PROCESS STATE SYNCHRONIZATION FOR MOBILITY SUPPORT IN MOBILE CLOUD COMPUTING

EJAZ AHMED

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

2016

PROCESS STATE SYNCHRONIZATION FOR MOBILITY SUPPORT IN MOBILE CLOUD COMPUTING

EJAZ AHMED

THESIS SUBMITTED IN FULFILMENT
OF THE REQUIREMENTS
FOR THE DEGREE OF PHD

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

2016

UNIVERSITI MALAYA

ORIGINAL LITERARY WORK DECLARATION

Name of Candidate: Ejaz Ahmed

Registration/Matrix No.: WHA120014

Name of Degree: Doctor of Philosophy

Title of Thesis: Process State Synchronization For Mobility Support in Mobile Cloud Computing

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

Mobile Cloud Computing (MCC) enables the resource-constrained mobile devices to execute the compute-intensive mobile applications either in client/server model or through cyber foraging of the tasks to the cloud servers. However, long or permanent network disconnections due to user mobility increase the execution time and in certain cases refrain the mobile devices from getting response back for the remotely performed execution. This research addresses the effect of mobility on application execution for mobile cloud-based interactive applications. We propose Process State Synchronization (PSS) as a mechanism to mitigate the impact of network disconnections on cloud-based interactive mobile applications. We investigate the impact of varying the synchronization interval size of PSS algorithms on the performance of application execution. The sufficient condition and upper bound for the synchronization interval size are also determined that minimize the execution time, computational wastage, synchronization overhead, and energy consumption. Results show that PSS reduces the execution time of applications up to 17% in case of intermittent disconnections. The comparison with existing offloading mechanisms shows that PSS reduces the execution time by up to 47% in case of intermittent network connectivity compared to COMET and by up to 35% in case of optimized VM-based offloading. The improvement in application execution time increases the usability of interactive mobile applications in MCC.

ABSTRAK

Mobile Cloud Computing (MCC) membolehkan sumber daripada peranti mudah alih yang terkekang melaksanakan pengiraan intensif daripada aplikasi-aplikasi mobil sama ada dalam model pelanggan/pelayan atau melalui kerja-kerja pencarian siber kepada pelayan-pelayan awan. Walau bagaimanapun, lama atau kekal pemutusan rangkaian yang disebabkan oleh mobiliti pengguna yang meningkatkan pelaksanaan masa dan dalam kes-kes tertentu menahan peranti mudah alih daripada mendapat kembali tindak balas yang dilakukan dari jarak jauh. Kajian ini dibuat untuk menangani kesan pergerakan ke atas tindak balas aplikasi masa dan pelaksanaan masa bagi aplikasi interaktif yang berasaskan awan mudah alih. Kami mencadangkan Process State Synchronization (PSS) atau Proses Penyelarasan Keadaan (PPK) sebagai mekanisme untuk meringankan impak daripada pemutusan rangkaian ke atas aplikasi interaktif mudah alih yang berasaskan awan. Kami menyiasat impak yang berbeza-beza daripada saiz selang masa penyelarasan algoritma PSS ke atas prestasi pelaksanaan aplikasi. Keadaan yang mencukupi dan batas atas yang terikat untuk saiz selang masa penyelarasan juga menentukan pengurangan masa pelaksanaan, hasil buangan pengkomputeran, penyelarasan penggunaan , dan penggunaan tenaga. Keputusan menunjukkan bahawa PSS mengurangkan masa pelaksanaan aplikasi hampir 17% dalam kes pemotongan sekala. Perbandingan dengan mekanisme pelepasan sedia ada menunjukkan bahawa PSS mengurangkan masa pelaksanaan sehingga 47% dalam kes sambungan rangkaian terputus-putus berbanding COMET sehingga 35% dalam kes pengptimuman perlepasan berasaskan VM. Penambahbaikan dalam perlaksanaan aplikasi masa dapat meningkatkan kebolehgunaan aplikasi mudah alih interaktif dalam MCC.

ACKNOWLEDGEMENTS

First of all, I am thankful to Allah Almighty for endowing me the strength, wisdom, and endless blessings to study.

I would like to sincerely express my deepest gratitude to my supervisor, Professor Dr. Abdullah Gani, and co-supervisor, Dr. Siti Hafizah ab Hamid, for their invaluable guidance, supervision, and encouragement to me throughout this research. Their continuous guidance and support assisted me conducting a valuable piece of research reported in this thesis. They provided me the opportunity to broaden my professional experience and prepare me for future challenges. Thereafter, I am deeply indebted and grateful to Dr. Anjum Naveed, Senior Lecturer, for his extensive technical guidance and personal involvement in this research. I am also thankful to Muhamad Afiq Zaini Alamar for proof reading of the Malay abstract.

I would also like to express my sincerest gratitude and special appreciation to my parents and siblings for their endless love and support during my PhD journey. Without their moral support, this dissertation would never have been completed. Words cannot express how grateful I am to my parents and siblings for all of the sacrifices that they have made on my behalf so I dedicate highest achievement of my student life to them.

Finally, I would like to thank University of Malaya for offering me a full research scholarship throughout my doctoral study. I would also like to thank Bright Sparks Unit and Faculty of Computer Science and Information Technology for the research grants and financial support.

TABLE OF CONTENTS

ORIGINAL LITERARY WORK DECLARATION ii

ABSTRACT iii

ABSTRAK iv

ACKNOWLEDGEMENTS v

TABLE OF CONTENTS vi

LIST OF FIGURES x

LIST OF TABLES xii

CHAPTER 1: INTRODUCTION 1

1.1 Domain Background 2
- 1.1.1 Cloud Computing 2
- 1.1.2 Mobile Cloud Computing 3
- 1.1.3 Cloud-based Mobile Application Execution 4
- 1.1.4 Impact of Network Connectivity 5

1.2 Research Motivation 6

1.3 Statement of The Problem 7

1.4 Statement of Objectives 9

1.5 Proposed Methodology 9

1.6 Layout of Thesis 12

CHAPTER 2: CLOUD-BASED MOBILE APPLICATION EXECUTION FRAMEWORKS 14

2.1 Cloud based Mobile Application Execution 14
- 2.1.1 Cloud-based Frameworks 15
- 2.1.2 Cloudlet-based Frameworks 20
- 2.1.3 Mobile Ad-hoc Cloud Frameworks 23
- 2.1.4 Hybrid Frameworks 24

2.2 Application Performance Enhancement by the Frameworks 29
- 2.2.1 Taxonomy of Application Performance Enhancement 29
  - 2.2.1.1 Cloud-centric Approaches 30
  - 2.2.1.2 Hybrid Approaches 34
  - 2.2.1.3 Mobile-centric Approaches 38
  - 2.2.1.4 Network-centric Approaches 42

2.3 Open Challenges: Application Performance Enhancement 44

2.4 Conclusion 48

CHAPTER 3: PROBLEM ANALYSIS OF CLOUD-BASED MOBILE APPLICATION EXECUTION IN DISRUPTIVE NETWORKS 51

3.1 Empirical Study: Network Disconnection and Application Execution 52
- 3.1.1 Experimental Setup and Data Collection 52
- 3.1.2 Mobility Impact on Device Connectivity and Response Time in 3G Network 54

3.1.3 Types of Network Connectivity Profiles 55
3.1.4 Execution Performance in Different Connection Profiles 56
3.1.4.1 Execution Time 56
3.1.4.2 Computation Wastage 58
3.1.5 State-of-the-art Frameworks Analysis 60
3.1.5.1 Execution Time 60
3.1.5.2 Computation Wastage 62
3.1.5.3 Energy Consumption 63
3.1.6 Impact of Varying the Disconnection Parameters 64
3.1.6.1 Execution Time 64
3.1.6.2 Computation Wastage 65
3.1.6.3 Cloud-based Application Execution Gain 65
3.1.7 Discussion 66
3.2 Formal Analysis: Network Disconnection and Application Execution 68
3.2.1 Formal Definitions 68
3.2.2 Connectivity Profiles and Execution Time 71
3.2.2.1 Same Cloud 72
3.2.2.2 Cloudlet-to-cloudlet 73
3.2.2.3 Cloud/Cloudlet-to-mobile 74
3.2.3 Disruptive Remote Execution Comparison with Non-disruptive Execution 75
3.2.3.1 Same Cloud 76
3.2.3.2 Cloudlet-to-Cloudlet 77
3.2.3.3 Cloud/Cloudlet-to-mobile 78
3.2.4 Formal Analysis of Application Execution Frameworks in Disruptive Network Conditions 80
3.2.4.1 Optimized VM based Cloudlet 80
3.2.4.2 COMET 81
3.3 Conclusion 83

