we describe the evaluation procedure in two parts of descriptive and inferential statistics. In each section, the detailed description of data generation process for compute power, data transfer, execution time and consumed energy are described and evaluated using statistical t-test inference method. Furthermore, we employed statistical modeling and used observation-based analysis focusing on independent replication method to devise the models for execution time and energy consumption in both local and PMCO execution modes. The devised models are validated through the split-sample approach, and the results of validation are reported. The results of performance evaluation are presented in next chapter that will be used to signify the strength and weaknesses our proposed framework.

CHAPTER 6: RESULTS AND DISCUSSION

In this chapter, we present results of our performance evaluation of the proposed model by analyzing system-level metrics, namely compute power, execution time and energy consumption of the device for execution of the experimental applications using series of experiments. The evaluation results are validated via statistical modeling built using independent replication of new dataset.

The remainder of this chapter is as follows. Section 6.1 presents our experimental results, and reports compute power of the experimental setup, impact on application execution time and energy consumption of the experimental workloads in local and process migration based computational offloading (PMCO) execution modes. The results of our statistical modeling are presented in section 6.1 to validate the empirical data. Comparative evaluations are presented in section 6.2, and finally, the chapter is concluded in section 6.3.

6.1 Performance evaluation results

Results of performance evaluation generated via experimental analysis are presented in this section in three parts. In the first part, data related to compute power, execution time analysis and consumed energy analysis are presented in part two and three respectively. The experimental analysis is performed to evaluate the performance of the proposed framework.

6.1.1 Execution time

This section presents temporal results of executing the experimental application of matrix multiplication in three execution environments. One environment is local in which the entire application, including intensive and non-intensive are executed on the mobile device. Whereas in the second environment which is Local_PMCO, the experimental ap-

plication of matrix multiplication is executed locally, through the environment of PMCO. While the last environment is PMCO, in which the experimental matrix multiplication application execution starts locally and then once the execution reaches a migration marker the application is offloaded to performed remotely using any of mobile cloud resource. Data related to execution in this section are gathered using experimental analysis. Several Tables and charts are used to demonstrate the findings.

Tables 6.1, 6.2 and 6.3 presents the temporal data related to application execution time collected in the local environment, in Local_PMCO and PMCO execution environments, respectively. Each of the tables presents mean execution time, standard deviation, error estimate, and execution time of eight workloads in eight intensity levels with 95% confidence interval.

The small error estimates based on the 95% confidence interval shown in the Tables 6.1, 6.2 and 6.3 ensure the reliability of the collected data during the experimentation process. For example, the maximum error estimate for the 600x600 matrix multiplication workload executed in PMCO execution time with 95% confidence interval is 17.02 ± 0.14 seconds, it can be interpreted as the PMCO execution time of the workload falls in the range of $(17.02 - 0.14) < \mu < (17.02 + 0.14)$. This inequality range presents the fact that if the experiment is repeated for that workload, the execution time value will fall in between this range with 95% confidence. To better demonstrate the significance of our achievements and efficiently interpret the results, we perform a comprehensive statistical analysis which is presented as follows.

Descriptive statistics of results in local and PMCO execution modes, including minimum, maximum, and mean execution time of the matrix multiplication workloads are summarized in Table 6.4 in eight intensity levels beside the mean execution time of all granularity levels. As descriptive statistics in the Table shows, executing the task on remote computing infrastructure instead of local device can reduce the execution time as

Table 6.1: Execution Time with 95% Confidence Interval in Local Execution Mode Generated via Experiments

Execution Trace	Matrix Multiplication Granularity							
	300x300	400x400	500x500	600x600	700x700	800x800	900x900	1000x1000
	Execution Time (in seconds)							
1	4.29	11.09	22.06	36.25	62.22	107.81	142.34	202.68
2	4.32	10.98	22.11	36.28	60.71	107.15	140.42	198.90
3	4.37	11.00	22.15	36.18	60.81	106.09	141.65	197.00
4	4.34	11.08	22.28	35.98	61.31	108.44	140.40	199.03
5	4.51	11.25	22.95	36.15	61.01	108.40	140.13	197.74
6	4.43	10.92	22.68	35.72	61.09	106.99	140.47	200.79
7	4.31	11.04	22.35	35.83	60.76	106.60	139.56	200.33
8	4.30	11.13	22.33	36.07	61.48	108.29	141.21	200.46
9	4.29	11.00	22.46	36.09	59.82	106.47	141.86	200.20
10	4.39	11.13	22.71	36.04	61.18	108.37	142.37	198.24
11	4.32	11.14	22.05	35.97	61.03	107.86	141.07	198.15
12	4.34	11.17	22.11	36.03	60.98	107.78	140.31	195.59
13	4.57	10.87	22.20	35.90	61.26	107.61	140.58	199.75
14	4.43	10.97	22.14	35.98	60.71	108.04	139.40	199.48
15	4.30	11.08	22.21	36.00	61.67	107.80	141.76	200.47
16	4.36	10.91	22.46	35.97	60.50	107.38	144.03	205.09
17	4.34	10.87	22.64	36.02	60.06	108.28	139.40	198.09
18	4.33	11.03	22.08	35.98	61.76	108.56	141.26	200.55
19	4.39	11.00	22.17	36.42	60.98	108.31	140.18	198.38
20	4.34	10.87	22.27	36.36	60.49	107.75	140.27	199.04
21	4.46	10.98	22.12	36.19	60.86	108.23	141.61	200.57
22	4.49	11.06	22.20	36.29	61.24	108.48	139.53	200.84
23	4.34	10.88	22.20	36.08	61.05	108.10	141.87	198.68
24	4.31	10.85	22.44	36.13	60.82	107.51	142.34	199.32
25	4.29	10.98	22.07	36.42	59.99	106.99	139.40	200.15
26	4.32	11.15	22.20	35.99	61.47	105.78	140.40	200.05
27	4.31	10.84	22.30	35.80	61.09	107.05	141.57	201.26
28	4.46	10.99	24.65	36.26	60.36	108.19	143.23	203.28
29	4.41	11.09	22.06	36.69	60.84	108.65	141.87	198.88
30	4.34	11.92	22.16	36.71	60.92	108.11	141.58	199.05
Min	4.29	10.84	22.05	35.72	59.82	105.78	139.40	195.59
Mean	4.37	11.04	22.36	36.13	60.95	107.70	141.07	199.73
Median	4.34	11.00	22.20	36.08	60.98	107.83	141.14	199.61
Maximum	4.57	11.92	24.65	36.71	62.22	108.65	144.03	205.09
Std. Deviation	0.07	0.20	0.49	0.23	0.52	0.76	1.17	1.86
Confidence Int.	0.03	0.07	0.17	0.08	0.18	0.27	0.42	0.66

Table 6.2: Execution Time with 95% Confidence Interval in Local_PMCO Execution Mode Generated via Experiments

Execution Trace	Matrix Multiplication Granularity							
	300x300	400x400	500x500	600x600	700x700	800x800	900x900	1000x1000
	Execution Time (in seconds)							
1	4.38	11.66	22.77	36.20	62.65	109.76	141.58	199.52
2	4.36	11.46	22.60	36.11	60.83	108.44	141.07	200.32
3	4.47	11.16	22.75	36.43	60.90	108.91	142.19	201.50
4	4.37	11.22	24.00	36.69	62.73	108.90	141.14	201.10
5	4.43	11.29	22.87	36.34	61.12	109.29	141.06	201.83
6	4.41	11.04	22.88	37.42	62.12	108.83	141.64	203.88
7	4.40	11.79	22.72	36.18	62.28	108.56	141.16	203.02
8	4.37	11.17	22.50	36.15	61.48	108.90	141.75	201.37
9	4.39	11.42	22.50	36.47	65.10	109.47	141.80	200.66
10	4.45	11.17	22.49	36.39	60.73	108.76	141.83	200.10
11	4.39	11.86	22.55	36.29	60.68	109.50	142.34	200.13
12	4.56	11.30	22.71	36.43	61.78	108.86	141.59	201.04
13	4.60	11.16	22.59	35.82	61.46	107.80	141.62	203.11
14	4.56	11.26	22.46	36.60	60.84	107.98	143.29	199.54
15	4.62	11.25	22.56	36.51	60.82	108.40	141.42	198.44
16	4.60	11.11	22.52	35.99	61.39	109.02	142.78	199.69
17	4.56	11.48	22.47	36.33	61.68	109.21	142.11	200.48
18	4.32	11.26	22.53	36.03	61.47	108.94	143.97	201.17
19	4.37	11.20	22.61	36.26	61.92	108.85	144.01	200.92
20	4.38	11.37	22.98	36.98	60.05	107.80	139.04	200.22
21	4.37	11.38	22.39	36.82	61.07	108.50	142.46	201.14
22	4.36	11.39	22.47	35.95	61.64	107.97	142.41	203.11
23	4.64	11.31	22.49	36.23	60.83	108.38	140.59	203.04
24	4.57	12.03	22.53	36.16	62.84	108.97	140.75	202.96
25	4.61	11.65	22.57	36.24	62.18	107.75	141.38	197.91
26	4.62	11.27	22.69	35.43	61.46	108.52	141.27	199.12
27	4.46	11.39	22.91	36.43	61.61	109.70	140.87	198.72
28	4.32	11.44	22.56	36.43	62.44	109.56	142.10	201.66
29	4.41	11.27	25.36	35.83	64.39	108.99	141.62	200.43
30	4.53	11.09	22.62	36.12	62.52	109.17	143.01	201.70
Min	4.32	11.04	22.39	35.43	60.05	107.75	139.04	197.91
Mean	4.46	11.36	22.75	36.31	61.77	108.79	141.80	200.93
Median	4.42	11.29	22.58	36.27	61.54	108.88	141.63	200.98
Maximum	4.64	12.03	25.36	37.42	65.10	109.76	144.01	203.88
Std. Deviation	0.10	0.23	0.57	0.37	1.07	0.56	1.01	1.50
Confidence Int.	0.04	0.08	0.20	0.13	0.38	0.20	0.36	0.54

Table 6.3: Execution Time with 95% Confidence Interval in PMCO Execution Mode Generated via Experiments

Execution Trace	Matrix Multiplication Granularity							
	300x300	400x400	500x500	600x600	700x700	800x800	900x900	1000x1000
Execution Time (in seconds)								
1	2.35	5.42	9.76	16.94	27.12	41.76	62.38	87.95
2	2.32	5.22	10.00	17.44	27.32	42.01	62.58	87.99
3	2.24	5.28	9.99	17.08	29.18	42.14	62.36	87.75
4	2.28	5.20	9.90	16.81	27.36	42.00	62.55	87.04
5	2.28	5.41	9.80	16.86	29.75	41.82	62.68	88.17
6	2.35	5.25	9.88	16.93	29.24	41.80	62.57	87.51
7	2.36	5.16	9.90	17.03	27.77	42.07	62.48	87.86
8	2.32	5.44	9.75	16.61	28.50	42.12	62.73	87.42
9	2.35	5.22	9.89	18.34	28.16	41.65	62.41	87.59
10	2.35	5.18	9.87	16.86	27.33	42.10	62.55	87.75
11	2.41	5.22	9.91	17.27	27.39	41.96	62.32	87.75
12	2.32	5.27	9.96	16.83	27.30	42.18	62.19	87.73
13	2.45	5.37	9.93	16.90	27.82	42.34	62.48	86.34
14	2.40	5.21	9.92	16.66	27.62	42.12	62.54	87.64
15	2.23	5.22	9.97	16.81	27.96	42.07	62.68	87.91
16	2.26	5.11	9.86	16.88	28.02	41.94	62.81	88.08
17	2.20	5.14	9.86	16.65	29.97	42.06	62.32	86.54
18	2.16	5.24	9.95	16.81	28.07	41.77	62.41	87.58
19	2.33	5.18	9.93	17.12	28.09	42.36	62.25	87.55
20	2.42	5.13	9.91	16.82	27.68	41.98	62.47	87.80
21	2.22	5.26	9.99	17.45	27.33	41.94	62.75	87.58
22	2.23	5.34	9.95	16.65	28.14	41.85	62.38	87.73
23	2.28	5.11	9.95	16.69	27.42	41.90	62.02	88.10
24	2.45	5.37	9.88	17.89	27.81	41.93	62.24	87.52
25	2.26	5.33	9.90	16.69	27.53	42.26	62.67	87.64
26	2.41	5.33	9.86	17.42	27.36	41.62	62.14	87.70
27	2.27	5.10	9.83	16.63	27.64	42.17	62.77	87.88
28	2.34	5.21	10.10	17.30	27.82	41.84	62.61	87.76
29	2.30	5.18	9.92	17.34	27.83	42.49	62.67	87.73
30	2.40	5.11	9.92	16.73	27.48	42.01	62.67	86.72
Min	2.16	5.10	9.75	16.61	27.12	41.62	62.02	86.34
Mean	2.32	5.24	9.91	17.02	27.93	42.01	62.49	87.61
Median	2.32	5.22	9.91	16.87	27.79	42.00	62.51	87.73
Maximum	2.45	5.44	10.10	18.34	29.97	42.49	62.81	88.17
Std. Deviation	0.08	0.10	0.07	0.40	0.73	0.20	0.20	0.43
Confidence Int.	0.03	0.04	0.03	0.14	0.26	0.07	0.07	0.15

Table 6.4: Descriptive Statistics of Execution Time Data Generated by standard experimentation

		Min	Mean	Median	Maximum	Std. Deviation	Confidence Int.
Mat. Mult. (300x300)	Local	4.29	4.37	4.34	4.57	0.07	0.03
	Local_PMCO	4.32	4.46	4.42	4.64	0.10	0.04
	PMCO	2.16	2.32	2.32	2.45	0.08	0.03
Mat. Mult. (400x400)	Local	10.84	11.04	11.00	11.92	0.20	0.07
	Local_PMCO	11.04	11.36	11.29	12.03	0.23	0.08
	PMCO	5.10	5.24	5.22	5.44	0.10	0.04
Mat. Mult. (500x500)	Local	22.05	22.36	22.20	24.65	0.49	0.17
	Local_PMCO	22.39	22.75	22.58	25.36	0.57	0.20
	PMCO	9.75	9.91	9.91	10.10	0.07	0.03
Mat. Mult. (600x600)	Local	35.72	36.13	36.08	36.71	0.23	0.08
	Local_PMCO	35.43	36.31	36.27	37.42	0.37	0.13
	PMCO	16.61	17.02	16.87	18.34	0.40	0.14
Mat. Mult. (700x700)	Local	59.82	60.95	60.98	62.22	0.52	0.18
	Local_PMCO	60.05	61.77	61.54	65.10	1.07	0.38
	PMCO	27.12	27.93	27.79	29.97	0.73	0.26
Mat. Mult. (800x800)	Local	105.78	107.70	107.83	108.65	0.76	0.27
	Local_PMCO	107.75	108.79	108.88	109.76	0.56	0.20
	PMCO	41.62	42.01	42.00	42.49	0.20	0.07
Mat. Mult. (900x900)	Local	139.40	141.07	141.14	144.03	1.17	0.42
	Local_PMCO	139.04	141.80	141.63	144.01	1.01	0.36
	PMCO	62.02	62.49	62.51	62.81	0.20	0.07
Mat. Mult. (1000x1000)	Local	195.59	199.73	199.61	205.09	1.86	0.66
	Local_PMCO	197.91	200.93	200.98	203.88	1.50	0.54
	PMCO	86.34	87.61	87.73	88.17	0.43	0.15

significant as 53%.

The improvement in execution time is not a function of intensity levels of the workload but is a function of compute power of the remote server and the underlying network conditions. The mean execution time savings are as high as 53%, 47%, 44%, 47%, 45%, 39%, 44%, 43% according to the increasing order of the workload intensities. As it is evident from this, and also discussed in the previous chapter that the data transmission time plays a significant role in the overall execution response time of the workload. Similarly, from the application execution time overhead point of view when the workloads are executed locally through the PMCO framework the degradation in increasing order of workload intensities are 2.16%, 2.81%, 1.73%, 0.50%, 1.32%, 0.99%, 0.51%, 0.59%. The degradation is not that much and is as low as 0.50% and goes up to 2.81%. Still, we could not find a correlation between the intensity level and the percentage of degradation as it is influenced by the internal operating system process/thread context switching which can

periodically or upon interrupts traverse to the checkpoint thread.

The mean execution time of workloads in local execution mode is as much as 1.88, 2.10, 2.25, 2.12, 2.18, 2.56, 2.25, 2.27 times more than PMCO execution according to the increasing intensity level which is remarkable despite the workloads. As described in the previous chapter, execution of each workload is repeated thirty times to enhance the reliability of performance evaluation. So, data plotted in Figure 6.1 are mean execution time of the workloads for Local, Local_PMCO and PMCO execution modes. Each diagonal bricks bar in Figure 6.1, represents the mean value of execution time measured using PMCO mode of thirty iterations for each corresponding matrix multiplication workload. Similarly, each diagonal strips bar represents the mean execution time measured using Local_PMCO mode of execution, while each checker patterned bar represents the corresponding execution time for local execution.