CHAPTER 4: PROCESS STATE SYNCHRONIZATION ALGORITHM 85

4.1 Process State Synchronization (PSS) 85
4.1.1 Cloud-based Application Execution 86
4.1.2 Process State Capture and Process Resumption 87
4.1.3 State Synchronization 89
4.1.3.1 Cloud-side Module 90
4.1.3.2 Mobile-side Module 91
4.2 Example Illustration 93
4.3 Mathematical Model of PSS 97
4.3.1 Application Execution Time 98
4.3.2 Application Execution Time with PSS 99
4.3.3 Sufficient Condition for Usefulness of PSS 102
4.3.4 Upper-bound on Synchronization Interval 102
4.4 Distinguishing Features of Proposed Algorithm 104
4.4.1 Adaptive Synchronization Interval 105
4.4.2 Lightweight Synchronization Mechanism 105
4.4.3 Distributed Algorithm 105
4.4.4 Two-way Synchronization 106
4.5 Conclusion 106

CHAPTER 5: EVALUATION 108

5.1 Introduction 108
5.2 Performance Evaluation 109

5.2.1 Experimental Setup 109
5.2.2 Connection Execution Profiles 111
5.2.3 Prototype Application and Performance Metrics 112
5.2.4 Data Gathering and Data Processing 113
5.3 Data Collected For Model Validation 114
5.4 Data Collected for Different Disconnection-Execution Profiles 116
5.4.1 Execution Time 116
5.4.2 Computation Wastage 118
5.5 Data Collection for Analyzing the Impact of Synchronization Interval 118
5.5.1 Impact of Synchronization Interval on Execution Time 119
5.5.2 Impact of Synchronization Interval on Number of Resynchronizations 120
5.5.3 Impact of Synchronization Interval on Synchronization Overhead 121
5.5.4 Impact of Synchronization Interval on Valuable Computation Performed by Process State Synchronization Enabled Mobile Device 122
5.5.5 Impact of Synchronization Interval on Cloud Computation Wastage 123
5.5.6 Impact of Synchronization Interval on Mobile Device Computation Wastage 124
5.5.7 Impact of Synchronization Interval on Number of Instructions Executed Between Last Sync Point and Disconnection 125
5.6 Data Collected For Performance Comparison of PSS with Optimized VM-based Cloudlet and COMET 126
5.6.1 Execution Time 126
5.6.2 Cloud Computation Wasted 127
5.6.3 Energy Consumption 127
5.7 Conclusion 128

CHAPTER 6: RESULTS AND DISCUSSIONS 130

6.1 Model Validation 130
6.1.1 Execution Time 130
6.1.2 Mobile Device Useful and Useless Computation Difference 131
6.2 Comparison of Mobile Application Execution in Different Connection Profiles 132
6.2.1 Execution Time 132
6.2.2 Computation Wastage 134
6.3 Performance Analysis of Process State Synchronization Algorithm 135
6.3.1 Impact of Synchronization Interval 136
6.3.1.1 Execution Time 136
6.3.1.2 Computation Wastage 136
6.3.1.3 Valuable Computation Performed on Mobile Device 140
6.3.1.4 Number of Resynchronizations 141
6.3.1.5 Instructions Executed Between Last Sync Point and Disconnection 142
6.3.1.6 Synchronization Overhead 143
6.3.1.7 Energy Consumption 143
6.3.2 Synchronization vs. Non synchronization-based Execution 144
6.3.3 Valuable Computation on Mobile Device 145
6.4 Comparison of PSS-based Execution with COMET and Optimized VM-based Execution 147
6.4.1 Execution Time 147
6.4.2 Computation Wasted 149
6.4.3 Energy Consumption 150
6.5 Conclusion 151

CHAPTER 7: CONCLUSIONS **153**
7.1 Reappraisal of the Research Objectives 153
7.2 Contribution of the Research 155
7.3 Research Scope and Limitations 159
7.4 Future Work 159

REFERENCES **161**

LIST OF FIGURES

Figure 1.1	Proposed Research Methodology	10
Figure 2.1	Cloud-based MCC Environment	15
Figure 2.2	Cloudlet-based MCC Environment	21
Figure 2.3	Mobile Ad-hoc cloud	23
Figure 2.4	Hybrid MCC Environment	25
Figure 2.5	Taxonomy of Application Performance Enhancement Approaches Employed by the Frameworks for the MCC	30
Figure 3.1	Data Collection Intra-city Metro Train Path Map (Courtesy: MYrapid http://www.myrapid.com.my)	53
Figure 3.2	Mobility Effect on 3G Network Connectivity and Application Response Time	55
Figure 3.3	Execution Time in Different Connection Profiles	58
Figure 3.4	Computation Wasted For Different Connection Profiles	59
Figure 3.5	Execution Time in Optimized VM-based Cloudlet, COMET, and Cloud Server Execution	61
Figure 3.6	Computation Wastage in Optimized VM-based Cloudlet and COMET	62
Figure 3.7	Energy Consumption in Optimized VM-based Cloudlet, COMET, and Mobile Device Execution	64
Figure 3.8	Impact of Disconnection Start Time on Cloud-based Application Execution Time	65
Figure 3.9	Impact of Disconnection Start Time on Computation Wasted	66
Figure 3.10	Impact of Expected Disconnection Interval Size on Cloud-based Application Execution Gain	67
Figure 4.1	Example Process State, highlighting the reference update required while resuming the process	88
Figure 4.2	Non-Synchronized Application Execution Sequence Diagram for MCC	94
Figure 4.3	Synchronized Application Resumption Sequence Diagram for MCC	95
Figure 4.4	Re-Synchronization Decision on Reconnect Sequence Diagram for MCC	97
Figure 4.5	Illustration of useful and useless computation disconnection intervals	100
Figure 4.6	Illustration of DC interval for useful and wasteful computations	101
Figure 4.7	Synchronization Interval For Balancing Trade-off between Mobile Device Useful and Useless Computation	103
Figure 5.1	Experimental Setup Illustration	110
Figure 6.1	Comparison of Execution Time Empirical Results with Mathematical Model Execution Time	131
Figure 6.2	Useful and Useless Mobile Computation Absolute Difference for Different Data Traces	133
Figure 6.3	Execution Time in Different Connection Profiles	134
Figure 6.4	Comparison of Computation Wastage With Different Connection Profiles	135
Figure 6.5	Impact of Synchronization Interval on Execution Time	137

Figure 6.6 Impact of Synchronization Interval on Mobile Device Computation Wastage 138
Figure 6.7 Impact of Synchronization Interval on Cloud Computation Wastage 139
Figure 6.8 Impact of Synchronization Interval on Valuable Computation Performed on Mobile Device 140
Figure 6.9 Impact of Synchronization Interval on Number of Resynchronizations 141
Figure 6.10 Impact of Synchronization Interval on Instructions Executed Between Last Sync Point and Disconnection 142
Figure 6.11 Impact of Synchronization Interval on Synchronization Overhead 143
Figure 6.12 Impact of Synchronization on Energy Consumption 144
Figure 6.13 Execution Time in Synchronization and Non-Synchronization 145
Figure 6.14 Valuable Instructions Percentage Computed on Mobile Device 146
Figure 6.15 Execution Time Comparison With State-of-the-art Application Execution Frameworks 148
Figure 6.16 Computation Wastage Comparison With State-of-the-art Application Execution Frameworks 149
Figure 6.17 Energy Consumption Comparison With State-of-the-art Application Execution Frameworks 150

LIST OF TABLES

Table 1.1	Thesis Layout	13
Table 2.1	Comparison of frameworks based on application execution performance enhancement cloud-centric approaches	35
Table 2.2	Comparison of frameworks based on application execution performance enhancement hybrid approaches	38
Table 2.3	Comparison of the frameworks based on the application execution performance enhancement mobile-centric approaches	39
Table 2.4	Comparison of frameworks based on the application performance enhancement network-centric approaches	43
Table 3.1	Execution Time Comparison in Different Connection Profiles	57
Table 3.2	Computation Wastage Comparison in Different Connection Profiles	59
Table 3.3	Execution Time in Optimized VM-based Cloudlet, COMET, and Cloud Server Execution	60
Table 3.4	Computation Wastage in Optimized VM-based Cloudlet and COMET	62
Table 3.5	Energy Consumption in Optimized VM-based Cloudlet, COMET, and Mobile Device Execution	63
Table 3.6	Impact of Disconnection Time on Cloud-based Application Execution Gain	66
Table 3.7	HOL Symbols used	69
Table 3.8	System variables	71
Table 4.1	Symbols and Their Descriptions	90
Table 4.2	Symbols and descriptions	98
Table 5.1	Systems specifications	111
Table 5.2	Mathematical model parameters and their corresponding values	114
Table 5.3	Application Execution Time Computed Through Mathematical Model and Experiments	115
Table 5.4	Useful Instructions Computed Through Mathematical Model and Experiments	115
Table 5.5	Wasted Instructions Computed Through Mathematical Model and Experiments	115
Table 5.6	Application Execution Time for Different Disconnection-Execution Profiles	117
Table 5.7	Computation Wastage for Different Disconnection-Execution Profiles	118
Table 5.8	Impact of Synchronization Interval on Execution Time	119
Table 5.9	Impact of Synchronization Interval on Number of Resynchronizations	120
Table 5.10	Impact of Synchronization Interval on Synchronization Overhead	121
Table 5.11	Impact of Synchronization Interval on Valuable Computation Performed by Process State Synchronization Enabled Mobile Device	122
Table 5.12	Impact of Synchronization Interval on Cloud Computation Wastage	123
Table 5.13	Impact of Synchronization Interval on Mobile Device Computation Wastage	124

Table 5.14 Impact of Synchronization Interval on Executed Instructions Between Last Sync Point and Disconnection 125
Table 5.15 Execution Time in Optimized VM-based Cloudlet, COMET and PSS-based Execution 126
Table 5.16 Comparison of Computation Wastage in Optimized VM-based Cloudlet, COMET and PSS-based Execution 127
Table 5.17 Comparison of Energy Consumption in Optimized VM-based Cloudlet, COMET and Process State Synchronization Algorithm 128

CHAPTER 1 : INTRODUCTION

The advancements in wireless technologies have shifted the computing paradigm from static computing to mobile computing where a mobile user could perform the computing tasks while user is on the move. Small sized easy to carry mobile computational units are being developed that could communicate over wireless communication medium. Mobile computing paradigm resulted in rapid growth of resource hungry mobile applications. The demand for wireless bandwidth soon became many folds, compared to the bandwidth available through most advanced wireless technologies. In addition, small size requirements of mobile devices resulted in limited computational resources and limited battery power. Researchers have overcome the limitations of mobile devices through mobile cloud computing (MCC).

MCC connects the resource limited mobile devices with the resource rich clouds using wireless communication. A significant part of the mobile application is executed in the cloud while mobile device is used only for user interaction and display of results. At the same time, the communication between mobile device and the cloud is minimized in an effort to reduce the use of resource limited communication medium. This thesis addresses the research problem of cloud based mobile application state loss in the domain of MCC. Therefore, we start the thesis as well as this chapter with the brief introduction to MCC.

This chapter is organized into six sections. In Section 1.1, we discuss the domain background of MCC, wireless communication limitations and their impact on cloud based mobile application execution. Section 1.2 presents the motivation for research study in the domain of MCC. Section 1.3 highlights the research gap, briefly explains the problem of state loss of cloud-based mobile applications and summarizes the statement of the problem. Section 1.4 enlists the research objectives of the research study carried out in this thesis. Section 1.5 summarizes the methodology followed in the research and

Section 1.6 sketches the layout for the rest of the thesis.

1.1 Domain Background

This section starts with the brief discussion on cloud computing that leads to the mobile cloud computing. Subsequently, we discuss the dependence of MCC on wireless communications and the limitations of wireless communications that affect the application execution in MCC. Finally, we discuss the cloud based mobile application execution and briefly introduce the problem of state loss in cloud based mobile application execution due to intermittent wireless connectivity.

1.1.1 Cloud Computing

Cloud computing is an on-demand computing paradigm, where computers and other devices can access and use the shared resources and data on user request. Cloud computing allows users to store and process their applications' data on the third-party data centers and enables enterprises to minimize the upfront infrastructure cost by leveraging the resources of cloud. The enterprises can focus on projects that improve their business growth. The cloud computing allows the organizations to develop their applications and run faster with less maintenance cost.

Cloud computing is directed to deliver variety of services to Internet users. The main services of cloud are infrastructure-as-a-service (IaaS) (Nguyen, Cheriet, & Lemay, 2013; Ghosh, Longo, Xia, Naik, & Trivedi, 2014; Xia et al., 2015), platform-as-a-service (PaaS) (Androcec, Vrcek, & Kungas, 2015; Ardagna et al., 2012; Boniface et al., 2010), and software-as-a-service (SaaS) (Ma & Kauffman, 2014; Wu, Garg, Versteeg, & Buyya, 2014; Ojala, 2013). These services are designed to meet the requirements of wider range of the user demands across the Internet. Similarly, cloud computing promises to deliver reliable, scalable, sustainable, and secure services to the cloud users. The cloud applications can leverage on parallel and distributed computing provided by cloud servers to run in an isolated virtual environment. Cloud users access the cloud services and resources through Internet technology, they use the services provided by the cloud and pay based

on their usage. Hence, such services are considered as a kind of utility services similar to traditional utility services e.g. water, electricity, and gas.

A number of cloud platforms are available. For example, Elastic Compute Cloud (EC2) (Amazon, 2014a) is one of the cloud platforms that provides distributed IaaS for users to run their different operating systems (OS), similarly Amazon S3 (Amazon, 2014b) provides storage in a highly secure, fast and scalable manner. Google App Engine (GAE) (Google, 2012) is an example of a PaaS that facilitates the users for hosting platform and enables them to deploy and run their specific web applications. DropBox (*DropBox*, 2015) and Facebook (*Facebook*, 2015) are web-based cloud applications that are based on “web 2.0" to provide SaaS. In summary, cloud computing is an emerging technology that significantly decreases ownership cost by removing necessities for maintenance of locally deployed organization owned servers. In addition, the cloud computing provides heterogeneous environments featuring varied hardware, business policies, and architectures.

1.1.2 Mobile Cloud Computing

The mobile devices can also be augmented using cloud platforms in a fashion similar to the static thin clients. This has resulted in a new computation paradigm of MCC. Although MCC is already in wide use, in the future this paradigm will be employed in several areas such as education (Wong, Chai, Zhang, & King, 2015), health-care (e.g. telesurgery and telemonitoring) (Benharref & Serhani, 2014; Bourouis, Zerdazi, Feham, & Bouchachia, 2013; X. Wang, Gui, Liu, Jin, & Chen, 2014), and social networking (X. Wang, Chen, Kwon, Yang, & Leung, 2013; Lingjun, Jingdong, Bowen, & Jianzhong, 2014; Y. Wang, Wu, & Yang, 2013). As technological advancements in manufacturing the mobile resources is relatively slower than the ever growing application requirements and expectations of mobile users, the resource augmentation of mobile devices is the preliminary requirement for delivering computing capabilities near to user expectations (Abolfazli, Sanaei, Ahmed, Gani, & Buyya, 2014). MCC provides a software-based

solution for resource constrained mobile devices to leverage the resources of the cloud servers.

Although MCC is extensively being used and is similar to the cloud computing in principle, there are additional challenges involved in MCC. A significant difference between thin clients and the mobile devices is power availability. While thin clients are constantly connected to the power supply, mobile devices have limited battery power. This necessitates the limited use of computational as well as expensive wireless communicational resources in case of mobile devices. Furthermore, the available bandwidth using wireless technologies is far less than the wired counterparts, adding further restrictions on the possible communications between the mobile device and the cloud.

1.1.3 Cloud-based Mobile Application Execution

Mobile devices can execute their applications on the clouds using one of the two application execution models. The application can be programmed as a distributed application with majority of the application executing in the cloud. In this case, the application is pre-installed on the cloud as well as the mobile device. The mobile devices accesses the specific cloud and the two ends collaboratively perform the user task where majority of execution is done by the cloud. This is the more widely used model. The model has two limitations. First, the application must be re-programmed for cloud based model in order for it to be executed as cloud based mobile application. Second, the mobile device can access the application only when it is connected with the specific cloud server. In case the connection to the specific cloud is lost, the application becomes inaccessible.

The second model has extensively been discussed in research whereby the application, its components or the virtual machine encapsulating the application are migrated to the cloud server at run time and the execution is followed. This model has significant advantages, however, it requires higher bandwidth for transfer of mobile application from device to the cloud. Irrespective of the execution model, the successful execution of cloud based mobile applications is highly dependent upon high quality wireless communication

between mobile devices and the clouds.

1.1.4 Impact of Network Connectivity

Mobile devices access the clouds either through WiFi or 3G/4G cellular technologies (Ahmed, Akhunzada, et al., 2015). WiFi technology provides higher bandwidth compared to 3G/4G cellular networks, however, the coverage range of WiFi hotspots is significantly lesser in comparison to cellular networks. WiFi as well as cellular technologies do not provide uniform bandwidth throughout the coverage area. In addition, both technologies provide shared medium access. Therefore, the number of users accessing the Internet using wireless resources significantly affects the available bandwidth and further complicates the non-uniform access problem, resulting in dynamically varying bandwidth. Within the coverage area, the bandwidth decreases to a level where no effective communication can take place. Consequently, the network appears disconnected for the applications. Similarly, there can be actual points of no coverage for the mobile device users where no signals are received from any wireless transmitter in the close vicinity.