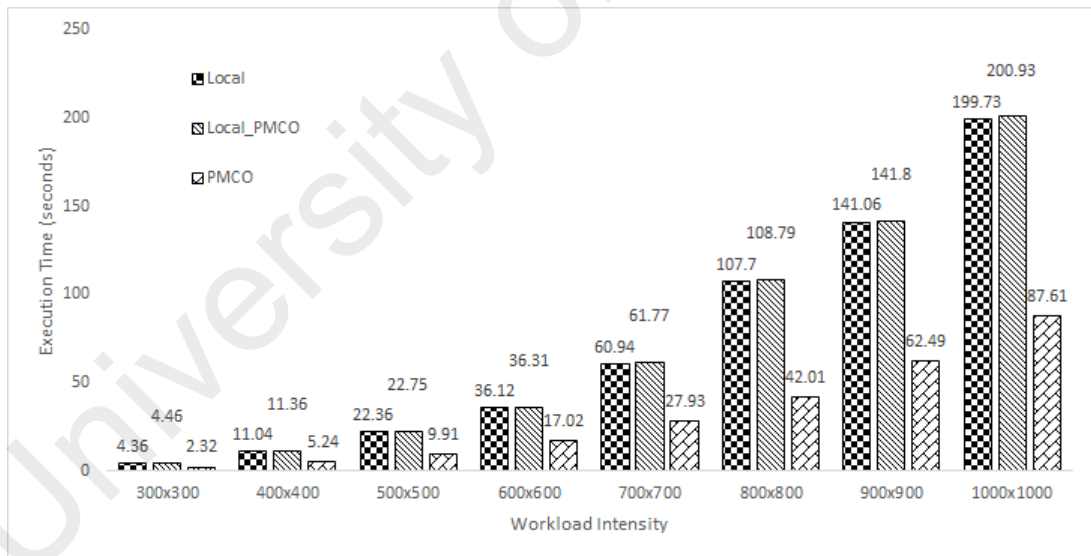


Figure 6.1: Execution time for matrix multiplication workloads generated via experimentation.

The graph in Figure 6.1 clearly depicts increasing complexity as the matrix multiplication intensity increases from left to right. The growth of the workloads intensity has a significant impact on the execution times when the workloads are entirely executed on the mobile device. However, the growth rate in PMCO execution mode is remarkably smaller

Table 6.5: t-Test: Paired Two Sample for Mean Execution Time of Local and PMCO

Pearson Correlation	0.9976199
Hypothesized Mean Difference	0
df	239
t Stat	16.98225931
P(T<=t) one-tail	2.91029×10^{-43}
t Critical one-tail	1.651254165
P(T<=t) two-tail	5.82057×10^{-43}
t Critical two-tail	1.969939406

than local execution. Execution of the last workload in our experiment takes more than 199 seconds to complete, which suggest incapacitation of executing higher workloads in the mobile device. Although the results of descriptive statistics summarized in Tables 6.1, 6.2, 6.3, and 6.4 and demonstrated in Figure 6.1 advocate remarkable improvement in execution time of the application in PMCO, further analysis is undertaken via paired samples t-test to ensure that the mean application execution times in local and PMCO modes are significantly different.

For this purpose, our null hypothesis H_o is that there is no reduction in the mean execution time of a workload when it is executed in PMCO mode as compared to the local execution mode. Table 6.5 presents the results of t-test over the mean execution of local execution and execution using PMCO based upon the experimental setup explained in section 5.1. Table indicates, that $t(239) = 16.98, p < 0.05$. The t-value and p-value advocate the significance of the difference between mean local device and PMCO execution time values. Positive t-value advocates that local execution takes more time than PMCO on average. The two-tail p -value is observed to 5.82057×10^{-43} . Hence, we reject the null hypothesis H_o and accept the alternate hypothesis that there is a significant improvement in the execution time when the workload is executed using PMCO execution mode. Therefore, the time saving using our proposed framework is significant compared to local execution mode.

For the local execution of low-intensity workloads, the native computing resources

of the mobile device suffice to complete the task without much maintenance operations (e.g., loading data into memory and interrupting CPU execution). At the beginning of the execution, all the data are loaded into the main memory and execution starts by the CPU. Once the execution is completed, the results are sent back to the main memory to present to the user. Such operation does not entertain I/O tasks unnecessarily. However, when workload intensity is high, computation in local execution mode demands more resources, including CPU, cache, RAM, and storage which are not available natively. Such constraints cause execution to prolong. For instance, in the absence of high clock speed CPU, the execution takes more CPU cycle to complete. Moreover, due to limitation in the main memory, it is not possible to store the entire data into the RAM for medium and high-intensity workloads. Thus, there will be continuous time-consuming I/O operations to load data into the main memory and store them into the peripheral RAM (storage) and vice versa. Such switching and I/O operations are highly contributing to the execution time prolonging in higher workloads.

Nevertheless, significant differences in local and PMCO execution enable mobile users to initiate PMCO execution of extremely huge workloads on their mobile devices toward gaining similar functionality experience as desktop computers. Such differences are better visible in Figure 6.2. Scattered triangles and squares across the graph and corresponding interpolating lines show the differences in achievements and the correlation between the workloads intensity and time saving. In the first workloads with low intensity, the difference between circle and triangle is comparatively smaller than of high intensity.

Execution time in local mode highly is affected by workload intensity and computing power of the mobile device (including CPU clock speed, RAM, storage, cache). However, as stated in the previous chapter, the PMCO execution time is dependent on other metrics, checkpoint/restart time, and underlying network bandwidth dynamics. Figure 6.3 shows

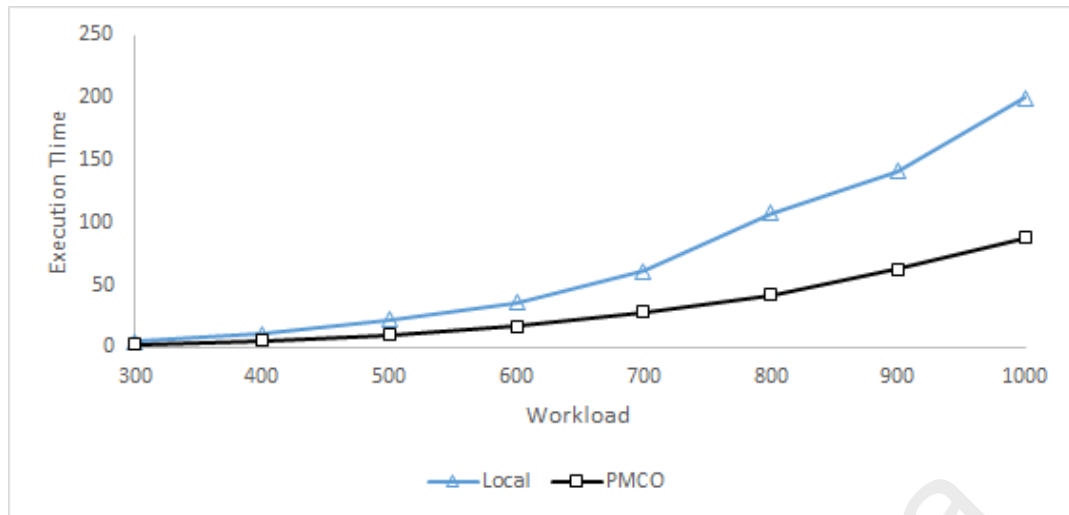


Figure 6.2: Scattered plot with interpolation lines for matrix multiplication execution time.

a stacked chart which details the timing of each contributing entity to the total execution time in PMCO execution mode. However, these contributing factors play a significant role in the overall offloading benefit. Figure 6.4 shows a 100% stacked chart, from which it is clear that in our experiment for the first workload the overhead in the application execution time because of the contributing factors is about 40%, while as the workload intensity increases this overhead of the contributing factors also decreases, so it further extend our discussion in the previous paragraphs that intenser the workload the beneficial it would be to offload. The solid black color in the bar represents the offloading time, the white chunk represents the response time, while the grey chunk is the checkpoint restart overhead in the application execution time. Finally, the blue chunk presents the execution time it takes on the remote server.

6.1.1.1 Validation

To validate the results of performance evaluation produced via experimental analysis presented in the main body of this section, statistical modeling is undertaken in this study whose results are shown in this section and we provide data related to the analysis of the execution time of eight workloads in two execution modes of local and PMCO.

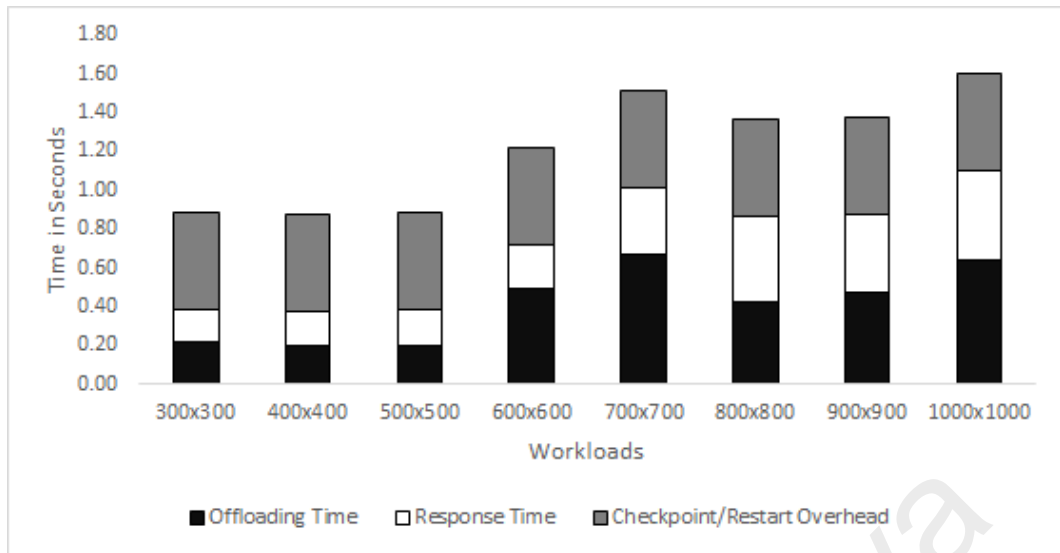


Figure 6.3: Breakdown of the contributing factors of remote execution time using PMCO.

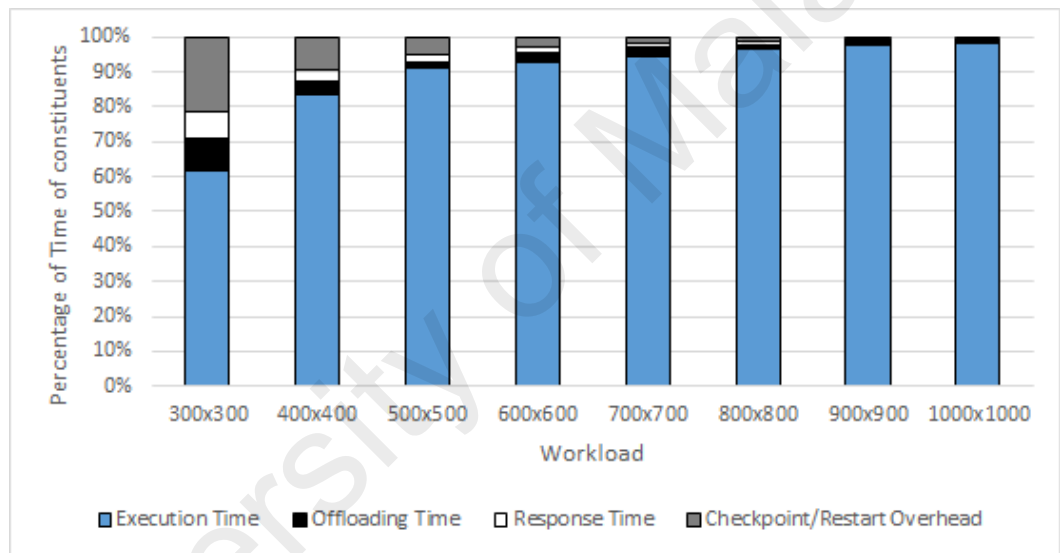


Figure 6.4: Impact of the contributing factors on remote execution time using PMCO.

In Chapter 5, we have presented the statistical model for execution time on local execution and PMCO execution mode. The statistical model for local execution is presented in equation (5.12), for quick reference, it is written here as:

$$T_{local}(W_i) = (2.02212695240863 \times 10^{-7} \times c_i) - 3.315317243 \quad (6.1)$$

Similarly, the statistical model for PMCO based execution is presented in equation (5.16), for quick reference it is written here as:

Table 6.6: The execution time data generated via statistical modeling for local and PMCO execution modes

Workload Intensity	Local	PMCO
Mat. Mult. (300x300)	2.144425528	0.945005636
Mat. Mult. (400x400)	9.626295252	4.171211136
Mat. Mult. (500x500)	21.96126966	9.484394836
Mat. Mult. (600x600)	40.36262493	17.40518034
Mat. Mult. (700x700)	66.04363722	28.45419124
Mat. Mult. (800x800)	100.2175827	43.15205114
Mat. Mult. (900x900)	144.0977376	62.01938364
Mat. Mult. (1000x1000)	198.897378	85.57681234

$$\begin{aligned}
 T_{PMCO}(W_i) = & 0.5 + (((6.97687 \times 10^{-8} \times c_i) + 0.064535721) + \\
 & ((8.46526 \times 10^{-8} \times d_i) + 0.1306192)) \\
 & + ((8.67706 \times 10^{-8} \times g_i) - 2.113132585)
 \end{aligned} \tag{6.2}$$

Table 6.6, presents the data generated using these two equations for local execution and PMCO execution according to the matrix multiplication workloads corresponding asymptotic bounds presented and explained in details in corresponding sections in Chapter 5.

The numerical results in Table 6.6 indicate that achievements in higher workloads are considerably higher than lower workloads. However, from a percentage point view, all the eight workloads save almost 43% to 44% of its execution time. This can also be explained the local execution takes up to 2.2 to 2.3 times more than the execution time on PMCO. Figure 6.5, presents the comparison between the mean execution time for the matrix multiplication workloads gathered via experimentation in both local and PMCO mode of execution vs. the execution time values generated by the statistical model. Each diagonal bricks bar in Figure 6.5, represents the mean value of execution time measured using PMCO mode executions of thirty iterations for each corresponding matrix multiplication workload. Similarly, each checker patterned bar represents the mean execution time of the corresponding workload in local execution. Each diagonal strips bar represents the

execution time generated using a statistical model for local execution. Similarly, each dotted pattern bar presents the execution time generated using statistical model PMCO based execution.

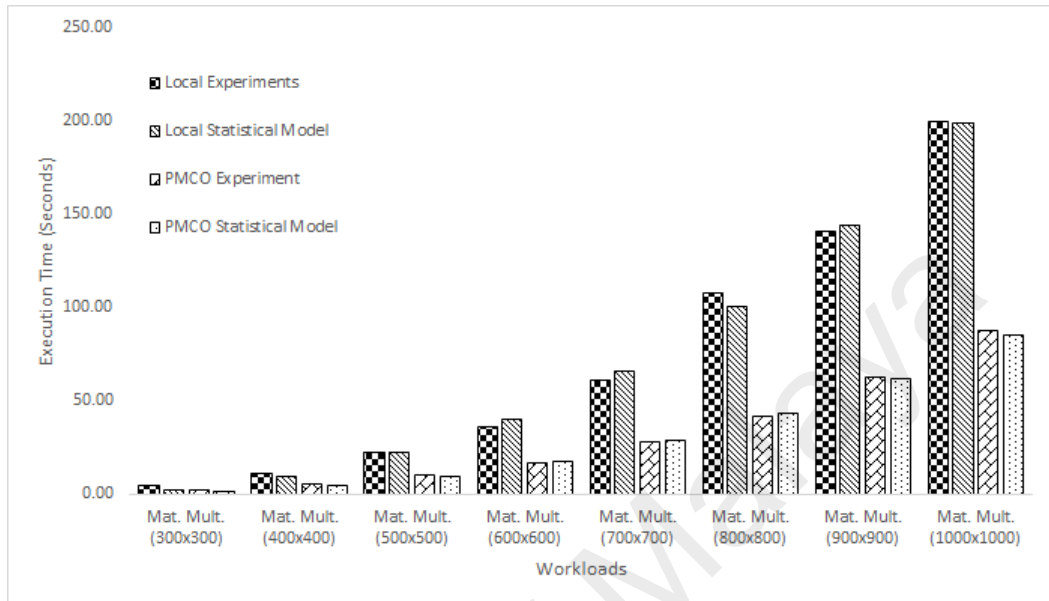


Figure 6.5: Execution time validation by experimentation vs. statistical model.

The graph shows there is no significant difference between the experimental mean execution time and of that generated using a statistical model for both execution modes. However, the experimental values and model slightly fluctuate from each other. This phenomenon can be better presented in scatter line plot in Figure 6.6, and 6.7 showing the local execution and PMCO execution time, respectively. Scattered triangles and squares across the graph and corresponding interpolating lines show the differences in achievements and the correlation between the workloads intensity and time saving through experimental data and statistical model. In Figure 6.6, and 6.7, the lines clearly overlapped each other and seemed that there is no difference between the experimental observations and model values.

In overall, the results of experimental analysis on the execution time show significant improvement in improving the application response time when our proposed model is deployed. This remarkable achievement is due to several factors including, lightweight

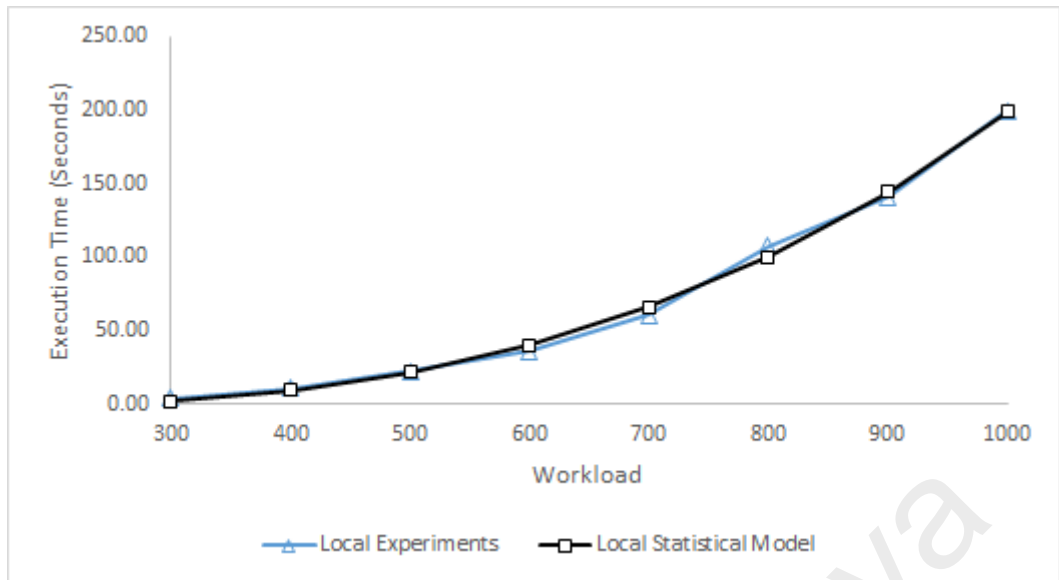


Figure 6.6: Scattered plot with interpolation lines for matrix multiplication execution time in local mode using experiments vs. statistical model.

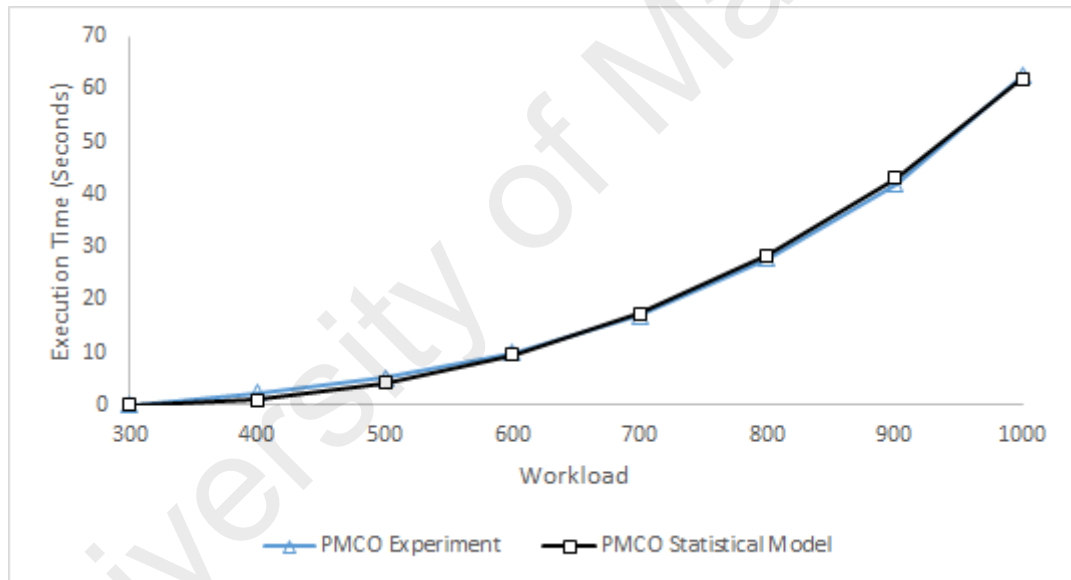


Figure 6.7: Scattered plot with interpolation lines for Matrix multiplication execution time in PMCO mode using experiments vs. statistical model.

nature of underlying checkpoint/restart technology, single-hop remote computing infrastructure, and homogeneity of the hardware and software infrastructure which are significant characteristics that are considered in design and development of the proposed framework. The results in this section are comparable with and supporting and validating the findings in the statistical analysis section.

6.1.2 Consumed energy

In this section, we present energy consumption in Joules of executing the experimental matrix multiplication application in Local, Local_PMCO and PMCO execution modes along with statistical comparison. Tables 6.7, 6.8, and 6.9 presents the data related to the energy consumed by the mobile device which are collected in Local, Local_PMCO and PMCO execution modes for eight granularity levels of matrices, respectively. Each table summarizes mean consumed energy, standard deviation, error estimate, and consumed energy with 95% confidence interval for thirty workloads of each eight intensity levels. Similar to the execution time, we present consumed energy with 95% confidence interval to enhance the reliability of our data. The small value of error estimates based on 95% confidence interval at the end of Tables 6.7, 6.8, and 6.9 testify reliability of collected energy data.

The minimum energy savings are as high as 80% to 87% for varying workloads. The mean energy saving is 79% to 86% for varying workloads. Descriptive statistics of analyzing consumed energy data are summarized in Table 6.10 including minimum, maximum, and mean consumed energy of eight workload intensities. As shown in the Table, there is significant energy saving when performing a task outside the mobile on the remote computing infrastructure. PMCO execution reduces mobile consumed energy as significant as 87% compared to the local execution. In average, consumed energy testifies the fact that local execution of the compute intensive workloads consumes 6.3 times more energy compared to the PMCO execution.

Moreover, executing workloads (according to the increasing order of intensity) inside the mobile device consumes as high as 4.94, 5.62, 6.55, 6.27, 6.25, 7.16, 6.04, and 6.37 times more energy than PMCO execution which is a remarkable achievement. Mean energy consumption for eight workloads, as the last segment in the Table 6.10 shows, is

Table 6.7: Consumed energy observations with 95% confidence interval in local execution mode gathered via PowerAPI during experimentation

Execution Trace	Matrix Multiplication Granularity							
	300x300	400x400	500x500	600x600	700x700	800x800	900x900	1000x1000
	Energy Consumed (in Joules)							
1	1.27	3.63	7.57	7.08	12.33	21.73	28.57	40.49
2	1.28	3.63	7.61	7.09	12.09	21.68	28.37	40.20
3	1.28	3.66	7.66	7.09	12.23	21.67	28.33	40.12
4	1.29	3.67	7.41	7.09	12.17	21.70	28.32	40.19
5	1.33	3.72	7.95	7.09	12.14	21.67	28.24	40.22
6	1.31	3.65	7.83	7.10	12.17	21.65	28.21	40.31
7	1.27	3.52	7.70	7.08	12.12	21.70	28.32	40.17
8	1.26	3.71	7.72	7.09	12.22	21.69	28.30	40.30
9	1.26	3.65	7.77	7.11	12.11	21.67	28.45	40.23
10	1.19	3.72	7.86	7.09	12.12	21.64	28.40	40.22
11	1.29	3.70	7.63	7.07	12.14	21.68	28.51	40.19
12	1.13	3.69	7.41	7.09	12.15	21.71	28.30	40.24
13	1.33	3.63	7.65	7.10	12.16	21.67	28.36	40.28
14	1.29	3.62	7.67	7.10	12.07	21.64	28.18	40.28
15	1.23	3.67	7.67	7.10	12.27	21.69	28.51	40.46
16	1.24	3.64	7.76	7.10	12.13	21.66	28.85	41.30
17	1.25	3.59	7.83	7.11	12.11	21.67	28.24	40.26
18	1.30	3.63	7.64	7.10	12.28	21.66	28.39	40.37
19	1.31	3.63	7.63	7.12	12.14	21.82	28.21	40.36
20	1.29	3.57	7.71	7.11	12.13	21.67	28.36	40.27
21	1.33	3.60	7.68	7.10	12.16	21.68	28.56	40.19
22	1.30	3.63	7.70	7.09	12.12	21.68	28.31	40.35
23	1.25	3.57	7.67	7.10	12.16	21.68	28.62	40.20
24	1.25	3.64	7.76	7.10	12.12	21.67	28.74	40.21
25	1.22	3.63	7.63	7.10	12.14	21.64	28.22	40.32
26	1.25	3.69	7.67	7.11	12.38	21.61	28.29	40.26
27	1.28	3.63	7.61	7.08	12.15	21.67	28.38	40.34
28	1.30	3.64	8.55	7.10	12.23	21.67	28.67	41.03
29	1.30	3.67	6.85	7.22	12.23	21.68	28.35	40.28
30	1.24	3.99	7.62	7.20	12.12	21.68	28.46	40.34
Min	1.13	3.52	6.85	7.07	12.07	21.61	28.18	40.12
Mean	1.27	3.65	7.68	7.10	12.17	21.68	28.40	40.33
Median	1.28	3.64	7.67	7.10	12.14	21.67	28.36	40.27
Maximum	1.33	3.99	8.55	7.22	12.38	21.82	28.85	41.30
Std. Deviation	0.04	0.08	0.25	0.03	0.07	0.04	0.17	0.24
Confidence Int.	0.02	0.03	0.09	0.01	0.03	0.01	0.06	0.09

Table 6.8: Consumed Energy observations with 95% Confidence Interval in Local_PMCO Execution Mode gathered via PowerAPI during experimentation

Execution Trace	Matrix Multiplication Granularity							
	300x300	400x400	500x500	600x600	700x700	800x800	900x900	1000x1000
	Energy Consumed (in Joules)							
1	1.33	3.94	8.07	7.10	12.30	21.87	28.45	40.38
2	1.30	3.90	7.99	7.07	12.17	21.78	28.26	40.21
3	1.33	3.78	8.03	7.08	12.12	21.76	28.32	40.24
4	1.30	3.80	8.47	7.13	12.38	21.79	28.30	40.47
5	1.30	3.76	8.07	7.09	12.12	21.76	28.30	40.76
6	1.29	3.77	8.02	7.26	12.29	21.79	28.25	40.81
7	1.34	3.99	7.94	7.08	12.24	21.70	28.36	40.65
8	1.33	3.77	7.92	7.11	12.05	21.75	28.29	40.50
9	1.29	3.87	7.94	7.09	12.91	21.77	28.37	40.69
10	1.32	3.77	7.92	7.11	12.11	21.79	28.40	40.38
11	1.33	3.99	7.97	7.08	12.13	21.73	28.38	40.39
12	1.36	3.79	8.03	7.08	12.16	21.75	28.54	40.22
13	1.31	3.82	7.95	7.09	12.17	21.79	28.27	40.43
14	1.34	3.82	7.94	7.19	12.04	21.79	28.63	40.47
15	1.41	3.79	7.93	7.17	12.07	21.78	28.38	40.23
16	1.38	3.79	7.83	7.11	12.11	21.75	28.38	40.26
17	1.38	3.94	7.96	7.07	12.15	21.63	28.89	40.43
18	1.31	3.80	7.95	7.10	12.15	21.73	28.79	40.59
19	1.30	3.80	7.97	7.11	12.32	21.77	28.72	40.25
20	1.32	3.90	8.06	7.18	12.08	21.77	28.31	40.46
21	1.29	3.86	7.91	7.19	12.13	21.78	28.54	40.27
22	1.26	3.84	7.90	7.07	12.13	21.82	28.35	41.02
23	1.41	3.87	7.91	7.09	12.13	21.80	28.26	41.05
24	1.38	4.12	7.86	7.14	12.34	21.84	28.34	40.97
25	1.37	3.94	8.00	7.07	12.37	21.83	28.19	40.39
26	1.39	3.80	7.98	7.10	12.13	21.81	28.29	40.57
27	1.35	3.90	8.04	7.08	12.16	21.77	28.35	40.40
28	1.26	3.88	7.18	7.08	12.32	21.79	28.44	40.29
29	1.29	3.80	8.99	7.06	12.60	21.83	28.28	40.26
30	1.36	3.77	8.01	7.08	12.40	21.74	28.50	40.37
Min	1.26	3.76	7.18	7.06	12.04	21.63	28.19	40.21
Mean	1.33	3.85	7.99	7.11	12.23	21.78	28.40	40.48
Median	1.33	3.82	7.97	7.09	12.15	21.78	28.36	40.41
Maximum	1.41	4.12	8.99	7.26	12.91	21.87	28.89	41.05
Std. Deviation	0.04	0.09	0.26	0.05	0.18	0.05	0.17	0.24
Confidence Int.	0.01	0.03	0.09	0.02	0.06	0.02	0.06	0.09

Table 6.9: Consumed Energy observations with 95% Confidence Interval in PMCO Execution Mode gathered via PowerAPI during experimentation

Execution Trace	Matrix Multiplication Granularity							
	300x300	400x400	500x500	600x600	700x700	800x800	900x900	1000x1000
	Energy Consumed (in Joules)							
1	0.28	0.55	1.23	1.05	1.88	2.94	4.67	6.34
2	0.28	0.56	1.25	1.16	1.80	3.07	4.73	6.28
3	0.24	0.58	1.17	1.13	2.06	3.07	4.75	6.16
4	0.26	0.62	1.19	1.14	1.84	3.04	4.75	6.39
5	0.25	0.72	1.18	1.13	2.17	3.02	4.88	6.14
6	0.24	0.68	1.21	1.12	2.08	3.06	4.33	6.26
7	0.27	0.66	1.33	1.13	1.93	3.03	4.76	6.36
8	0.24	0.65	1.20	1.12	2.01	2.97	4.72	6.31
9	0.27	0.66	1.31	1.20	1.94	2.99	4.69	6.28
10	0.25	0.60	1.25	1.12	1.88	3.12	4.77	6.27
11	0.28	0.67	0.98	1.16	1.87	3.06	4.71	6.41
12	0.25	0.70	1.16	1.11	1.93	2.94	4.76	6.22
13	0.25	0.69	0.78	1.14	1.95	3.10	4.66	6.30
14	0.26	0.62	1.13	1.12	1.89	3.07	4.74	6.47
15	0.22	0.68	1.14	1.13	1.95	3.04	4.79	6.43
16	0.25	0.67	1.25	1.14	1.98	3.00	4.72	6.39
17	0.24	0.62	1.12	1.11	2.18	2.99	4.66	6.22
18	0.26	0.67	1.27	1.11	1.99	3.02	4.76	6.38
19	0.26	0.59	1.18	1.14	1.99	3.10	4.63	6.39
20	0.26	0.66	1.24	1.13	1.90	3.07	4.60	6.31
21	0.22	0.61	1.20	1.21	1.91	3.07	4.70	6.44
22	0.25	0.71	1.22	1.08	1.97	2.81	4.76	6.16
23	0.26	0.59	1.10	1.10	1.86	3.06	4.68	6.35
24	0.27	0.67	1.15	1.19	1.86	3.02	4.70	6.45
25	0.25	0.70	1.20	1.12	1.93	2.99	4.84	6.41
26	0.27	0.71	1.14	1.16	1.90	3.00	4.28	6.37
27	0.24	0.65	1.07	1.10	1.86	2.93	4.69	6.34
28	0.26	0.63	1.21	1.19	1.97	3.00	4.77	6.27
29	0.28	0.67	1.11	1.15	1.96	3.09	4.77	6.38
30	0.29	0.67	1.17	1.10	1.91	3.05	4.72	6.33
Min	0.22	0.55	0.78	1.05	1.80	2.81	4.28	6.14
Mean	0.26	0.65	1.17	1.13	1.95	3.02	4.70	6.33
Median	0.26	0.66	1.19	1.13	1.93	3.03	4.72	6.34
Maximum	0.29	0.72	1.33	1.21	2.18	3.12	4.88	6.47
Std. Deviation	0.02	0.05	0.10	0.03	0.09	0.06	0.12	0.09
Confidence Int.	0.01	0.02	0.04	0.01	0.03	0.02	0.04	0.03

Table 6.10: Descriptive Statistics of Consumed Energy Data gathered via PowerAPI

		Min	Mean	Median	Maximum	Std. Deviation	Confidence Int.
Mat. Mult. (300x300)	Local	1.13	1.27	1.28	1.33	0.04	0.02
	Local_PMCO	1.26	1.33	1.33	1.41	0.04	0.01
	PMCO	0.22	0.26	0.26	0.29	0.02	0.01
Mat. Mult. (400x400)	Local	3.52	3.65	3.64	3.99	0.08	0.03
	Local_PMCO	3.76	3.85	3.82	4.12	0.09	0.03
	PMCO	0.55	0.65	0.66	0.72	0.05	0.02
Mat. Mult. (500x500)	Local	6.85	7.68	7.67	8.55	0.25	0.09
	Local_PMCO	7.18	7.99	7.97	8.99	0.26	0.09
	PMCO	0.78	1.17	1.19	1.33	0.10	0.04
Mat. Mult. (600x600)	Local	7.07	7.10	7.10	7.22	0.03	0.01
	Local_PMCO	7.06	7.11	7.09	7.26	0.05	0.02
	PMCO	1.05	1.13	1.13	1.21	0.03	0.01
Mat. Mult. (700x700)	Local	12.07	12.17	12.14	12.38	0.07	0.03
	Local_PMCO	12.04	12.23	12.15	12.91	0.18	0.06
	PMCO	1.80	1.95	1.93	2.18	0.09	0.03
Mat. Mult. (800x800)	Local	21.61	21.68	21.67	21.82	0.04	0.01
	Local_PMCO	21.63	21.78	21.78	21.87	0.05	0.02
	PMCO	2.81	3.02	3.03	3.12	0.06	0.02
Mat. Mult. (900x900)	Local	28.18	28.40	28.36	28.85	0.17	0.06
	Local_PMCO	28.19	28.40	28.36	28.89	0.17	0.06
	PMCO	4.28	4.70	4.72	4.88	0.12	0.04
Mat. Mult. (1000x1000)	Local	40.12	40.33	40.27	41.30	0.24	0.09
	Local_PMCO	40.21	40.48	40.41	41.05	0.24	0.09
	PMCO	6.14	6.33	6.34	6.47	0.09	0.03

observed as high as 6.15 times more than energy consumption of the application when intensive tasks run outside the mobile device. For instance, if the energy required to locally run the eighth workload is 40.33J, by utilizing the proposed framework in this study, the same workload consumes as low as 6.33J+ energy by performing intensive task(s) outside the mobile device which is remarkable. Similarly, from the overhead point of view when the workloads are executed locally through the PMCO framework the degradation are not that much and is as low as 0.01% to 5%, approximately. As described in the previous chapter, execution of each workload is repeated thirty times to enhance the reliability of performance evaluation. So, data plotted in Figure 6.8 are mean consumed energy of the workloads for Local, Local_PMCO and PMCO execution modes. Each diagonal bricks bar in Figure 6.8, represents the mean value of compute power measured using PMCO mode of thirty iterations for each corresponding matrix multiplication workload. Similarly, each diagonal strips bar represents the mean consumed energy measured using Local_PMCO mode of execution, while each checker patterned bar represents the

corresponding compute power for local execution.