The performance of cloud-based mobile applications, particularly the execution of interactive applications is severely affected by the limited and non-uniform bandwidth and frequent network disconnections. Network disconnections result in loss of interaction with the cloud based application for the period of disconnection. In the worst case, the application can be terminated on the cloud server, its soft state can be cleared or the mobile device may not be able to connect back to the same cloud service or the cloudlet. Many a times, such disconnections mean the entire computation done at the cloud server is wasted and the user needs to re-execute the application either locally or on alternate cloud/cloudlet service.

In addition to intrinsic limitations of wireless technologies, the user mobility is one of the key factors that contributes to the performance degradation of MCC service. The user mobility causes the network disconnections when user moves into crowded areas, areas of poor coverage or the areas of no coverage, all of which may lead to application

disruption. The application disruption can result in loss of the application state on the cloud, which in turn leads to application discontinuation, re-migration of the application to another cloud, or re-execution of the application on the mobile devices from the initial stage. This induces the additional delay during the execution of the application.

1.2 Research Motivation

Cisco research (Cisco, 2015) reports that global mobile devices and connections have increased from 6.9 billion in 2013 to 7.4 billion in 2014. The research also shows global mobile data traffic grew from 1.5 exabytes per month at the end of 2013 to 2.5 exabytes per month at the end of 2014. Such explosive growth of mobile devices, mobile connections, and global mobile data traffic is an evidence of the increasing use of mobile devices. The report further presents the prediction of 11-fold growth in mobile cloud traffic from 2014 (2 exabytes/month) to 2019 (21.8 exabytes/month). The cloud application traffic was 81 percent of the total mobile data traffic in 2014 that is expected to reach the 90 percent of total mobile data traffic. Such tremendous increase in mobile cloud traffic is an evidence of the increasing use of distributed cloud based mobile applications.

Recent studies also show that commute time and the use of mobile applications during commute in metropolitan cities has significantly increased with the growth in population. U.S.A. census bureau data shows that 10.8 million people in U.S.A. commute for an hour or more to their work each way on a daily basis (Forbes, 2015). Another study reports that 84% of smart phone users use their mobile devices during commute hours (Chitika, 2015). The applications used can be entertainment based in the form of online games, video streaming and social interaction. The applications can also be productivity based, extending the complete office setup for the mobile users while they commute. Consider the example of Microsoft eBus introduced in 2007 where every seat is equipped with a thin client connected to the Microsoft network. The employees can connect to the network and start working while they commute to the office, making their commute time as official work hours. We can safely conclude that a significant number of cloud based

mobile applications and resultant traffic is generated by the users while they are on the move.

In this context, MCC enables the execution of the compute-intensive mobile applications by leveraging the resources of the cloud while the user is on move. The traditional application execution frameworks mainly focus on offloading process and give a relatively less consideration for handling the issue of network disconnection during the application execution. Hence, there is a need for a solution to address the limitations arising from network disruptions. Such a solution can significantly increase the possibility and range of applications that can be made available for the users while the users are on the move. The proposed research will be a big step in this direction and can become part of the future mobile cloud computing technologies.

1.3 Statement of The Problem

During the recent years, a number of researchers have proposed the cloud-based mobile application execution frameworks. These frameworks focus on partitioning the legacy applications to separate the compute-intensive parts for cloud execution, wrapping the application in virtual machines to make them platform independent, application transfer, and result collection from the cloud servers.

Majority of the cloud based mobile application execution frameworks such as mobile application execution framework (Hung, Shih, Shieh, Lee, & Huang, 2012) and MAUI (Cuervo et al., 2010) only focus on application migration decision without emphasizing on the mobility issue during the execution of the application on the cloud server. However, some of the state-of-the-art application execution frameworks, such as COSMOS (Shi et al., 2014) and dynamic offloading algorithm (Y. Zhang, Niyato, Wang, & Tham, 2014), address the issue of the network disconnection by incorporating the risk of network failure and user mobility speed in offloading decision making. These frameworks re-execute the application either on a new cloudlet or on mobile device locally after network disconnection. Hence, the execution based on these frameworks faces additional

execution time and computation wastage on the cloud server.

The frameworks such as optimized VM-based cloudlet (Ha, Pillai, Richter, Abe, & Satyanarayanan, 2013) and COMET (Gordon, Jamshidi, Mahlke, Mao, & Chen, 2012) exchange the executed portion of the application with the mobile device during the execution on the cloud server. The frameworks which exchange the intermediate execution results with the mobile device face high overhead because of the exchange of VM-level information. In case of network disconnection, the task can be resumed on the mobile device from the last point of synchronization; however, upon reconnection, the execution states can not be sent back to the cloud server for further execution. Similarly, the optimized VM-based cloudlet requires 12 seconds for VM-overlay transfer, which may be long enough not to allow complete transfer before disconnection. Hence, it is concluded that existing solutions do not fit for the uninterrupted execution of the mobile application on the cloud server.

Based on this discussion, we can say that the problem of state loss of the cloud based mobile application execution upon mobile device disconnection from cloud network has not been addressed. The highlighted research gap leads us to the research statement for this thesis.

MCC enables the users to leverage the cloud resources for enhancing the computational capabilities of mobile devices while on the move. However, the network connection becomes unstable while user is on the move that causes application execution disruption. The disruption leads loss of application state, execution discontinuation, re-execution of the whole task from initial stage on mobile device, or remigrating the application to another device. Such execution disruptions increase the application execution time and cause the computation wastage that degrade the user experience and decrease the usability of cloud-based mobile applications.

1.4 Statement of Objectives

We address the state loss problem of the cloud based mobile application execution at an event of network disconnection. We define following objectives that are to be achieved in order to attain the aim of this research.

1. Review the application execution frameworks in MCC for acquiring the insight on the state-of-the-art with reference to network disconnection issue during the execution of cloud-based mobile application.

2. Investigate the impact of user mobility on cloud based mobile application execution in the cloud.

3. Design and develop the solution for saving the execution state in the form of process state synchronization algorithm for mobility support to minimize the execution time, computational wastage, synchronization overhead, and energy consumption.

4. Develop mathematical model for the proposed solution, validate the model using empirical analysis and compare the performance of proposed solution with the state-of-the-art cloud based mobile application execution frameworks.

1.5 Proposed Methodology

We divide the whole research into four main phases as shown in Figure 1.1 to achieve the set of objectives defined in Section 1.4. Each research phase is targeted to achieve an objective.

We review the state-of-the-art credible application execution frameworks in MCC to investigate the support provided by the frameworks for handling the network disconnection issue. The traditional computational offloading frameworks are classified into three main categories: a) cloud-based frameworks, b) cloudlet-based frameworks, and c) hybrid frameworks. The investigation also reveals that these frameworks can be classified into three classes with respect to addressing the network disconnection issue during the application execution on the cloud server. One class of frameworks incorporate the risk

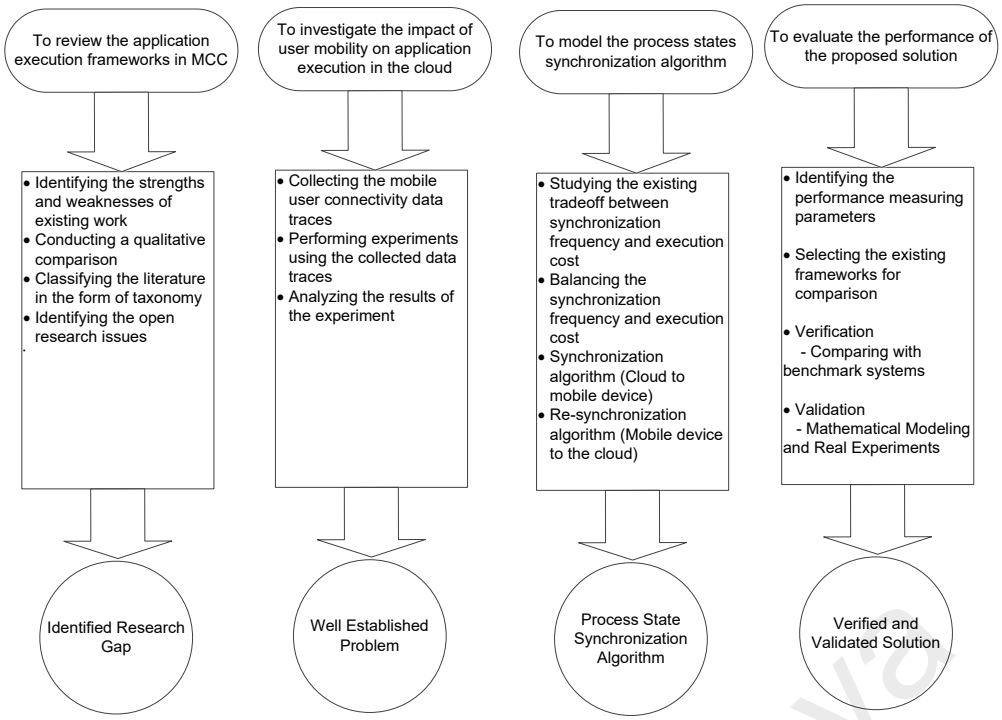


Figure 1.1: Proposed Research Methodology

of network failure and user mobility speed in offloading decision making and re-execute the application on new cloudlet or on mobile device locally after network disconnection. Another class of frameworks exchanges the executed portion of the application with the mobile device before the mobile device disconnects from the cloud. Third class of application execution frameworks only focuses on application migration decision without emphasizing on the mobility issues. We have also identified the issues in the application execution frameworks that affect the overall performance of the application execution on the cloud.

Second phase of research involves investigating the research problem by conducting experiments in an emulated environment of MCC in lab. The mobile device connectivity data traces are collected and used in the experiments. The impact of user mobility on user connectivity, application response time, application execution time, execution time gain ratio, and computation wastage are investigated.

In next phase of the research, we propose a distributed process states synchronization (PSS) algorithm that enables the application execution frameworks to exchange the intermediate results between the mobile device and the cloud server. The aim of the pro-

posed solution is to minimize the execution time and synchronization overheads of an application in case of network disconnections due to user mobility. The execution time is minimized by exchanging the process states from the cloud server to the mobile device, where the execution is resumed during the network disconnection. The process states of the executed application are resynchronized back with the cloud server on the connection reestablishment.

We implement and evaluate the proposed algorithm in emulated MCC environment. A simple interactive prototype application is designed and developed for the mobile device, which is tested with different user connectivity profiles in the emulated MCC environment. A mathematical model is proposed and validated against empirical results. Matlab implementation of the model has been used to generate results and 50% results have been empirically verified. Experimental results are validated by benchmarking prototype application under different conditions of mobile user connectivity. The data are collected by testing PSS for 30 different mobile user connectivity data traces in intra-city metro train mobility scenarios. The confidence interval for the experiment is computed with 99% degree for the sample space of 30 values. The algorithm performance is verified by comparing experimental results of PSS with that of the state-of-the-art application execution frameworks including optimized VM-based cloudlet and COMET.

1.6 Layout of Thesis

Research thesis is a detailed research study of the problem "Process State Synchronization for Mobility support in Mobile Cloud Computing" therefore the thesis has been organized into chapters for clear understanding of the matter. Table 1.1 presents the organization of thesis. This thesis is composed of seven chapters that is organized as follows:

Chapter 2 presents a review of the state-of-the-art application execution frameworks proposed for MCC and investigates the critical aspects of the frameworks with respect to network disconnections 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

Table 1.1: Thesis Layout

What?	Why?	How?
Introduction	(a) Highlighting the reason for the research (b) Stating the problem and presenting the objectives (c) Discussing the thesis organization	(a) Stating the rational for undertaking the research, (b) Formally writing the statement of problem and statement of objectives
Literature Review	(a) Classifying and investigating the strengths and weaknesses of the state-of-the-art literature (b) Identifying the open issues	(a) Critical analysis of the existing solutions (b) Devisal of the taxonomy and comparison based on the taxonomy
Problem Analysis	(a) Comprehensive understanding of the mobility impact on application execution (b) Understanding the impact of the problem	(a) Empirical study using real mobility scenarios from metro transit MCC environment
Process State Synchronization Algorithm	(a) Giving the clear understanding of the proposed solution to the reader	(a) Presenting the pseudo code of the algorithm and giving an illustrative example
Data Collection	(a) Discussing the data generation process and to report the collected data	(a) Discussion on data collection method (b) Explaining the tools used for evaluating the proposed solution (c) Reporting the collected data
Results and Discussion	(a) Highlighting the effectiveness of the proposed solution by analyzing the experimental results	(a) Sharing the insights gained from the experimental results (b) Comparing the performance of PSS-based execution with optimized VM-based cloudlet and COMET
Conclusion	(a) Summarizing the findings of the research work and highlighting the significance of the proposed solution (c) Discussing the limitations of the research work and proposing future directions of the research	(a) Reporting the re-examination of the research objectives

from the thematic taxonomy. The issues to application execution frameworks are also identified and discussed in the chapter.

Chapter 3 analyzes the mobility impact on application response time in wireless network. The mobile user connectivity in scenarios of intra-city metro train mobility is also studied. The impact of user mobility on application execution in MCC environment is also analyzed by executing the application in intra-city metro train mobility scenarios. The analysis of the problem is carried out with different connection profiles and for two state-of-the-art application execution frameworks. The analysis shows that the state-of-the-art frameworks lack of features to handle the issue of network disconnection during the execution of the application on cloud server.

Chapter 4 presents a process state synchronization mechanism that aims to solve the issue of network disconnections during the execution of 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 process state synchronization in terms of synchronization overhead, synchronization frequency, and energy consumption. 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 significance of the proposed solution, discusses the limitations of the research work and proposes future directions of the research.

CHAPTER 2 : CLOUD-BASED MOBILE APPLICATION EXECUTION FRAMEWORKS

We started the thesis with basic introduction of mobile cloud computing. We also introduced the problem of state loss in cloud based mobile applications during execution on the cloud. The purpose of this chapter is to present the review of the literature related to the problem. A number of application execution frameworks have been proposed in the literature. These frameworks manage different aspects of cloud based mobile application execution. We start by reviewing the application execution frameworks and highlight their management of network resources and disconnections. Subsequently, we present taxonomy of the frameworks with reference to application performance, specifically application execution time, which is directly affected by network disconnections. Finally, we highlight open challenges along with problem addressed in this thesis and conclude the chapter.

The chapter is organized into four sections. In Section 2.1, we present the state-of-the-art application execution frameworks and their management of network resources and disconnections. Section 2.2 presents taxonomy of the various approaches employed by the application execution frameworks to improve the application execution time in the cloud, which is a factor affected by network disconnections. Section 2.3 highlights the open challenges in optimizing the application execution time in MCC environment and Section 2.4 summarizes the chapter with conclusive remarks.

2.1 Cloud based Mobile Application Execution

In this section, we present a comprehensive survey on the state-of-the-art mobile application execution frameworks for MCC. The application execution frameworks are analyzed with respect to various features, particularly mobility support and execution failure handling during the network disconnections. We have published the conducted survey on the topic in (Ahmed, Gani, Sookhak, Ab Hamid, & Xia, 2015; Ahmed, Gani,

Khan, Buyya, & Khan, 2015). The application execution frameworks can be categorized into four types: cloud-based frameworks, cloudlet-based frameworks, mobile ad hoc cloud-based frameworks, and hybrid frameworks.

2.1.1 Cloud-based Frameworks

The mobile device that uses the remote cloud to augment resources, acts as a thin client while connecting through any of the wireless technologies to a remote cloud server (X. Zhang, Jeong, Kunjithapatham, & Gibbs, 2010), as shown in Figure 2.1.

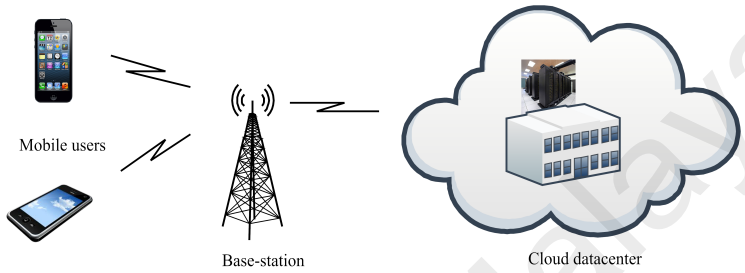


Figure 2.1: Cloud-based MCC Environment

The cloud-based frameworks enjoy a diverse range of available services in the cloud, higher computational power, lower computation latency, and the on-demand availability of resources (Abolfazli et al., 2014; Whaiduzzaman, Haque, Rejaul Karim Chowdhury, & Gani, 2014). However, the communication between the mobile device and the cloud goes through the Internet, which experiences high latency, non-guaranteed bandwidth, bursty losses and non-deterministic traffic load along the path. The Internet link can be unavailable for multiple reasons such as user mobility (Mehendale, Paranjpe, & Vempala, 2011). These factors increase the probability of network connectivity disruption during the access of services provided by the remote cloud. Therefore, cloud based frameworks need to consider the network connectivity as a major design challenge. In the following, we discuss the frameworks, while keeping network connectivity in mind.