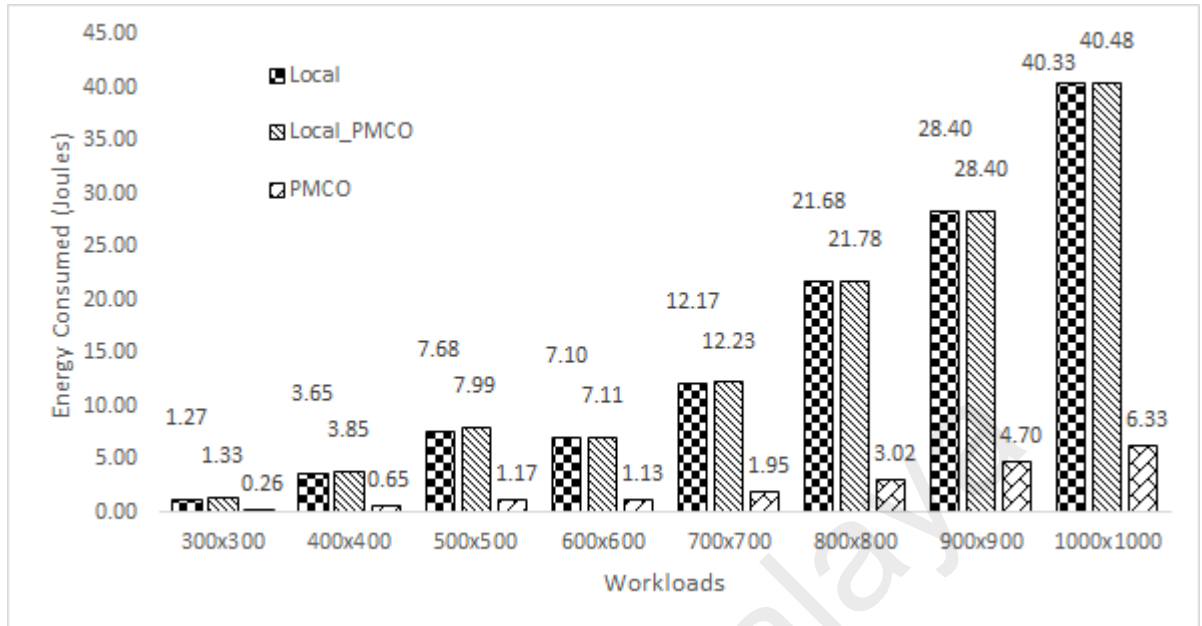


Figure 6.8: Energy consumed for matrix multiplication workloads gathered via Power-API.

The graph in Figure 6.8 clearly depicts increasing complexity as the matrix multiplication intensity increases from left to right. The growth of the workloads has a significant impact on the consumed energy when the workloads are entirely executed on the mobile device. However, the reduction growth rate of consumed energy in PMCO execution mode is remarkably smaller than local execution. Execution of the last workload in our experiment takes more than 40.33 joules to complete, which suggest incapacitation of executing higher workloads in the mobile device. Although the results of descriptive statistics summarized in Tables 6.7, 6.8, 6.9, and 6.10 and demonstrated in Figure 6.8 advocate remarkable improvement in execution time of the application in PMCO, further analysis is undertaken via paired samples t-test to ensure that the mean application energy consumption in local and PMCO modes are significantly different.

For this purpose, our null hypothesis H_0 is that there is no reduction in the mean energy consumption of a workload when it is executed in PMCO mode as compared to the local execution mode. Table 6.11 presents the results of t-test over the mean execution of

Table 6.11: t-Test: Paired Two Sample Mean of Consumed Energy of workloads of Local and PMCO Mode of execution

Pearson Correlation	0.995926991
Hypothesized Mean Difference	0
df	239
t Stat	18.46918176
P(T<=t) one-tail	3.19384×10^{-48}
t Critical one-tail	1.651254165
P(T<=t) two-tail	6.38767×10^{-48}
t Critical two-tail	1.969939406

local execution and execution using PMCO based upon the experimental setup explained in section 5.1. Table indicates, that $t(239) = 18.46, p < 0.05$. The t-value and p-value advocate the significance of the difference between mean local device and PMCO execution time values. Positive t-value advocates that local execution takes more time than PMCO on average. The two-tail p -value is observed to 6.38767×10^{-48} . Hence, we reject the null hypothesis H_0 and accept the alternate hypothesis that there is a significant reduction in the consumed energy when the workloads are executed using PMCO execution mode. Therefore, the time saving using our proposed framework is significant compared to local execution mode.

Nevertheless, significant differences in local and PMCO energy consumption enable mobile users to offload workloads using the proposal PMCO of extremely huge workloads on their mobile devices toward gaining increased battery lifetime. Such differences are better visible in Figure 6.9. Scattered triangles and squares across the graph and corresponding interpolating lines show the differences in achievements and the correlation between the workloads intensity and time saving. In the first workloads with low intensity, the difference between circle and triangle is comparatively smaller than of high intensity. However, when we observe the line of PMCO energy consumed it is not going up abruptly as the local execution is going, as the workload intensity increases.

Energy Consumption in local execution mode is highly affected by workload inten-

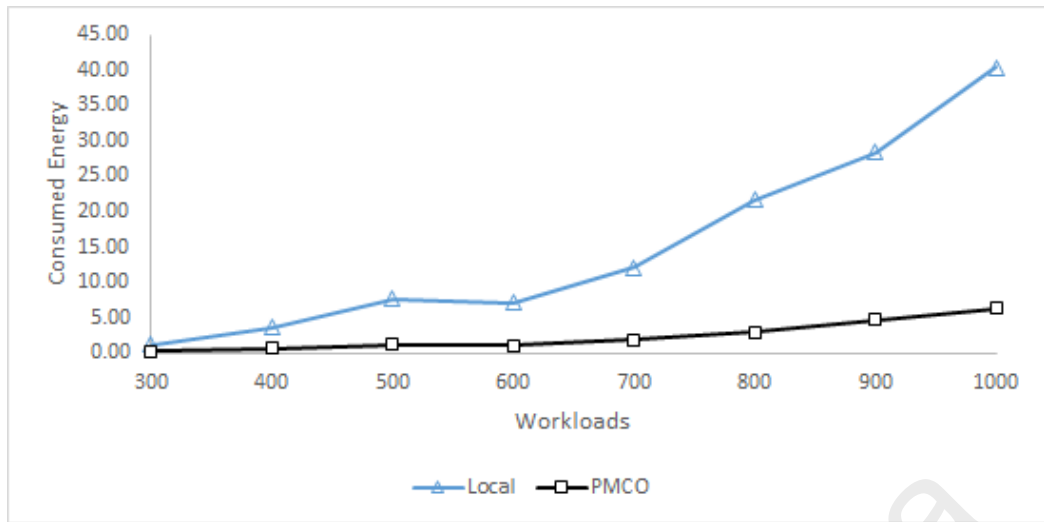


Figure 6.9: Scattered plot with interpolation lines for matrix multiplication energy consumption.

sity and its execution time, along with the power rating of the mobile device. However, as stated in the previous chapter the PMCO energy consumption is dependent on the total execution response time which is also presented in the previous section. For the sake of completeness, this phenomenon is again presented here by scattered plots in Figure 6.10 and 6.11. The figures clearly represent this behavior as the execution increases the energy consumption also increases.

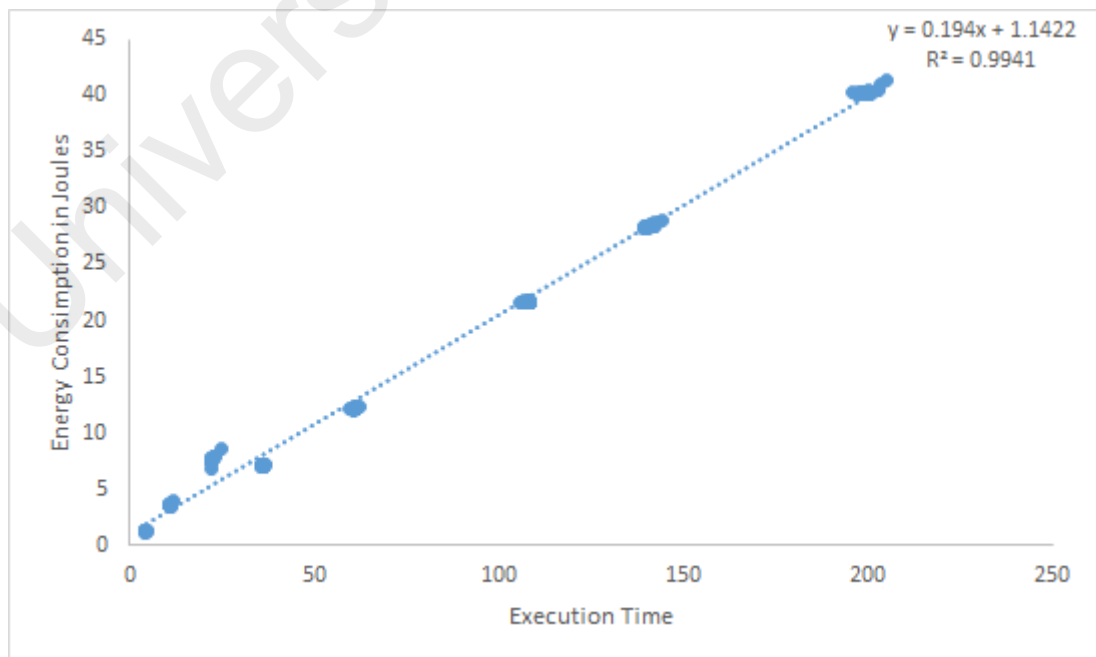


Figure 6.10: Scatter plot showing linearity correlation between local execution time and consumed energy.

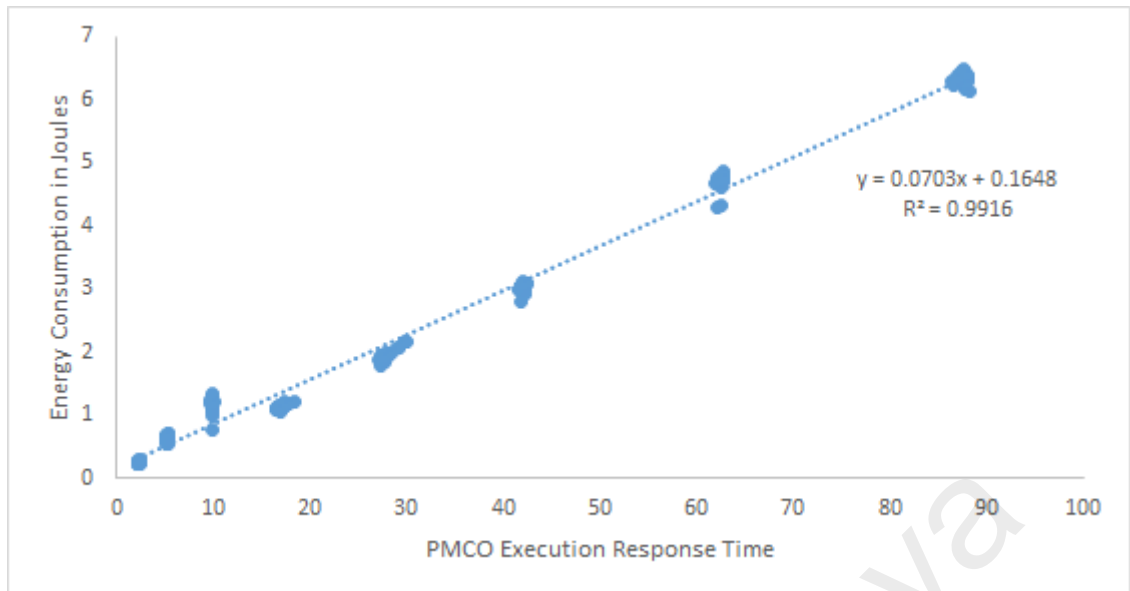


Figure 6.11: Scatter plot showing linearity correlation between PMCO execution response time and consumed energy.

Lastly, we would like show the energy consumption of the benchmark applications we utilized to measure the compute power of the proposed framework. Figure 6.12 are mean consumed energy of the benchmark workloads for Local, Local_PMCO and PMCO execution modes. Each diagonal bricks bar in the Figure 6.12, represents the mean value of compute power measured using PMCO mode of thirty iterations for each corresponding benchmark application. Similarly, each diagonal strips bar represents the mean consumed energy measured using Local_PMCO mode of execution, while each checker patterned bar represents the corresponding compute power for local execution.

The graph in Figure 6.12 clearly depicts the reduction growth rate of consumed energy in PMCO execution mode and is remarkably smaller than local execution. Execution of the Linpack workload in our experiment takes 6.24 joules to complete in local execution mode and has an overhead of around 0.20 joule on it when the same workload is executing locally using the PMCO. While the improvement is substantial as it only takes 2.15 joules on average when executed using PMCO, which is almost 2.5 to 3 times less than the local consumption.

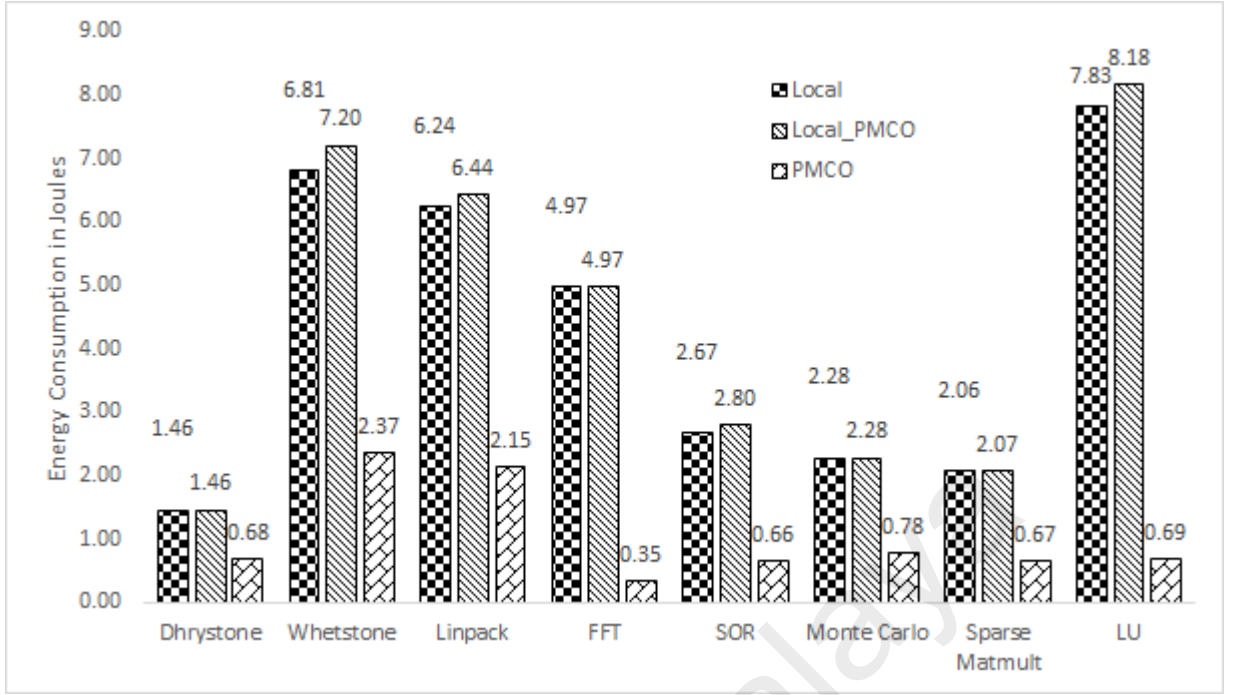


Figure 6.12: Energy consumed for benchmark application workloads gathered via PowerAPI.

6.1.2.1 Validation

To validate the results of consumed energy evaluation produced via experimental analysis presented in the main body of this section, statistical modeling is undertaken in this study whose results are shown in this section and we provide data related to the analysis of consumption of eight workloads in two execution modes of local and PMCO.