Zhang *et al.* have proposed an adaptive platform independent elastic application execution framework (X. Zhang et al., 2010) to enable the use of remote cloud resources in a seamless manner. The framework provides optimal elasticity by incorporating mul-

tiple factors, such as the availability of mobile devices and the cloud resources, users' preferences, and the application performance measures. The proposed framework partitions the application into multiple components called weblets. The distinctive attribute of the framework is that the elastic partitioned components are replicated across multiple cloud servers that enhances availability and reliability. To attain this objective, multiple elasticity patterns, such as splitter, aggregator, and replication, are defined. In the replication pattern, multiple servers in the cloud execute multiple replicas of a single interface. Therefore, the replica failure does not compromise the operation of the system. However, the framework needs developer support to find the organization of the weblets by considering the data dependencies, resource requirements, and functionalities. Although the framework minimizes the execution cost due to network failure, the replication of weblets increases the overall maintenance cost. This is also not an effective solution to replicate one application on multiple servers for the execution. Similarly, last hop link failure leads to failure of the complete service.

Yang *et al.* have proposed an adaptive framework for optimal dynamic partitioning and execution of data stream applications (Yang et al., 2012) that aims to maximize the speed and throughput. The framework consists of modules running on both sides (mobile device and the cloud server). Every mobile application in the framework has an application master on the cloud side, which is responsible for adaptive partitioning and distributed execution. When an application runs on the mobile device, a request is directed to the resource manager component in the cloud. The resource manager ascribes an application master to deal with the request. Thereafter, the application master requests the mobile device for the characteristics of the device, collected using profiler, that are incorporated with static application characteristics stored in the cloud. Optimization solver, which is implemented on cloud side, is used to dynamically compute application partitioning. Although the framework supports multi-tenancy and provides scalable, adaptive, and dynamic partitioning, the framework does not handle the situation that arises after the network disconnection.

S. Kosta *et al.* proposed ThinkAir (Kosta, Aucinas, Hui, Mortier, & Zhang, 2012), a dynamic and adaptive framework that exploits multiple mobile device virtualizations to simultaneously execute multiple offloaded methods to reduce the application execution time. Different sized VMs are allocated to the user applications, depending on the application demand. ThinkAir framework comprises of three main components: the profilers, the application server, and the execution environment. The profiler provides information for offloading decisions. The application server manages the cloud side of the offloaded application components and is responsible for handling the client requests, allocating resources, and facilitating with parallel processing. The execution environment consists of a simple programming library, compiler, and execution controller. The library makes the programmer's job easier. the compiler has code generators to translate the annotated code. The execution controller determines whether to offload the execution of a particular method or let it continue locally on the mobile device. ThinkAir offloading decision is complex due to the incorporation of multiple parameters in the decision, increases the decision time. The framework re-executes the application on the mobile device when the network failure occurs.

CloneCloud (Chun, Ihm, Maniatis, Naik, & Patti, 2011) supports a flexible application partitioner that empowers the unmodified mobile applications to seamlessly offload the compute-intensive partitions to the trusted remote cloud. The system supports dynamic profiling and static analysis to partition the mobile application. The main goal of the partitioning is to optimize the energy usage and the execution time. Unlike its counterparts, the CloneCloud partitions the application at thread level. Thread state migration at pre-determined check points along with device replica on the cloud is used for migration. The overhead of CloneCloud comes from mobile device synchronization with the cloud, which must be done securely to ensure privacy. Moreover, the framework migrates a single thread at one time that restrains the gain of concurrent execution of components to minimize the execution time at the remote cloud server. Similarly, network disconnections are not handled other than restarting the threads locally.

Hung *et al.* proposed a cloud-based mobile application execution framework (Hung et al., 2012). The framework involves installation of proposed agent program, allocation of delegate system, creation of a virtual environment, cloning of operating environment, migration of applications, and synchronization of user data and application. To avoid the loss of input data, application replay technique is integrated with a state-saving scheme. The framework does not demand the application redesign; however, the application level state saving approach and the data categorizations are added to prioritize the data synchronization. The proposed framework only requires the transfer of the application saved states instead of the states of the entire VM that results into low overhead. Although the framework reduces the amount of data transfer, it still faces the delay induced by the run-time agent program installation, VM creation, and states migration that hinders the realization of smooth application execution for the MCC. Moreover, the mobile device re-executes the application locally when the network disconnection occurs.

Lee *et al.* proposed application replication-based execution framework for the application execution (Lee, 2012). The offloading decision is performed on the cloud server. The framework comprises of a toolkit that facilitates the deployment of run-time infrastructure and development of the applications. For the first execution, the application is transported from the mobile device, but for the rest of the requests, the application is replicated from the previous WorkerNode. The framework exchanges a lot of control and context information with the cloud data center to execute the complex offloading decision algorithm. Therefore, the solution is not scalable for highly dynamic workload environments and network conditions. The data consistency among different nodes is a big challenge. No support for network disconnections is provided in the framework

Verbelen *et al.* introduced AIOLOS (Verbelen, Simoens, De Turck, & Dhoedt, 2012a), an adaptive offloading decision engine with support for dynamic resource and network condition consideration. However, the adaptive offloading algorithm is compute- and energy-intensive. Although the network state is monitored, it is only considered during offloading decision. Giurgiu *et al.* proposed a middleware framework (Giurgiu, Riva,

Juric, Krivulev, & Alonso, 2009) that automatically and dynamically partitions and offloads various parts of an application to the cloud. The optimization objective function of the framework minimizes the interaction latency between the mobile device and the cloud server, while taking care of the exchanged data overhead. The framework uses two different algorithms, namely: ALL and K-step where the problem is formulated as static and dynamic optimization problem, respectively. Similar to other frameworks, both frameworks have no support for network disconnections and re-executes the application locally on the mobile device when the network disconnection occurs.

Coalesced offloading framework (Xiang, Ye, Feng, Li, & Li, 2014) was proposed by Xiang *et al.*, that saves energy by leveraging the property of multiple applications to bundle their code offload requests. The problem of coalesced offloading is formulated as a joint optimization problem with the objective to minimize the response time and energy cost. Two online algorithms, called as Ready, Set, Go (RSG), are proposed to solve the formulated optimization problem. Without any prior knowledge of upcoming code offloading requests, the online algorithms outperforms the optimal offline algorithm. Although the middleware framework reduces the interaction latency, the consumption graph modelling, optimal cut finding algorithm, two-step dynamic partitioning analysis, and intensive profiling consume computational resources of the mobile device. Therefore, the compute-intensive nature of the framework increases the overall application execution time. The framework executes the application on a local mobile device when the network disconnection occurs.

It is obvious to observe that none of the discussed frameworks have adequate support for the network disconnection. The primary focus of cloud based frameworks is application partitioning, reduction in interaction between cloud and the mobile device to manage latency, managing changing network conditions and energy efficiency of the mobile device. Apart from these efforts, some of the research works focus on efficient designing of data centers as reported in (Shuja et al., 2014, 2012). Clearly, without proper support for network disconnections, significant computation can be wasted and leads to high

Figure 2.2: Cloudlet-based MCC Environment

overhead. Furthermore, given the present state of wireless networks and keeping user mobility in view, handling network disconnections is a non-trivial task.

2.1.2 Cloudlet-based Frameworks

Cloudlets are small scale clouds that are generally accessible over a single hop either through WiFi or through cellular technology (Zhao, Xu, Chi, Zhu, & Cao, 2012). Consequently, the problem of managing the network latency and variable network conditions reduces significantly (Satyanarayanan, Bahl, Caceres, & Davies, 2009). In (Shaukat, Ahmed, Anwar, & Xia, 2016), we conducted a comprehensive survey on the cloudlet architectures, applications, and open research challenges. Figure 2.2 shows a cloudlet server being accessed by the mobile devices. A large body of work exists in this domain. In this section, we highlight limited work that has been selected based on the number of citations and novelty.

Fesehaye *et al.* proposed a design of a cloudlet-based network and an adaptive decision-based service architecture (Fesehaye, Gao, Nahrstedt, & Wang, 2012). The cloudlet-based network consists of the cloudlets server and mobile nodes affiliated with the closer cloudlet. Two new routing algorithms are proposed following the centralized

and distributed approaches. Although the cloudlet-based approach outperforms the cloud-based approach in terms of access latency, it still suffers from the initial delays due to service discovery, authentication of mobile users, and joining process. Another limitation is on the size of the network that affects the cloudlet-based approach. Mirror server-based framework has been proposed in (Zhao et al., 2012) that maintains the VM template for each of the mobile device inside the telecommunication network computing infrastructure. The framework feasibility is tested by designing a protocol, performing scalability test, and developing synchronization methods. The framework has high synchronization overhead that consumes significant network bandwidth. In case of the network disconnection, the mobile device has to re-execute the whole application in the cloud.