In Chapter 5, we have presented the statistical model for consumed energy on local execution and PMCO execution mode. The statistical model for energy consumption in local execution is presented in equation (5.18), for quick reference it is written here as:

$$E_m(W_i) = (0.193971484 \times T_{local}(W_i)) + 1.14220578 \quad (6.3)$$

Similarly, the statistical model for PMCO based execution is presented in equation (5.23), for quick reference it is written here as:

$$E_{PMCO}(W_i) = (0.070284574 \times T_{PMCO}(W_i)) + 0.164839255 \quad (6.4)$$

Table 6.12: The energy consumption data generated via statistical modeling for local and PMCO execution modes

Workload Intensity	Local	PMCO
Mat. Mult. (300x300)	1.558163182	0.231258573
Mat. Mult. (400x400)	3.009432558	0.458011053
Mat. Mult. (500x500)	5.402065853	0.831445906
Mat. Mult. (600x600)	8.971404048	1.388154941
Mat. Mult. (700x700)	13.95278812	2.164729966
Mat. Mult. (800x800)	20.58155905	3.197762789
Mat. Mult. (900x900)	29.09305783	4.523845218
Mat. Mult. (1000x1000)	39.72262542	6.17956906

Table 6.12, presents the data generated using these two equations of statistical model for consumed energy in local execution and PMCO execution according to the matrix multiplication workloads corresponding asymptotic bounds presented and explained in details in corresponding sections in Chapter 5.

The numerical results in Table 6.12 indicate that achievements in higher workloads are considerably higher than lower workloads. However, from a percentage point view, all the eight workloads save almost 84 to 85% of its energy consumption when executed using PMCO according to the statistical model. This can also be explained the local execution takes up to 14 to 15 times more energy than the execution time on PMCO. Figure 6.13, presents the comparison between the mean energy consumption for the matrix multiplication workloads gathered via experimentation in both local and PMCO mode of execution vs the energy consumption data generated by the statistical model. Each diagonal bricks bar in the Figure 6.13, represents the mean value of energy consumption measured using PMCO mode executions of thirty iterations for each corresponding matrix multiplication workload. Similarly, each checker patterned bar represents the mean energy consumption of the corresponding workload in local execution. Each diagonal strips bar represents the energy consumption generated using statistical model for local execution. Similarly, each dotted pattern bar presents the energy consumption generated using the statistical model PMCO based execution.

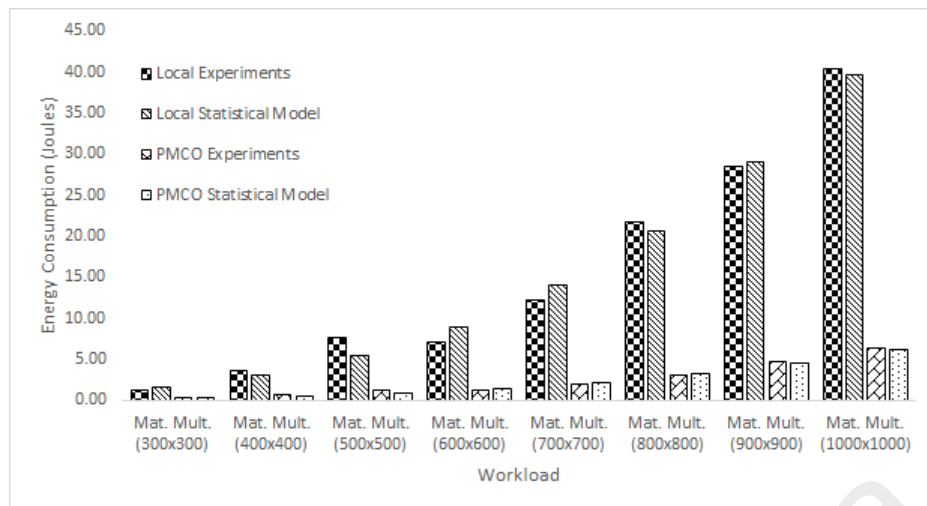


Figure 6.13: Consumed energy validation by experimentation vs. statistical model.

The graph shows there is no significant difference between the experimental mean energy consumption and of that generated using statistical model for both execution modes. However, the experimental values and model slightly fluctuate from each other. This phenomenon can be better presented in scatter line plot in Figure 6.14, and 6.15 presented the local execution and PMCO energy consumption, respectively. Scattered triangles and squares across the graph and corresponding interpolating lines show the differences in achievements and the correlation between the workloads intensity and time saving through experimental data and statistical model.

In Figures 6.14, and 6.15 the lines clearly overlapped in most of the workloads. However, the real experiments details deviate slightly as the power consumption of the mobile device depends on many internal components and operating systems procedure which in some instance may consume extra energy. From these scatter line plots it seems that there is no significant difference between the experimental observations and model values as they are nearly overlapping each other.

In overall, the results of experimental analysis on the energy consumption show significant improvement in reducing the energy consumption of a mobile device when our proposed model is deployed. This remarkable achievement is due to several factors

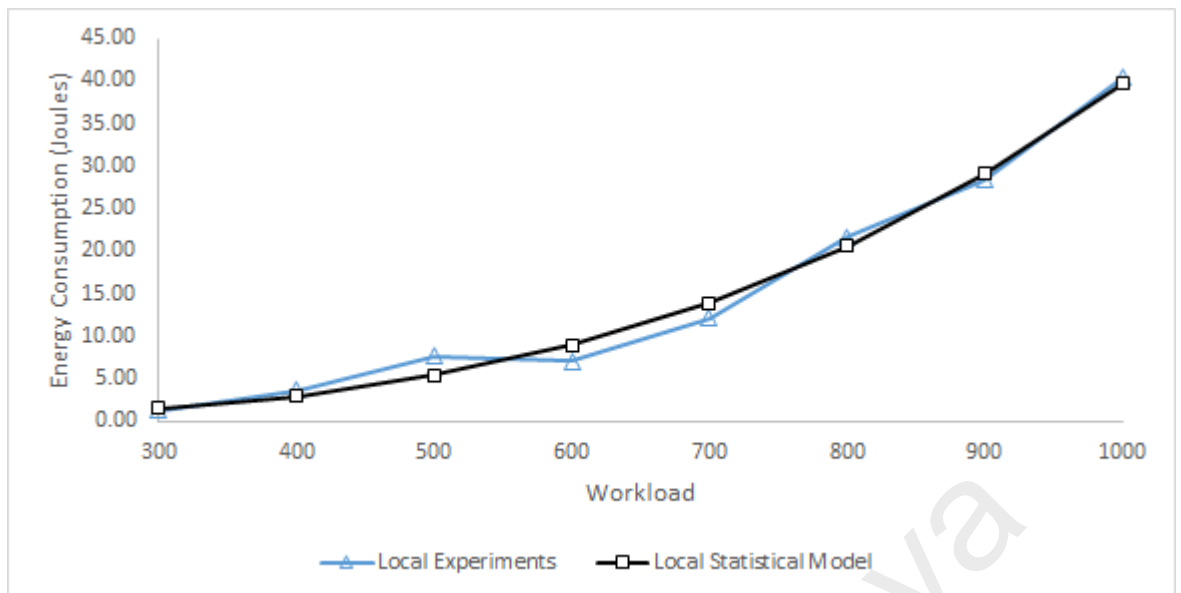


Figure 6.14: Scattered plot with interpolation lines for matrix multiplication energy consumption in local mode using experiments vs. statistical model.

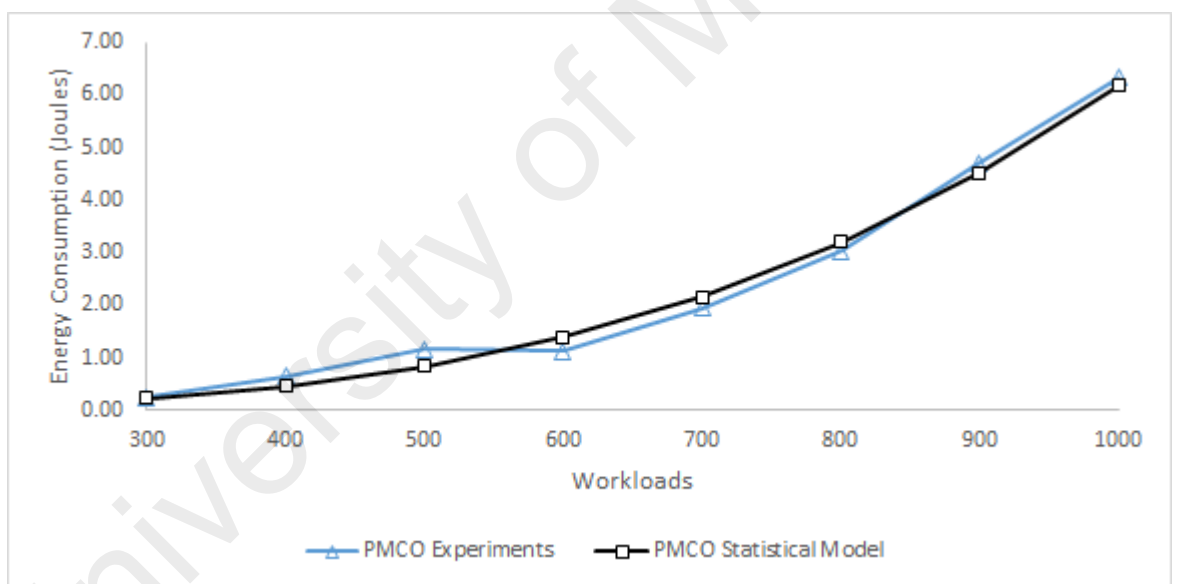


Figure 6.15: Scattered plot with interpolation lines for matrix multiplication energy consumption in PMCO mode using experiments vs. statistical model.

including, lightweight nature of underlying checkpoint/restart technology, single-hop remote computing infrastructure which are the significant characteristics that are considered in the design and development of the proposed framework. The results in this section are comparable with and supporting the findings in the statistical analysis section.

6.1.3 Compute power

This section presents temporal results of executing experimental benchmarking applications in three execution environments. One environment is local whereas in the second environment which is Local_PMCO, the benchmark is executed locally through the environment of PMCO, while the last environment is PMCO, in which the benchmark execution starts locally and then once the execution reaches a migration marker is offloaded to performed remotely using any of mobile cloud resource. Data related to compute power in this section are gathered using experimental analysis. Several tables and charts are used to demonstrate the findings.

Tables 6.13, 6.14, and 6.15 presents the temporal data related to the compute power measured by executing the benchmarking applications in local environment, in Local_PMCO and PMCO execution environments, respectively. Each of these tables presents individual values of each execution and followed by a summary of descriptive statistics containing mean compute power measured, standard deviation, error estimate based on 95% confidence interval. Column Execution trace contains the trace number of the execution of the benchmarking workloads whose measurement of compute power are used to calculate mean values of the minimum, maximum, and mean represented in the Tables.

The small error estimates presented using 95% confidence interval shown at the end of the Tables 6.13, 6.14, and 6.15 ensure the reliability of the collected data during real time experimentation. For example, the maximum error estimate for the Linpack workload executed in PMCO execution with 95% confidence interval means that the PMCO execution of the Linpack workload falls in the range of $(148.64 - 0.72) < \mu < (148.64 + 0.72)$. This range shows that if the execution is repeated, the compute power value falls in this range with 95% confidence. To better demonstrate the significance of our achievements and effectively interpret the results, we perform a comprehensive statistical

Table 6.13: Compute Power Observations in Local Execution Environment Generated via Standardized Benchmarking Mechanisms

Execution Trace	Scimark2							
	D.Stone MIPS	W.Stone MWIPS	L.Pack MFLOPS	FFT	SOR	MC MFLOPS	Mat.	LU
1	374.90	170.32	36.68	9.06	52.93	10.75	23.19	33.94
2	374.43	169.95	37.54	8.88	52.93	10.75	23.27	33.94
3	377.59	171.94	37.09	8.72	53.23	10.75	23.19	33.96
4	378.00	172.66	36.36	8.61	53.37	10.75	23.19	33.96
5	375.20	170.23	36.62	8.5	53.37	10.75	23.36	33.96
6	374.75	170.69	37.2	8.67	53.23	10.75	23.1	33.96
7	375.17	173.72	37.22	9	53.23	10.75	23.19	33.96
8	375.56	172.14	37.5	8.56	53.23	10.72	23.19	33.96
9	378.45	169.79	36.53	8.56	53.37	10.75	23.19	33.96
10	375.44	170.27	37.07	8.63	53.37	10.72	23.19	33.96
11	373.02	170.18	36.51	8.86	53.23	10.72	23.27	33.96
12	374.80	171.82	37.19	9.02	53.37	10.72	23.19	33.94
13	364.56	169.67	36.47	8.87	53.23	10.75	23.19	33.93
14	374.64	169.43	36.41	8.63	53.23	10.72	23.27	33.94
15	378.23	170.25	36.97	8.65	53.37	10.75	23.27	33.91
16	353.79	173.31	36.11	8.75	53.37	10.72	23.19	33.96
17	375.86	172.71	36.55	8.63	53.37	10.75	23.27	33.94
18	365.60	168.71	37.44	8.6	53.23	10.72	23.27	33.94
19	372.15	170.11	36.39	8.71	53.23	10.72	23.27	33.93
20	375.32	172.55	36.47	8.89	53.23	10.72	23.19	33.96
21	377.88	172.78	37.31	8.61	53.23	10.75	23.19	33.96
22	373.98	171.77	36.35	8.66	53.37	10.72	23.19	33.96
23	377.87	168.61	36.66	8.76	53.37	10.75	23.27	33.96
24	365.87	170.34	37.36	8.88	53.23	10.72	23.19	33.96
25	374.00	173.01	36.26	8.71	53.08	10.75	23.19	33.98
26	372.85	172.79	36.55	8.6	53.23	10.72	23.27	33.98
27	372.63	169.63	32.34	8.73	53.23	10.72	23.19	33.98
28	379.01	170.78	36.45	8.59	53.23	10.72	23.27	33.98
29	352.71	173.61	35.98	8.65	53.08	10.75	23.19	33.98
30	353.14	172.90	37.07	9.03	53.23	10.75	23.27	33.96
Min	352.71	168.61	32.34	8.5	52.93	10.72	23.1	33.91
Mean	372.25	171.22	36.62	8.73	53.25	10.74	23.22	33.96
Median	374.78	170.73	36.59	8.69	53.23	10.75	23.19	33.96
Maximum	379.01	173.72	37.54	9.06	53.37	10.75	23.36	33.98
Std. Deviation	7.37	1.53	0.92	0.15	0.12	0.015	0.051	0.016
Confidence Int.	2.64	0.55	0.33	0.06	0.042	0.005	0.018	0.005

Table 6.14: Compute Power Observations in Local_PMCO Execution Environment Generated via Standardized Benchmarking Mechanisms

Execution Trace	Scimark2							
	D.Stone MIPS	W.Stone MWIPS	L.Pack MFLOPS	FFT	SOR	MC MFLOPS	Mat.	LU
1	369.42	165.73	35.72	9.01	52.93	10.75	23.27	33.96
2	362.00	166.04	35.07	8.66	53.37	10.75	23.27	33.96
3	370.71	169.12	36.04	8.76	53.23	10.75	23.19	33.94
4	367.20	166.92	34.61	8.70	53.37	10.69	23.27	33.94
5	364.41	167.51	36.21	8.68	53.23	10.69	23.1	33.93
6	362.27	166.81	35.61	8.72	53.23	10.69	23.19	33.93
7	356.09	169.78	36.17	8.74	53.23	10.69	23.27	33.86
8	364.00	163.79	35.16	8.65	53.23	10.69	23.27	33.93
9	370.38	166.27	33.12	8.70	53.23	10.75	23.27	33.96
10	370.89	167.44	35.92	8.57	53.23	10.72	23.19	33.98
11	368.64	167.11	36.25	8.72	53.23	10.75	23.19	33.96
12	365.90	169.96	35.84	8.69	53.08	10.72	23.27	33.94
13	364.48	152.87	36.45	8.76	53.23	10.65	23.19	33.98
14	367.75	168.70	35.19	8.75	53.23	10.75	23.19	33.94
15	370.43	169.38	36.23	8.72	53.23	10.72	23.19	33.89
16	366.82	165.78	35.73	8.69	53.23	10.75	23.19	33.89
17	367.46	166.14	36.21	8.75	53.08	10.79	23.27	33.96
18	371.42	167.29	35.30	8.65	53.37	10.72	23.27	33.96
19	346.76	169.87	35.93	8.72	53.37	10.69	23.27	33.98
20	341.84	169.85	36.67	8.69	53.37	10.69	23.27	33.94
21	345.28	167.12	35.25	8.71	53.23	10.69	23.19	33.94
22	368.00	167.47	36.34	9.01	53.23	10.69	23.19	33.93
23	371.05	166.88	36.48	8.71	53.08	10.69	23.19	33.96
24	370.77	170.55	36.10	8.73	53.37	10.72	23.19	33.98
25	370.41	168.56	36.28	8.65	53.23	10.75	23.19	33.98
26	367.94	165.59	35.63	8.66	53.37	10.75	23.27	33.93
27	367.28	156.71	36.18	8.7	53.37	10.75	23.27	33.93
28	366.4	166.797	34.8	8.75	53.08	10.75	23.19	33.96
29	366.86	168.037	35.92	8.58	53.23	10.75	23.1	33.94
30	367.59	167.282	35.8	8.77	53.23	10.75	23.19	33.98
Min	341.84	152.87	33.12	8.57	52.93	10.65	23.10	33.86
Mean	365.02	166.71	35.74	8.72	53.24	10.72	23.22	33.95
Median	367.37	167.20	35.92	8.71	53.23	10.72	23.19	33.94
Maximum	371.42	170.55	36.67	9.01	53.37	10.79	23.27	33.98
Std. Deviation	7.68	3.64	0.71	0.09	0.11	0.03	0.05	0.03
Confidence Int.	2.75	1.30	0.26	0.03	0.04	0.01	0.02	0.01

Table 6.15: Compute Power Observations in PMCO Execution Environment Generated via Standardized Benchmarking Mechanisms

Execution Trace	Scimark2							
	D.Stone MIPS	W.Stone MWIPS	L.Pack MFLOPS	FFT	SOR	MC MFLOPS	Mat.	LU
1	1321.49	673.973	150.68	39.91	165.19	36.47	78.29	116.35
2	1183.74	679.497	150.68	44.2	161.02	36.28	78.29	105.49
3	1242.79	669.178	150.66	41.62	161.02	43.44	78.29	107.18
4	1461.87	678.755	150.62	40.98	160.35	40.18	81.53	106.5
5	1485.42	660.74	150.65	40.21	186.04	36.18	78.05	108.4
6	1485.34	678.905	150.55	40.82	176.6	36.37	78.29	108.75
7	1477.70	668.609	150.65	40.67	161.02	36.28	92.09	106.16
8	1484.01	656.2	150.65	41.3	161.02	37.18	94.81	105.99
9	1458.32	679.355	150.66	41.14	162.39	37.49	95.17	110.01
10	1484.34	675.035	149.48	44.2	163.77	37.91	95.17	107.18
11	1459.59	678.985	149.31	39.76	160.35	40.18	95.17	110.74
12	1473.17	680.827	149.48	42.11	161.02	39.71	95.17	119.26
13	1202.26	679.488	149.47	40.82	161.02	36.18	94.81	106.16
14	1283.93	678.666	149.45	40.98	161.02	38.79	95.17	117.79
15	1200.83	661.582	148.06	40.67	161.02	40.92	95.17	106.5
16	1201.83	667.007	149.04	39.91	160.35	39.24	95.17	106.33
17	1201.19	679.832	145.71	39.32	194.04	43.02	78.29	106.33
18	1459.60	671.305	149.12	40.98	196.03	36.18	78.29	112.42
19	1368.67	679.703	146.75	38.75	193.55	37.91	78.29	106.5
20	1202.82	679.34	146.16	39.03	195.53	36.18	78.53	106.5
21	1424.17	678.056	144.61	39.76	196.03	36.28	78.05	116.35
22	1450.49	679.731	148.95	41.62	196.53	42.47	77.58	105.99
23	1320.51	677.679	147.17	40.67	196.03	39.83	81.53	115.34
24	1182.77	678.727	144.56	42.44	196.53	36.18	76.65	117.37
25	1279.18	672.571	148.52	41.78	161.02	36.28	95.17	116.96
26	1202.41	605.06	148.89	40.67	161.02	37.7	93.43	106.5
27	1192.62	603.506	149.1	40.67	159.02	37.7	94.81	107.7
28	1199.95	680.891	146.26	39.61	196.53	36.18	94.81	110.38
29	1202.46	674.217	149.06	39.76	196.03	37.7	78.29	128.95
30	1202.22	541.573	144.33	41.3	160.35	39.36	78.53	106.5
Min	1182.77	541.57	144.33	38.75	159.02	36.18	76.65	105.49
Mean	1326.52	665.63	148.64	40.85	174.05	38.19	86.10	110.42
Median	1302.22	677.85	149.11	40.745	161.705	37.7	81.53	107.44
Maximum	1485.42	680.89	150.68	44.2	196.53	43.44	95.17	128.95
Std. Deviation	124.91	30.16	2.02	1.27	16.43	2.19	8.28	5.65
Confidence Int.	44.70	10.79	0.72	0.45	5.88	0.79	2.96	2.02

Table 6.16: Descriptive Statistics of Compute Power Data Generated by standard benchmarking applications

		Min	Mean	Median	Maximum	Std. Deviation	Confidence Int.
Dhrystone	Local	352.71	372.25	374.78	379.01	7.37	2.64
	Local_PMCO	341.84	365.02	367.37	371.42	7.68	2.75
	PMCO	1182.77	1326.52	1302.22	1485.42	124.91	44.70
Whetstone	Local	168.61	171.22	170.73	173.72	1.53	0.55
	Local_PMCO	152.87	166.71	167.20	170.55	3.64	1.30
	PMCO	541.57	665.63	677.87	680.89	30.16	10.79
Linpack (MFLOPS)	Local	32.34	36.62	36.59	37.54	0.92	0.33
	Local_PMCO	33.12	35.74	35.92	36.67	0.71	0.26
	PMCO	144.33	148.64	149.11	150.68	2.02	0.72
Scimark-FFT (MFLOPS)	Local	8.50	8.73	8.69	9.06	0.16	0.06
	Local_PMCO	8.57	8.72	8.71	9.01	0.09	0.03
	PMCO	38.75	40.86	40.75	44.20	1.27	0.45
Scimark-SOR (MFLOPS)	Local	52.93	53.25	53.23	53.37	0.12	0.04
	Local_PMCO	52.93	53.24	53.23	53.37	0.11	0.04
	PMCO	159.02	174.05	161.71	196.53	16.44	5.88
Scimark-MonteCarlo (MFLOPS)	Local	10.72	10.74	10.75	10.75	0.02	0.01
	Local_PMCO	10.65	10.72	10.72	10.79	0.03	0.01
	PMCO	36.18	38.19	37.70	43.44	2.20	0.79
Scimark-SparseMatMult (MFLOPS)	Local	23.10	23.22	23.19	23.36	0.05	0.02
	Local_PMCO	23.10	23.22	23.19	23.27	0.05	0.02
	PMCO	76.65	86.10	81.53	95.17	8.28	2.96
Scimark-LU (MFLOPS)	Local	33.91	33.96	33.96	33.98	0.02	0.01
	Local_PMCO	33.86	33.95	33.94	33.98	0.03	0.01
	PMCO	105.49	110.42	107.44	128.95	5.65	2.02

analysis which is presented as follows.

Descriptive statistics of results in local, Local_PMCO and PMCO execution modes, including minimum, maximum, and mean compute power measured using benchmarking workloads are summarized in Table 6.16. As descriptive statistics in the Table shows, executing the task on remote computing devices instead of local device can improve the compute capabilities significantly, depending upon the compute capabilities of the remote computing infrastructure

As presented in Table 6.16, in dhrystone, the minimum increase in compute power using the PMCO mode are as high as 317.73% from the mean compute power in local execution. The maximum increase for dhrystone MIPS is approximately 399.038% from the mean value of dhrystone MIPS on the local execution. Similarly, for the benchmarking workloads, the mean compute power measured in PMCO execution environment, is at least three times more than the local execution which is remarkably high. Similarly, to ensure the lightweight feature of our proposed PMCO, we infer that it should light over-

head, which is also evident. For example, if the dhrystone workload when executed locally using PMCO components only deteriorates as low as 0.33% and as high as 8.17%.

As described in the previous chapter, execution of each benchmark is repeated thirty times to enhance the reliability of performance evaluation. So, data plotted in Figure 6.16 and 6.17 are mean compute power measured from the benchmark workloads for both local, Local_PMCO and PMCO execution modes. Each diagonal bricks bar in the Figure, represents the mean value of compute power measured using PMCO mode of thirty iterations for each corresponding benchmark workload. Similarly, each diagonal strips bar represents the mean compute power measured using Local_PMCO mode of execution, while each checker patterned bar represents the corresponding compute power for local execution.

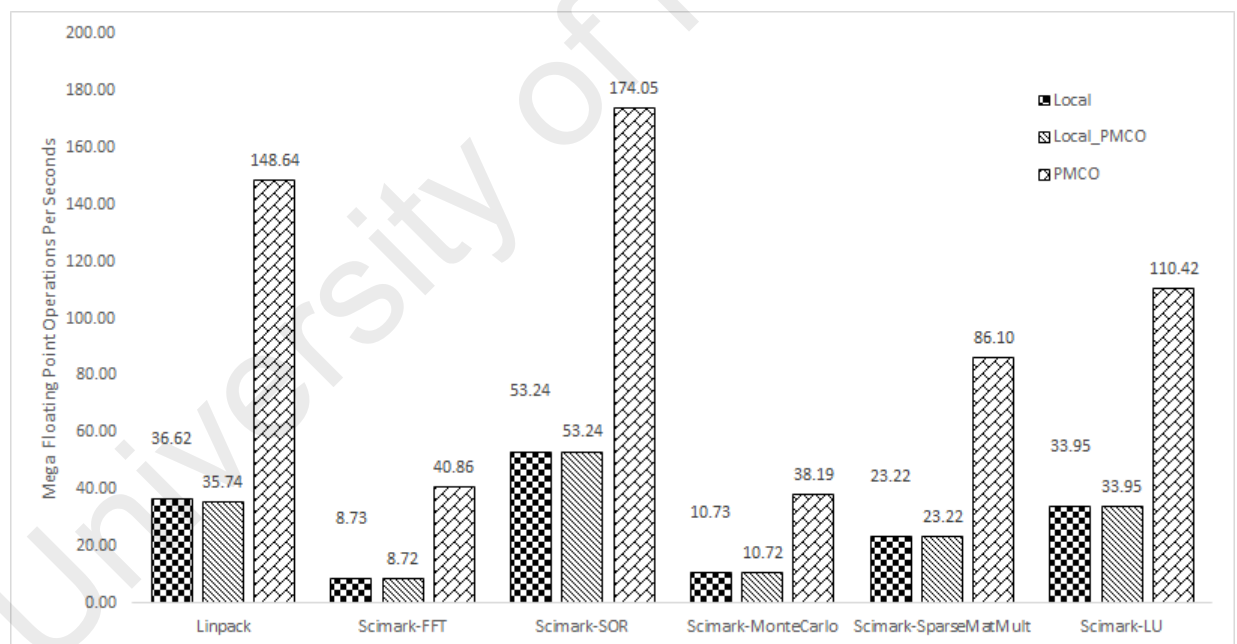


Figure 6.16: Mean MFLOPS values for 30 observations of Linpack and Scimark benchmarks generated via Local, Local_PMCO and PMCO executions.

The graph clearly depicts the increase in compute power which is disposed to the local mobile device through PMCO mode of execution. The graph also clearly demonstrates that there is very less margin of degradation if a benchmark is executed locally

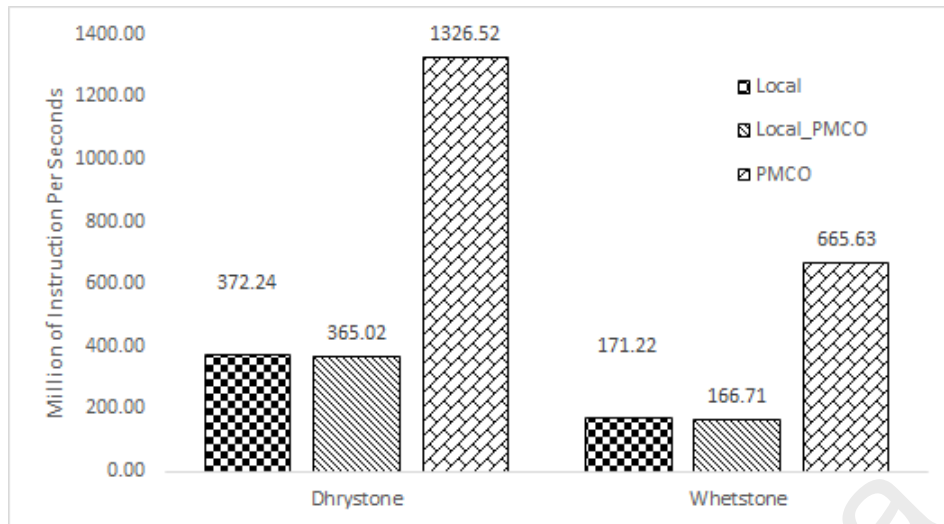


Figure 6.17: Mean MIPS and MWIPS values for 30 observations of Dhrystone and Whetstone benchmarks generated via Local, Local_PMCO and PMCO executions.

using the PMCO components.

From the perspective of inferential statistics, we have applied a one-tailed t-test over the results of local compute and compute power improved through PMCO. The t-test will allow us to verify the significance of the results and decrease the possibility that an observation of data does not occur due to chance. The null hypothesis H_0 is that there is no improvement in the compute power when the benchmark is executed locally over the when the benchmark is executed using the PMCO framework. Table 6.17, presents the details of the t-test over the means of compute power of the local device and the compute power when the benchmarks are executing using the proposed framework on the experimental setup discussed in section 5.1. The p -value is observed around 0.041 which according to the central theorem is less than 0.05 for 95% confidence interval. Hence we reject the null hypothesis H_0 and infer that there is a significant improvement in the compute power of local mobile device when utilizing the PMCO based framework. Lastly, due to the no correlation between the independent variable (the computational intensity of the each benchmark workload over the course of the experimental observation of thirty runs), it is not possible to produce a linear model for the compute power. But still

Table 6.17: t-Test: Paired Two Sample for Mean Compute power in local execution mode and PMCO mode

	Local	PMCO
Mean	88.74375	323.80125
Variance	15847.70277	205580.2379
Pearson Correlation	0.998857957	
Hypothesized Mean Difference	0	
df	7	
t Stat	-2.028686764	
P(T<=t) one-tail	0.041035829	
t Critical one-tail	1.894578605	

the t-test gives us enough verification about the improvement measured in the proposed technique.

6.2 Comparative evaluations

The results of our comparative study are presented in this section. The discussion in section demonstrates that the performance of our framework is comparatively more efficient than that of offloading in using contemporary techniques of Code Offloading, Thread Synchronization. The comparisons are done based on the qualitative as the effect in execution time may not be different from other systems as it is a function of compute power of the remote computing power. Furthermore, the code migration and thread synchronization are different paradigms, and the comparison would be biased, as they need application level virtualization, which on its own has performance bottlenecks as compared with native execution which is the focus of this research.

However, from the compute overhead point view Table 6.18 presents a comparison when a benchmark workload is executed locally using the components of proposed model and one of the contemporary thread synchronization method (i.e. COMET). The values in the table are based upon 95% confidence interval of thirty observations. Now the table testifies that the proposed PMCO approach is significantly better than the COMET when the network is disconnected or the application cannot be offloaded as we did not modify

Table 6.18: Comparison when benchmark are executed locally using COMET and PMCO

	COMET Local	PMCO Local
SciMark (FFT)	4.23	8.72
SciMark (SOR)	11.647	53.24
SciMark (MonteCarlo)	0.717	10.72
SciMark (Sparse MatMult)	6.806	23.22
SciMark (LU)	7.785	33.95

any system default components to enable migration. This phenomenon is also presented in Figure 6.18

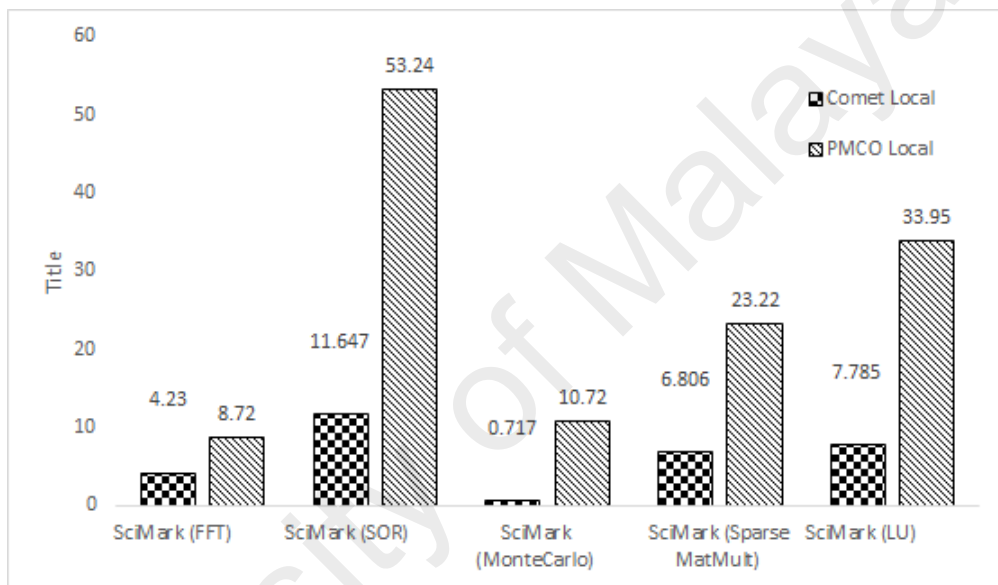


Figure 6.18: Comparison between Comet and PMCO when applications cannot be offloaded.

The graph clearly, shows the performance degradation caused by COMET when the applications are not offloaded. Besides the performance difference of application level virtualization and native execution, the performance difference 2 times to almost 10 times when we see the results of Scimark Monte Carlo benchmark.

Moreover, the amount data transfer can be considered as a factor in reducing the offloading expense and can be used as comparison criterion with other offloading systems. Now from a data transmission point of view, the comparison between the proposed model and COMET is purely qualitative as our analysis in Chapter 3 shows that COMET's communication with the server cannot be controlled as they are synchronizing the shared

Table 6.19: Comparison between conventional code offloading and PMCO

	Code Offloading	PMCO	Difference
Mat. Mult. (300x300)	1084116	4391059.067	-3306943.067
Mat. Mult. (400x400)	1924116	4875541.233	-2951425.233
Mat. Mult. (500x500)	3004116	5492616.867	-2488500.867
Mat. Mult. (600x600)	4324116	6253364.567	-1929248.567
Mat. Mult. (700x700)	5884116	7162103.2	-1277987.2
Mat. Mult. (800x800)	7684116	8190105.033	-505989.0333
Mat. Mult. (900x900)	9724116	9368659.467	355456.5333
Mat. Mult. (1000x1000)	12004116	10687204.8	1316911.2

memory all time. However, the proposed system utilize the network on-demand basis.