Kovachev *et al.* have presented the XMPP-based mobile cloud middleware (Kovachev, Cao, & Klamma, 2012) that provides dynamic application partitioning and adaptive offloading to the nearby resource-rich devices. The offloading decision is based on a context-aware cost model. The objective of the model is to make intelligent decisions considering the constraints imposed by the wireless environments. The middleware must exchange a number of control messages especially when the mobile device and network conditions are highly dynamic, resulting in high overhead. The proposed solution suggests to re-execute the application locally when the network disconnection occurs. Verbelen *et al.* proposed a fine-grained dynamic cloudlet approach (Verbelen, Simoens, De Turck, & Dhoedt, 2012b) that deploys a cloudlet on any device in local area network with sufficient available resources. The unit of deployment is a component. Although the dynamic cloudlet overcomes the limitations of static cloudlet, there exist challenges of optimal tasks mapping that are critical for optimizing application execution performance. Moreover, the mobile device has to re-execute the application locally when the network disconnection occurs.

The best mechanism to cater the user mobility while executing the mobile application on cloud/cloudlet has been proposed by Gordon *et al.* (Gordon et al., 2012) in the form of COMET. The framework transparently offloads the multi-threaded applications

Figure 2.3: Mobile Ad-hoc cloud

to a server in local proximity. COMET incorporates the workload of machines in the decision making process of threads migration. Moreover, COMET leverages on the distributed shared memory techniques, such as field level granularity, to attain the execution state consistency across the mobile device and the cloud server. Although COMET transparently migrates the code to the cloud, the framework has high synchronization overhead between the mobile device and the cloud server. Furthermore, in case of disconnection, the task can be resumed on the mobile device from last point of synchronization; however, upon reconnection, the application is not transferable to another cloud/cloudlet or for that matter, even the same cloud/cloudlet. COMET also uses periodic synchronization as proposed in the present work; however, the performance of the synchronization and the synchronization parameters have not been studied at all.

2.1.3 Mobile Ad-hoc Cloud Frameworks

The mobile ad-hoc clouds (Huerta-Canepa & Lee, 2010) is a group of mobile devices that shares resources with each other in the local vicinity to reduce the resource limitations of individual devices. Figure 2.3 shows mobile ad-hoc cloud. These clouds pose

significant additional challenges. This includes management responsibilities for mobile devices, authentication, resource monitoring and task scheduling in distributed manner, while ensuring minimum energy consumption. A selective set of literature is presented here.

A framework for mobile ad-hoc cloud has been presented by Huerta *et al.* (Huerta-Canepa & Lee, 2010). The framework enables a mobile device to discover other mobile devices in the local vicinity that are in a stable mode. To provide the virtual cloud service, an architecture is proposed that has five components: application manager, context manager, offloading manager, resource manager, and peer-to-peer manager. Each of the mobile device is required to detect the neighbours and keep the profile updated. The framework does not provide the support for handling the mobility during the execution of an application. Pocket cloudlet architecture (Koukoumidis, Lymberopoulos, Strauss, Liu, & Burger, 2012) leverages on the large available memory capacity of the mobile devices to alleviate the latency and energy issues in accessing the distant cloud services. Although the pocket cloudlet maximizes the hit rate and minimizes the overall service latency and energy consumption, the pocket cloudlet does not ensure the data integrity for a user who is accessing the data from the pocket cloudlet on his own device. The research work also does not discuss the incentives for mobile users to share the limited resources of the mobile device.

2.1.4 Hybrid Frameworks

Some of the existing frameworks e.g. (Satyanarayanan et al., 2009; Goyal & Carter, 2004; Kovachev, Yu, & Klamma, 2012; Cuervo et al., 2010; Marinelli, 2009) leverage on the hybrid platforms available in different clouds. The hybrid frameworks overcome the limitations of both clouds as well as cloudlets. Figure 2.4 illustrates the execution environment of hybrid frameworks.

Satyanarayanan *et al.* (Satyanarayanan et al., 2009) proposed a dynamic VM synthesis approach that is employed for migrating an application. Mobile device sends a small

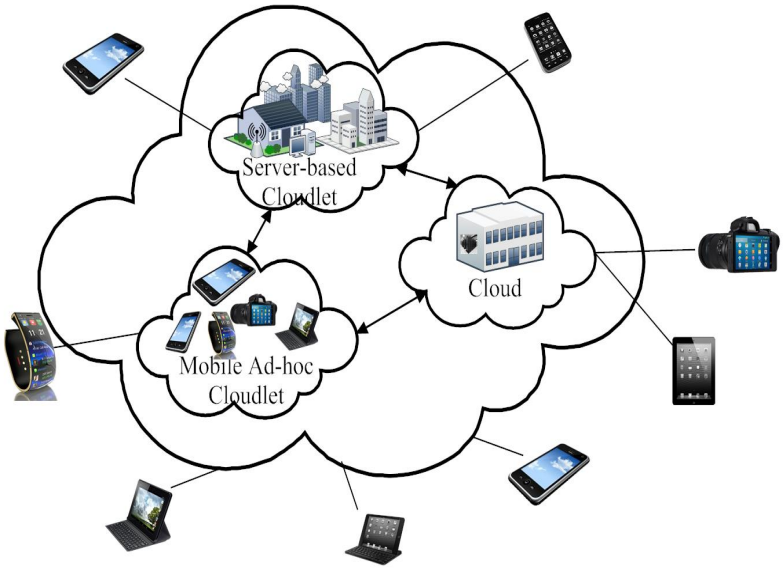


Figure 2.4: Hybrid MCC Environment

VM overlay to the cloudlet that already has a base VM from which the overlay VM was derived. Thereafter, the cloudlet infrastructure derives the launch VM by applying overlay to the base VM. Unlike traditional VM migration, the VM-synthesis has significantly reduced the data transfer size. Ha *et al.* (Ha et al., 2013) improved the performance of dynamic synthesis process by reducing the overlay size using a series of optimization. An aggressive deduplication approach eliminates the duplicated data between different sources, such as Input/Output cache and virtual memory, to shrink the VM overlay. The shrinked size of VM overlay reduces the transmission delay and energy consumption on the mobile device for transmission. Another approach of bridging the semantic gap between the low-level representation of memory and disk, and higher-level abstraction also reduces the additional data in VM overlay. In VM overlay, the contents of memory pages freed or files deleted should not be included in overlay. Lastly, the pipelining reduces the three delay-intensive steps of VM synthesis by beginning the later steps before the preceding ones complete.

The VM based mechanisms do not propose any mechanism for mobility estimation. The overlay VM is created on demand. However, these solutions can be complemented by the mobility prediction mechanisms (Modares, Moravejosharieh, Lloret, & Salleh, 2014; Pirozmand, Wu, Jedari, & Xia, 2014; Vu, Nguyen, Nahrstedt, & Richerzhagen, 2015) to

automate the process of overlay creation based on predicted user mobility. The reduced size of overlay VM still requires up to 12 seconds of transfer time, which may not be long enough to allow complete transfer before disconnection.

Mobile Augmentation Cloud Services (MACS) (Kovachev, Yu, & Klamma, 2012) is an adaptive middleware framework that provides lightweight application partitioning, seamless computational offloading, and resource monitoring. MACS enables mobile applications to take benefit from the seamless computational offloading into a remote or nearby cloud. The developers do not need to change the application model to run an application in the cloud. Application modules are divided into two groups. One group runs on a mobile device and the other group runs in the cloud. The partitioning decision is transformed into an optimization problem that is solved by a solver to dynamically partition the application. MACS continuously monitors the environment parameters and service execution, and adapts the partitioning and offloading. However, MACS requires a strong developer support for structuring the code in a model. An additional layer of proxy service adds more delay in the application execution. The framework re-executes the application on the mobile device when the network disconnection occurs.

A lightweight cyber foraging framework is devised in (Goyal & Carter, 2004) for mobile and embedded resource constrained devices. A lightweight discovery protocol is used to discover potential surrogates. The proposed framework follows a client/server architecture that enables the mobile devices to offload the compute-intensive tasks on the surrogate server to utilize the computational resources of the server. The framework also provides a support to simultaneously configure the multiple surrogate servers. A single server can also run configurable number of virtual servers with the features of elasticity, resource control, isolation, and simple cleanup. Each migrated application runs on isolated virtual server. The cryptographic measures are employed by the framework to ensure the secure communication between mobile device and surrogate server. The framework uses low overhead client side authentication and ciphers. The proposed solution facilitates the end user with the benefit of low latency communication and minimal

overhead of privacy and security.