When it is required to offload, then offload otherwise stay sleep.

Similarly, when compared with code offloading systems, the results for the amount of data transfer for matrix multiplication workload of different granularity is presented in Table 6.19. The conventional code offloading system will offload the data and code segment both and in our experimental setting, the size of the code segment is (4116 bytes) while the rest is the $N^3 \times 3 \times 4$. Where N is the dimension of the matrix, the factor 3 is the number of matrices (two input and one output), while 4 is the number of bytes for a single coefficient. The difference column in the table is calculated using $CodeOffloading - PMCO$. Now the table clearly, demonstrates, the larger the problem instance, the more efficient PMCO becomes. PMCO compress the data and executable checkpoint in a package for delivery. Finally, the overhead both in term of bytes containing metadata and compute time it takes in the serialization process of the code offloading is not added it up. Lastly, the proposed PMCO framework does not need the application binary on the server side make it much secure and independent and loosely coupled as compared with the other traditional systems.

6.3 Conclusion

In this chapter, the results of performance evaluation of our proposed framework via experimental analysis and statistical analysis are reported and illustrated using several t

ables and figures. The results are presented in sections 6.1, and 6.2 are synthesized in section 6.6. Using paired samples t-test, we demonstrated significant time and energy efficiency yield from experimental and statistical modeling when offloading a compute intensive executing process using PMCO over the experimental infrastructure. The performance evaluation of the framework is carried out using workloads of eight intensity levels to effectively highlight the correlation between workloads and time saving as well as energy saving. Although the proposed solution is remarkably effective at all intensity levels, the findings are more significant when the workloads are highly intensive. Time and energy efficiency rates are increasing as the workload intensities are increased.

The evaluation results testified about 44% time efficiency as well as 85% energy efficiency when the execution of the experimental workload is performed outside the mobile device, using the proposal PMCO. The results of empirical setup and statistical modeling are synthesized to demonstrate petty differences between the reported achievements. The supportive results of real-time experiments and statistical modeling unveil lightweight nature of the framework and its usability and successful adoption in real scenarios. Our secondary experiments testified that the lightweight feature of our proposed framework does not deteriorate the performance of the mobile device when it is used to trigger local execution.

CHAPTER 7: CONCLUSION

In this chapter, we present conclusions on the research undertaken in this thesis and highlight the future works. We re-examine the aim and objectives of this research to ensure that they are realized through the work reported in this thesis. We present the contributions of this study and highlight the significance and novelty of the proposed framework.

The remainder of this chapter is as follows. Section 7.1 presents the efforts undertaken to fulfill the aim and objectives of this study. The contributions of the thesis are presented in section 7.2. Significance and limitation of the framework proposed in this research are highlighted in section 7.3. Finally, Section 7.4 presents and future works.

7.1 Retrospection of the research objectives

The problem of computational offloading due to the introduction Android Runtime Environment for efficient execution of compute-intensive mobile applications in the resource-scarce mobile environment because and its impact on application execution time and energy consumption has been investigated and addressed in this thesis. Five objectives were set for the research in Section 1.4. We revisit these objectives and highlight how the research study met the objectives.

The first objective was to review the computational offloading frameworks in MCC for acquiring the insight on the state-of-the-art concerning migration mechanism issues during the execution and migration of mobile application on remote infrastructure. A thematic taxonomy of conducted literature review has been devised to achieve the objective of the literature review. We have studied the state-of-the-art literature from web resources and online digital libraries including IEEE, ACM, Springer, and Elsevier. We have collected and credible research literature in the broader domain of mobile computing and mobile cloud computing and reviewed the literature on computational offloading frameworks. Some features employed by frameworks to optimize the computational offloading

performance have been identified. Qualitative analysis was done to investigate the critical aspects of state-of-the-art computational offloading frameworks performance and to determine the open issues for computational offloading in MCC. The second objective was to investigate the computational offloading in general and then investigate the major existing computational offloading mechanisms that can be used to migrate a mobile application to the remote infrastructure. To achieve this objective, we have simulated the computational offloading benefit analysis and provide insights over the limitation of computational offloading. Furthermore, we have implemented the classified computational mechanisms; the analysis shows that under very realistic conditions, the existing frameworks inefficiencies. The research gap identified in the review is experimental demonstrated and proof of concept experiment is performed. This established the problem of process migration based computational offloading in MCE as non-trivial.

The third objective was to design and develop the solution for process migration based computational offloading to minimize the execution time, energy consumption and increase the compute power of the local mobile device. A process migration based computational offloading framework along with an offloading algorithm has been proposed to address the issue of application migration on the cloud server and vice versa. The offloading algorithm exchanges the process using checkpointed states of the process under execution with the remote server and the mobile device during the offloading transaction.

For the fourth objective, we successfully attain our objective by evaluating the performance of proposed MCC framework via series of experiments in real MCC environment using Android based smartphones. We performed several experiments for evaluation purpose. Firstly, we performed a series of experiments developed using experimental workloads on an Android-based smartphone. Application execution time and consumed energy of the compute intensive mobile application in local and remote execution modes are measured, collected, analyzed, and synthesized to demonstrate the performance of

the proposed framework. The results of this performance evaluation experiments unveiled more than 56% time saving and more than 84% energy saving which is significant. We further evaluated the lightweight feature of the proposed framework by measuring the energy consumption and application execution time overhead caused by the framework. The overhead is observed by comparing the results of workloads executed in a local execution with workload executed through the environment of the proposed framework, and the overhead seems to be less than 5%. Furthermore, the overhead when compared with COMET which is around ten times worse when an application is executed locally using COMET. This feature, in turn, testifies the lightweight characteristics of the proposed solution.

For the final objective, the performance evaluation results are validated using statistical modeling. We use linear regression analysis as the most predominant observation-based modeling approach to derive the statistical models of execution time and energy consumption of the mobile application on local and PMCO execution modes. We create a data set using independent replication method to train the regression models and derive the statistical models. The statistical models are validated using split-sample approach, and the results are used to verify the model's validity. We synthesize the results of time and energy generated via experimentation and statistical modeling to validate performance evaluation results of our framework. The results advocate reliability and validity of the proposed framework and ensure that we achieved our aim.

Considering successful accomplishment of our objectives in this thesis, we conclude that the aim of this research is successfully realized.

7.2 Research contributions

In this thesis, we have produced several contributions to the body of knowledge that are described as follows.

7.2.1 Taxonomy of computational offloading issues in the client sub-system

In this thesis, we produced a comprehensive taxonomy on the technical issues that arises in mobile devices. Some of these issues are the root causes and motivation for the offloading of the mobile applications to the remote computing infrastructure. While some the issues in the taxonomy present the technical issues that needs to addressed while developing a computational offloading system for augmentation of mobile devices. The taxonomy is presented in Section 2.1.

7.2.2 Taxonomy of computational offloading functions

In this thesis, we produced a comprehensive survey of computation offloading mechanisms that are published in the literature. We have reviewed computational offloading frameworks by critically reviewing literary articles extracted from major scholarly digital libraries that are presenting the state-of-the-art research to devise the taxonomy of the computation offloading functions in MCC. The taxonomy based on the functional aspects of the computational offloading solutions is first of its kind in the literature, and it as one of the significant contributions of this research. The taxonomy is presented in Chapter 2, Section 2.2.

7.2.3 Empirical analysis of existing computation offloading solutions

We contributed to the body of knowledge by investigating the benefit analysis of computational offloading in general and the factors affecting the computational benefits and their implications of utilizing computational offloading in mobile empowerment solutions. We perform comprehensive analytical and experimental analyses on the classified computational offloading migration mechanism to augment the computational resource of mobile devices. We investigated and demonstrated significant limitations of utilizing the existing migration mechanism during the computational offloading transaction. The analytical experiment revealed the impact on the augmentation performance, whereas

the volume of data transfer has remarkable impacts on the efficacy of augmentation of resource-constraint mobile devices using remote computing resources.

7.2.4 Lightweight process migration based computational offloading framework

In this thesis, we presented one of the earliest works to leverage process migration employing checkpoint/restart in design and development of a lightweight computational offloading framework in MCC. Our proposed framework exploits computational powers of available homogeneous remote computing devices available in the mobile cloud environment and aims to achieve efficiency in execution of compute intensive mobile applications. This framework achieves its aim by transparently checkpointing the process that needs to be offloaded. Once checkpointed the checkpoint package is transferred to the nearby connected resource rich computing device and restart the execution over there. Afterward, the execution at the server-side reach a synchronization point it is again checkpointed on the server and the updated checkpoint image is then sent to the requesting mobile device to restart the process. The proposed framework is loosely coupled from the availability of application source code or binary on the server side as the checkpoint package itself contain the binary instruction of the checkpointed process. The proposed lightweight computational offloading framework is presented in Chapter 4, as an effort to deploy checkpoint-restart mechanism for mobile device augmentation. The proposed framework could address high computational and communication limitations of accessing remote resources, by providing high scalability features. Using extensive analysis and experimentation we observed capabilities of proposed solution can increase the network life of wireless network based computational systems.

7.2.5 Evaluation and validation of the proposed solution

We contributed to the knowledge by implementing, evaluating, and validating the performance of our lightweight process migration based computational offloading framework to

demonstrate its reliable and valid functionality, and significance. A detailed description of the systems and data that are used to evaluate and validate the proposal are reported in Chapter 5, and the results of performance evaluation and validation are presented in Chapter 6. The results reveal feasibility and functionality of our proposed framework. The results also verify that utilizing our proposed framework can achieve execution efficiency and save execution time and energy as significant as 50% in average. The results explicitly ensure feasibility and lightweight characteristics of the framework and advocate that objectives of this study are fulfilled and the aim is realized.

7.3 Significance and limitations of the proposed solution

Designing our framework based on process migration based computational offloading framework for mobile device augmentations has the following benefits:

- **Generalized:** Process migration based upon checkpoint/restart provide the ability to handle both migration-aware and unaware applications. Migration-aware applications have been coded to take advantage of process migration explicitly. Dynamic process migration can automatically migrate these applications to save mobile device energy and improve its computational power.
- **Ability offloading application without needing application binary on the server side:** Checkpoint/Restore methods in user space package the application binary and state into one package which can be transferred to the remote device and executed directly. Removing the requirement of application binary or source on the remote computing platform, as the checkpoint image is a package of both the application plus its data.
- **Portability:** Process migration can be performed almost on every type of computing infrastructure with minimal modification to the operating system kernel. So the

proposed computational offloading system can be ported to any of the supported hardware and software platforms which has the checkpoint features.

- **Loose Coupling:** Process migration based computational is loosely coupled with the underlying mobile device environment so if an application which does not need to be migrated can still be executed locally with minimal performance losses.
- **Saving Computations:** As an underlying principle of process migration the only checkpoint/restart in mobile devices can also help mobile applications to checkpointed in critical battery conditions and then later restarted either locally or remotely.
- **Resource localization:** Mobile applications which are distant from the device which has the data that the application is using tend to spend most of their time in performing communication between the mobile device and remote computing device for the sake of accessing the data. The process migration framework can be used to migrate the process closer to the data that it is processing, thereby ensuring it spends most of its time doing useful work.
- **User Preference:** Our framework, is based upon preference based system defined by the user to control the behavior of the computational offloading system. Further, most of the existing offloading mechanism uses HTTP-based communication while we focused to use a single TCP socket to get some improvement using a single connection.

Additionally, our framework based on Checkpoint/Restore in user space has a shortcoming also, but it applies to any process migration based mechanism and also applies to

VM Migration:

- Platform Dependent: Our proposed checkpoint restore based computational mechanism is platform dependent meaning that if the mobile device hardware platform is ARM, then you need an ARM hardware platform on the server side. This platform independence also applies to VM Migration based computational offloading systems. The proposed framework requires a homogeneous hardware and software environments in the mobile device and remote compute infrastructure because the checkpointed process state is platform dependent not agnostic. However, to enable the process migration based computational offloading between heterogeneous systems one can use a certain type of manual transformation of a process state from one architecture to another (Chanchio & Sun, 2002). However, this will also have a mapping overhead in term of compute power along with differently compiled application for every platform. Second, emulating the alien platform over the available platform using binary translators, such as QEMU (Bellard, 2005) which can also bring a lot of performance degradation and the purpose of offloading will not be achieved.

7.4 Future prospects

Through our theoretical and experimental analysis on computational offloading in general and our designed framework, we know that there are still some issues which can be studied further in the future using the lens of process migration in computational offloading for mobile systems. Here we present two important issues that needs to addressed in order establish an optimal process migration based computational offloading ecosystem.

- In MCE, due to intermittent connectivity and non-seamless wireless coverage, and mobility of the end user remote computing infrastructure (such as Clouds, Cloudlets, Ad-Hoc Mobile Nodes) may not be available or disconnected once an application is migrated to any of the mentioned remote computing infrastructure (Wang et al.,

2014). However, this disconnection may result in loss of computation and deteriorate the end user QoS/QoE requirements. Furthermore, due to the mobility of the user a device cannot re-establish the connection with one-hop augmentation devices such as cloudlet or ad-hoc mobile nodes. To save the computation once performed mobility assisted server-to-server computational offloading mechanism is the need of the day for MCEs.

- To attain optimal performance from any computational offloading system, the code must be partitioned to identify the intensive, non-intensive, local resource and user interacting fragments. To allow the already developed application to take advantage of the mobile cloud using computational offloading those application must be partitioned so that the migration/synchronization points are identified. These partitioning should be automatic such as employed in CloneCloud (Chun et al., 2011) and its variants to transform any mobile applications to a mobile cloud application. The partitioning mechanism should exploit and modify the AHOT compilation process of ART, to modify, annotate and add special migration primitives to the DEX intermediate code before transforming it into machine dependent binaries. These migration markers should be then used to trigger the checkpoint-restart the application on different nodes in mobile cloud systems.

7.5 Scholarly publications

The following is the list of accepted articles during the course of this research.

- **Yousafzai, Abdullah**, Abdullah Gani, Rafidah Md Noor, Mehdi Sookhak, Hamid Talebian, Muhammad Shiraz, and Muhammad Khurram Khan. "Cloud resource allocation schemes: review, taxonomy, and opportunities." *Knowledge and Information Systems* (2016): 1-35.

- **Yousafzai, Abdullah**, Abdullah Gani, Rafidah Md Noor, Anjum Naveed, Raja Wasim Ahmad, and Victor Chang. "Computational offloading mechanism for native and android runtime based mobile applications." *Journal of Systems and Software* 121 (2016): 28-39.
- **Yousafzai, Abdullah**, Victor Chang, Abdullah Gani, and Rafidah Md Noor. "Directory-based incentive management services for ad-hoc mobile clouds." *International Journal of Information Management* (2016).
- **Yousafzai, Abdullah**, Victor Chang, Abdullah Gani, and Rafidah Md Noor. "Multimedia augmented m-learning: Issues, trends and open challenges." *International Journal of Information Management* 36, no. 5 (2016): 784-792.

University of Malaya

REFERENCES

- Abolfazli, S., Sanaei, Z., Ahmed, E., Gani, A., & Buyya, R. (2014). Cloud-based augmentation for mobile devices: Motivation, taxonomies, and open challenges. *IEEE Communications Surveys Tutorials*, 16(1), 337-368. doi: 10.1109/SURV.2013.070813.00285
- Afonso, V., Bianchi, A., Fratantonio, Y., Doupé, A., Polino, M., de Geus, P., . . . Vigna, G. (2016). Going native: Using a large-scale analysis of android apps to create a practical native-code sandboxing policy.
- Ahmed, E., Gani, A., Khan, M. K., Buyya, R., & Khan, S. U. (2015). Seamless application execution in mobile cloud computing: Motivation, taxonomy, and open challenges. *Journal of Network and Computer Applications*, 52, 154-172.
- Alliance, O. (2009). Osgi alliance: Osgi service platform, core specification, release 4, version 4.2. *OSGi Alliance*.
- AlZain, M. A., Pardede, E., Soh, B., & Thom, J. A. (2012). Cloud computing security: from single to multi-clouds. In *System science (hicss), 2012 45th hawaii international conference on* (pp. 5490-5499).
- Ansel, J., Arya, K., & Cooperman, G. (2009). Dmtpc: Transparent checkpointing for cluster computations and the desktop. In *Parallel & distributed processing, 2009. ipdps 2009. ieee international symposium on* (pp. 1-12).
- Bellard, F. (2005). Qemu, a fast and portable dynamic translator. In *Usenix annual technical conference, freenix track* (pp. 41-46).
- Bent, K. (2012). *Arm snags 95 percent of smartphone market, eyes new areas for growth*. <http://www.crn.com/news/components-peripherals/240003811/arm-snags-95-percent-of-smartphone-market-eyes-new-areas-for-growth.htm>. Retrieved from <http://www.crn.com/news/components-peripherals/240003811/arm-snags-95-percent-of-smartphone-market-eyes-new-areas-for-growth.htm> ((Visited on 09/10/2015))
- Bourdon, A., Nouredine, A., Rouvoy, R., & Seinturier, L. (2013). Powerapi: A software library to monitor the energy consumed at the processlevel. *ERCIM News*, 2013(92).
- Bourguiba, M., Agha, K., & Haddadou, K. (2012). Improving networking performance in virtual mobile clouds. In *Network of the future (nof), 2012 third international conference on the* (pp. 1-6).
- Buckley, S. (2013). *Art experiment in android kitkat improves battery life and speeds up apps*. <http://www.engadget.com/2013/11/06/new-android-runtime-could-improve-battery-life/>. ((Visited on 03/30/2015))
- Chanchio, K., & Sun, X.-H. (2002). Data collection and restoration for heterogeneous process migration. *Software: Practice and Experience*, 32(9), 845-871.

- Chao, P.-C., & Sun, H.-M. (2013). Multi-agent-based cloud utilization for the it office-aid asset distribution chain: An empirical case study. *Information Sciences*, 245, 255–275.
- Chen, E., & Itoh, M. (2010, June). Virtual smartphone over ip. In *World of wireless mobile and multimediantworks (wowmom), 2010 ieee international symposium on a* (p. 1-6). doi: 10.1109/WOWMOM.2010.5534992
- Chun, B.-G., Ihm, S., Maniatis, P., Naik, M., & Patti, A. (2011). Clonecloud: Elastic execution between mobile device and cloud. In *Proceedings of the sixth conference on computer systems* (pp. 301–314). New York, NY, USA: ACM. Retrieved from <http://doi.acm.org/10.1145/1966445.1966473> doi: 10.1145/1966445.1966473
- Chun, B.-G., & Maniatis, P. (2009). Augmented smartphone applications through clone cloud execution. In *Hotos* (Vol. 9, pp. 8–11).
- Cohen, J. (2008, March). Embedded speech recognition applications in mobile phones: Status, trends, and challenges. In *Acoustics, speech and signal processing, 2008. icassp 2008. ieee international conference on* (p. 5352-5355). doi: 10.1109/ICASSP.2008.4518869
- Committee, T. I. S., et al. (2001). Executable and linkable format (elf). *Specification, Unix System Laboratories*.
- Courtois, P.-J., Heymans, F., & Parnas, D. L. (1971). Concurrent control with “readers” and “writers”. *Communications of the ACM*, 14(10), 667–668.
- Cuervo, E., Balasubramanian, A., Cho, D.-k., Wolman, A., Saroiu, S., Chandra, R., & Bahl, P. (2010). Maui: making smartphones last longer with code offload. In *Proceedings of the 8th international conference on mobile systems, applications, and services* (pp. 49–62).
- Dean, J., & Ghemawat, S. (2008). Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1), 107–113.
- 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), 1587–1611.
- Dou, A., Kalogeraki, V., Gunopulos, D., Mielikainen, T., & Tuulos, V. H. (2010). Misco: a mapreduce framework for mobile systems. In *Proceedings of the 3rd international conference on pervasive technologies related to assistive environments* (p. 32).
- Drissi, M., & Oumsis, M. (2015). Performance evaluation of multi-criteria vertical handover for heterogeneous wireless networks. In *Intelligent systems and computer vision (iscv), 2015* (pp. 1–5).
- F., E. (2014). *64-bit android* and android run time*. <https://software.intel.com/en-us/android/articles/64-bit-android-and-android-run-time>. ((Visited on 06/30/2015))

- Flinn, J., & Mao, Z. M. (2011). Can deterministic replay be an enabling tool for mobile computing? In *Proceedings of the 12th workshop on mobile computing systems and applications* (pp. 84–89). New York, NY, USA: ACM. Retrieved from <http://doi.acm.org/10.1145/2184489.2184507> doi: 10.1145/2184489.2184507
- Flores, H., Srirama, S., & Buyya, R. (2014). Computational offloading or data binding? bridging the cloud infrastructure to the proximity of the mobile user. In *Mobile cloud computing, services, and engineering (mobilecloud), 2014 2nd ieee international conference on* (p. 10-18). doi: 10.1109/MobileCloud.2014.15
- Forbes. (2013). *Arm holdings and qualcomm: The winners in mobile.* <http://www.forbes.com/sites/darcytravlos/2013/02/28/arm-holdings-and-qualcomm-the-winners-in-mobile/>. Retrieved from <http://www.forbes.com/sites/darcytravlos/2013/02/28/arm-holdings-and-qualcomm-the-winners-in-mobile/> ((Visited on 09/10/2015))
- Gentry, C., et al. (2009). Fully homomorphic encryption using ideal lattices. In *Stoc* (Vol. 9, pp. 169–178).
- Gomez, L., Neamtiu, I., Azim, T., & Millstein, T. (2013). Reran: Timing- and touch-sensitive record and replay for android. In *Proceedings of the 2013 international conference on software engineering* (pp. 72–81). Piscataway, NJ, USA: IEEE Press. Retrieved from <http://dl.acm.org/citation.cfm?id=2486788.2486799>
- Google. (2013). *Art and dalvik.* <https://source.android.com/devices/tech/dalvik/>. Retrieved from <https://source.android.com/devices/tech/dalvik/> ((Visited on 03/30/2015))
- Gordon, M. S., Jamshidi, D. A., Mahlke, S. A., Mao, Z. M., & Chen, X. (2012). Comet: Code offload by migrating execution transparently. In *Osdi* (pp. 93–106).
- Gu, Z., & Zhao, Q. (2012). A state-of-the-art survey on real-time issues in embedded systems virtualization.
- Hemminger, S. (2005). Network emulation with netem.
- 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), 573 - 587. Retrieved from <http://www.sciencedirect.com/science/article/pii/S0898122111009084> (Advances in context, cognitive, and secure computing) doi: <http://dx.doi.org/10.1016/j.camwa.2011.10.044>
- Kalic, G., Bojic, I., & Kusek, M. (2012). Energy consumption in android phones when using wireless communication technologies. In *Mipro, 2012 proceedings of the 35th international convention* (pp. 754–759).
- Kemp, R., Palmer, N., Kielmann, T., & Bal, H. (2012). Cuckoo: A computation offloading framework for smartphones. In M. Gris & G. Yang (Eds.), *Mobile computing, applications, and services* (Vol. 76, p. 59-79). Springer Berlin Heidelberg. Retrieved from http://dx.doi.org/10.1007/978-3-642-29336-8_4 doi: 10.1007/978-3-642-29336-8_4

- Kosta, S., Aucinas, A., Hui, P., Mortier, R., & Zhang, X. (2012, March). Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading. In *Infocom, 2012 proceedings ieee* (p. 945-953). doi: 10.1109/INFCOM.2012.6195845
- Kovachev, D., Cao, Y., & Klamma, R. (2012). Augmenting pervasive environments with an xmpp-based mobile cloud middleware. In M. Gris & G. Yang (Eds.), *Mobile computing, applications, and services* (Vol. 76, p. 361-372). Springer Berlin Heidelberg. Retrieved from http://dx.doi.org/10.1007/978-3-642-29336-8_25 doi: 10.1007/978-3-642-29336-8_25
- Kovachev, D., Yu, T., & Klamma, R. (2012, July). Adaptive computation offloading from mobile devices into the cloud. In *Parallel and distributed processing with applications (ispa), 2012 ieee 10th international symposium on* (p. 784-791). doi: 10.1109/ISPA.2012.115
- Kumar, K., Liu, J., Lu, Y.-H., & Bhargava, B. (2013). A survey of computation offloading for mobile systems. *Mobile Networks and Applications*, 18(1), 129–140.
- Kumar, K., & Lu, Y.-H. (2010). Cloud computing for mobile users: Can offloading computation save energy? *Computer*, 43(4), 51–56.
- Lawton, K. P. (1996). Bochs: A portable pc emulator for unix/x. *Linux Journal*, 1996(29es), 7.
- Lee, B.-D. (2012). A framework for seamless execution of mobile applications in the cloud. In Z. Qian, L. Cao, W. Su, T. Wang, & H. Yang (Eds.), *Recent advances in computer science and information engineering* (Vol. 126, p. 145-153). Springer Berlin Heidelberg. Retrieved from http://dx.doi.org/10.1007/978-3-642-25766-7_20 doi: 10.1007/978-3-642-25766-7_20
- Liu, J., Ahmed, E., Shiraz, M., Gani, A., Buyya, R., & Qureshi, A. (2015, February). Application partitioning algorithms in mobile cloud computing. *J. Netw. Comput. Appl.*, 48(C), 99–117. Retrieved from <http://dx.doi.org/10.1016/j.jnca.2014.09.009> doi: 10.1016/j.jnca.2014.09.009
- Lloret, J., Garcia, M., Tomas, J., & Rodrigues, J. J. (2014). Architecture and protocol for intercloud communication. *Information Sciences*, 258, 434–451.
- Luo, H., & Shyu, M.-L. (2011). Quality of service provision in mobile multimedia-a survey. *Human-centric computing and information sciences*, 1(1), 1–15.
- Mahmoodi, S. E., Uma, R. N., & Subbalakshmi, K. P. (2016a). Optimal joint scheduling and cloud offloading for mobile applications. *IEEE Transactions on Cloud Computing*, PP(99), 1-1. doi: 10.1109/TCC.2016.2560808
- Mahmoodi, S. E., Uma, R. N., & Subbalakshmi, K. P. (2016b). Optimal joint scheduling and cloud offloading for mobile applications. *IEEE Transactions on Cloud Computing*, PP(99), 1-1. doi: 10.1109/TCC.2016.2560808
- Marinelli, E. E. (2009). *Hyrax: cloud computing on mobile devices using mapreduce*

(Unpublished master's thesis). School of Computer Science.

- Mell, P., & Grance, T. (2011). The nist definition of cloud computing.
- Nethercote, N., & Seward, J. (2007). Valgrind: a framework for heavyweight dynamic binary instrumentation. In *Acm sigplan notices* (Vol. 42, pp. 89–100).
- Plank, J. S., Beck, M., Kingsley, G., & Li, K. (n.d.). *Libckpt: Transparent checkpointing under unix*.
- Portokalidis, G., Homburg, P., Anagnostakis, K., & Bos, H. (2010). Paranoid android: Versatile protection for smartphones. In *Proceedings of the 26th annual computer security applications conference* (pp. 347–356). New York, NY, USA: ACM. doi: 10.1145/1920261.1920313
- Rahimi, M. R., Ren, J., Liu, C. H., Vasilakos, A. V., & Venkatasubramanian, N. (2014). Mobile cloud computing: A survey, state of art and future directions. *Mobile Networks and Applications*, 19(2), 133–143.
- Ranjan, R., & Buyya, R. (2009). Decentralized overlay for federation of enterprise clouds. *Handbook of Research on Scalable Computing Technologies*, 191.
- Rellermeyer, J. S., Alonso, G., & Roscoe, T. (2007). R-osgi: distributed applications through software modularization. In *Proceedings of the acm/ifip/usenix 2007 international conference on middleware* (pp. 1–20).
- Rellermeyer, J. S., Riva, O., & Alonso, G. (2008). Alfredo: an architecture for flexible interaction with electronic devices. In *Proceedings of the 9th acm/ifip/usenix international conference on middleware* (pp. 22–41).
- Richardson, T., Stafford-Fraser, Q., Wood, K. R., & Hopper, A. (1998). Virtual network computing. *IEEE Internet Computing*, 2(1), 33–38.
- Saab, S. A., Saab, F., Kayssi, A., Chehab, A., & Elhajj, I. H. (2015). Partial mobile application offloading to the cloud for energy-efficiency with security measures. *Sustainable Computing: Informatics and Systems*, -. Retrieved from <http://www.sciencedirect.com/science/article/pii/S2210537915000396> doi: <http://dx.doi.org/10.1016/j.suscom.2015.09.002>
- Satyanarayanan, M., Bahl, P., Caceres, R., & Davies, N. (2009, Oct). The case for vm-based cloudlets in mobile computing. *Pervasive Computing, IEEE*, 8(4), 14-23. doi: 10.1109/MPRV.2009.82
- Shi, C., Habak, K., Pandurangan, P., Ammar, M., Naik, M., & Zegura, E. (2014). Cosmos: Computation offloading as a service for mobile devices. In *Proceedings of the 15th acm international symposium on mobile ad hoc networking and computing* (pp. 287–296). New York, NY, USA: ACM. Retrieved from <http://doi.acm.org/10.1145/2632951.2632958> doi: 10.1145/2632951.2632958
- 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.

- Simanta, S., Ha, K., Lewis, G., Morris, E., & Satyanarayanan, M. (2013). A reference architecture for mobile code offload in hostile environments. In D. Uhler, K. Mehta, & J. Wong (Eds.), *Mobile computing, applications, and services* (Vol. 110, p. 274–293). Springer Berlin Heidelberg. Retrieved from http://dx.doi.org/10.1007/978-3-642-36632-1_16 doi: 10.1007/978-3-642-36632-1_16
- Soyata, T., Muraleedharan, R., Funai, C., Kwon, M., & Heinzelman, W. (2012, July). Cloud-vision: Real-time face recognition using a mobile-cloudlet-cloud acceleration architecture. In *Computers and communications (iscc), 2012 ieee symposium on* (p. 000059-000066). doi: 10.1109/ISCC.2012.6249269
- Surie, A., Lagar-Cavilla, H. A., de Lara, E., & Satyanarayanan, M. (2008). Low-bandwidth vm migration via opportunistic replay. In *Proceedings of the 9th workshop on mobile computing systems and applications* (pp. 74–79). New York, NY, USA: ACM. Retrieved from <http://doi.acm.org/10.1145/1411759.1411779> doi: 10.1145/1411759.1411779
- Vasudevan, N., & Venkatesh, P. (n.d.). Design and implementation of a process migration system for the linux environment.
- Verbelen, T., Simoens, P., Turck, F. D., & Dhoedt, B. (2012). Aiolos: Middleware for improving mobile application performance through cyber foraging. *Journal of Systems and Software, 85(11)*, 2629 - 2639. Retrieved from <http://www.sciencedirect.com/science/article/pii/S0164121212001641> doi: <http://dx.doi.org/10.1016/j.jss.2012.06.011>
- Verbelen, T., Stevens, T., Simoens, P., Turck, F. D., & Dhoedt, B. (2011). Dynamic deployment and quality adaptation for mobile augmented reality applications. *Journal of Systems and Software, 84(11)*, 1871 - 1882. Retrieved from <http://www.sciencedirect.com/science/article/pii/S016412121100166X> (Mobile Applications: Status and Trends) doi: <http://dx.doi.org/10.1016/j.jss.2011.06.063>
- Wang, C., Li, Y., & Jin, D. (2014, Oct). Mobility-assisted opportunistic computation offloading. *IEEE Communications Letters, 18(10)*, 1779-1782. doi: 10.1109/LCOMM.2014.2347272
- Wang, C., & Li, Z. (2004). Parametric analysis for adaptive computation offloading. In *Acm sigplan notices* (Vol. 39, pp. 119–130).
- Whaiduzzaman, M., Gani, A., & Naveed, A. (2014a). An empirical analysis of finite resource impact on cloudlet performance in mobile cloud computing. In *Ceet-2014*.
- Whaiduzzaman, M., Gani, A., & Naveed, A. (2014b, Dec). Pefc: Performance enhancement framework for cloudlet in mobile cloud computing. In *Robotics and manufacturing automation (roma), 2014 ieee international symposium on* (p. 224-229). doi: 10.1109/ROMA.2014.7295892
- Wohlmacher, P. (2000). Digital certificates: a survey of revocation methods. In *Proceedings of the 2000 acm workshops on multimedia* (pp. 111–114).

- Wolski, R., Gurun, S., Krintz, C., & Nurmi, D. (2008, April). Using bandwidth data to make computation offloading decisions. In *Parallel and distributed processing, 2008. ipdps 2008. ieee international symposium on* (p. 1-8). doi: 10.1109/IPDPS.2008.4536215
- Yang, S., Kwon, D., Yi, H., Cho, Y., Kwon, Y., & Paek, Y. (2014, Nov). Techniques to minimize state transfer costs for dynamic execution offloading in mobile cloud computing. *Mobile Computing, IEEE Transactions on*, 13(11), 2648-2660. doi: 10.1109/TMC.2014.2307293
- Yousafzai, A., Chang, V., Gani, A., & Noor, R. M. (2016a). Directory-based incentive management services for ad-hoc mobile clouds. *International Journal of Information Management*, 36(6), 900–906.
- Yousafzai, A., Chang, V., Gani, A., & Noor, R. M. (2016b). Multimedia augmented m-learning: Issues, trends and open challenges. *International Journal of Information Management*, 36(5), 784 - 792. Retrieved from <http://www.sciencedirect.com/science/article/pii/S0268401216302936> doi: <http://dx.doi.org/10.1016/j.ijinfomgt.2016.05.010>
- Yousafzai, A., Gani, A., Noor, R. M., Naveed, A., Ahmad, R. W., & Chang, V. (2016). Computational offloading mechanism for native and android runtime based mobile applications. *Journal of Systems and Software*, 121, 28 - 39. Retrieved from <http://www.sciencedirect.com/science/article/pii/S0164121216301364> doi: <http://doi.org/10.1016/j.jss.2016.07.043>
- Zhao, B., Xu, Z., Chi, C., Zhu, S., & Cao, G. (2012). Mirroring smartphones for good: A feasibility study. In P. S enac, M. Ott, & A. Seneviratne (Eds.), *Mobile and ubiquitous systems: Computing, networking, and services* (Vol. 73, p. 26-38). Springer Berlin Heidelberg. Retrieved from http://dx.doi.org/10.1007/978-3-642-29154-8_3 doi: 10.1007/978-3-642-29154-8_3