Although the proposed framework is lightweight as it does not require client to run any middle-ware software, the framework is based on VMs that suffer from VM deployment and management overheads. Due to the time consuming VM mechanisms, the solution is unfavorable for the execution of an application in the MCC. Moreover, the framework uses the component annotation that requires a developer support, and existing applications must be rewritten to incorporate a component level annotation. The framework also re-executes the application on a mobile device when the network disconnection occurs.

MAUI (Cuervo et al., 2010) is proposed as an energy-aware fine grained, method level mobile application offloading mechanism. MAUI supports a semi-dynamic partitioning; wherein, programmers annotate an application with a considerably less effort. Although MAUI significantly improves the battery life of a mobile device, it does not address scalability, QoS, and transmission latency. Therefore, MAUI lacks implementing favorable measures to ensure the seamless execution of an application within the MCC. Moreover, the framework re-executes the application on a mobile device when the network disconnection occurs.

A Hyrax-based platform is designed in (Marinelli, 2009) to provide cloud computing services on a group of mobile devices and servers. The platform employs a mechanism for tolerating node departure. Hyrax is developed by extending the Hadoop (White, 2012) framework for Android mobile device. The MapReduce (Dean & Ghemawat, 2008) guidelines are followed in the development of Hyrax while using Hadoop as an open source implementation of the MapReduce. The platform relies on a centralized server and a group of mobile devices. The server runs two client processes of MapReduce, JobTracker and NameNode, to coordinate the overall execution process on a group of mobile devices. The mobile devices run two Hadoop processes, DataNode and TaskTracker, to accept tasks from the server. The Hyrax facilitates applications in transparently accessing and utilizing the distributed resources. Similar to the cloudlet access, the accessibility of

the cloud is also supported when sufficient resources are unavailable on the nearby mobile devices.

The discovery of a centralized server, access, and connection establishment among the servers is a time-intensive task that induces the delay before executing the actual application. The high overhead of the Hyrax also induces a delay while running the application on the cloud. Moreover, the framework re-executes the application on a mobile device when the network disconnection occurs.

A decentralized computation offloading game (X. Chen, 2015) is designed to model the offloading decision making problem among mobile users. The game considers both computation and communication cost of MCC. The authors have analyzed the computation offloading game in two types of wireless access scenarios: homogeneous and heterogeneous. In homogeneous scenario, the existence of Nash equilibrium is guaranteed by admitting the beneficial cloud computing group structure. In heterogeneous scenario, the existence of Nash equilibrium is guaranteed by admitting the finite improvement property. The decentralized computation offloading mechanism enables the mobile users to make the decisions locally that reduces the controlling and signaling overhead of the cloud and makes it scalable. Although the solution reduces the controlling and signaling overhead of the cloud, structuring a game such that an equilibrium is always reachable is difficult.

An offloading framework to solve the optimal application management problem in MCC is presented in (S. Chen, Wang, & Pedram, 2013). The execution is either performed locally or remotely on a cloud server. The local processing unit can dynamically adjust the processing speed and power consumption of the processing unit by employing dynamic voltage and frequency scaling (DVFS). Moreover, the transmitter can adaptively select the most appropriate modulation scheme and bit rate for offloading request considering the channel capacity, the number of waiting requests, and server congestion levels. A semi-Markov decision process is used to model the mobile device where actions are decision pairs (bit rate, DVFS level) for the transmitter and the local processor in the mobile device. The objective function of the framework is a linear combination of energy loss of

the mobile device battery and average request response time. Moreover, the framework can derive the corresponding transmission scheme, offloading rate, and optimal DVFS policy for different server congestion levels, wireless environment, and workload characteristics. The identification of optimal transmission scheme, offloading rate, and optimal DVFS policy needs continuous monitoring of the running environment.

2.2 Application Performance Enhancement by the Frameworks

The mobile devices access the clouds for two reasons. Firstly, the clouds and cloudlets have abundant resources in the form of storage, services and applications and computational capacity. With reference to this thesis, computational capacity is the major factor because of which the clouds are accessed. Secondly, to save the battery power and enhance the battery life of the device. In this section, we present the taxonomy with reference to application performance enhancement in terms of execution time and computational capability. Note the network disconnections directly impact the execution time, making it a factor of interest. The taxonomy is an outcome of a comprehensive survey of the state-of-the-art frameworks presented in Section 2.1. This section also investigates the advantages and disadvantages of application performance enhancement approaches. The strengths and weaknesses of the state-of-the-art frameworks with respect to application execution time are also presented.

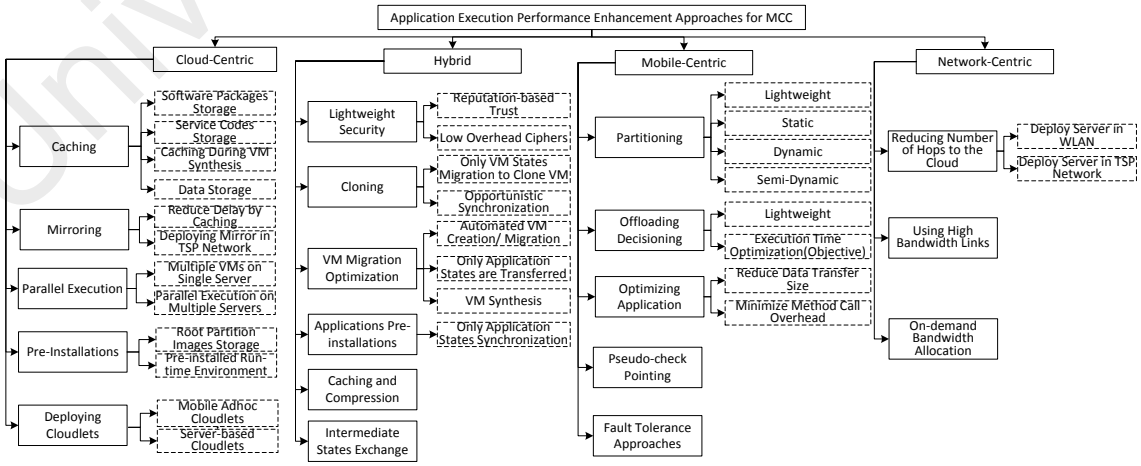


Figure 2.5: Taxonomy of Application Performance Enhancement Approaches Employed by the Frameworks for the MCC

2.2.1 Taxonomy of Application Performance Enhancement

Figure 2.5 shows the thematic taxonomy of the application performance enhancement approaches employed by the execution frameworks for the MCC. The approaches are categorized based on implementation locations into four classes, namely: mobile-centric, cloud-centric, network-centric, and a hybrid approaches.

2.2.1.1 Cloud-centric Approaches

Cloud-centric approaches are implemented in the cloud to enhance the application execution performance. Cloud-centric approaches are caching (Zhao et al., 2012; Kovachev, Yu, & Klamma, 2012; Satyanarayanan et al., 2009; Fesehaye et al., 2012; Goyal & Carter, 2004), mirroring (Zhao et al., 2012), parallel execution (Kosta et al., 2012), pre-installations (Goyal & Carter, 2004), and deploying cloudlets (Verbelen et al., 2012b; Marinelli, 2009; Fesehaye et al., 2012; Satyanarayanan et al., 2009; Huerta-Canepa & Lee, 2010). Table 2.1 shows the comparative summary of the frameworks based on the application execution performance enhancement cloud-centric approaches.

Caching: Caching enables the framework to store the remote data locally to cope with longer WAN delays. Caching is further classified into four classes according to the contents, namely: software package caching (Goyal & Carter, 2004), service code caching (Kovachev, Yu, & Klamma, 2012), VM synthesis caching (Satyanarayanan et al., 2009), and data caching (Fesehaye et al., 2012). Software package caching approach employs caching to store software packages. Such an approach allows the trusted user to install packages directly without facing any download delay. Service code caching approach is used to store the service code (jar files) when client sends data to the cloud for the very first time. The VM synthesis caching approach is used either as a pre-fetching technique to reduce the VM synthesis delay by caching the VM overlay or caching the VM launch for future use. Lastly, the data caching approach temporarily stores the data such as video streams on the local cloudlet. Thereafter, the cloudlet sends the cached data to the subsequent mobile users on their requests. The caching of data on the local cloudlet reduces

the latency involved in video streaming.

The caching of contents on local servers reduces the WAN latency; thereby, minimizing data access delay. The caching integration with pre-fetching in the VM synthesis process reduces the VM synthesis delay. Moreover, the caching also lessens the network redundant traffic. In spite of the aforementioned advantages, the caching increases the storage overheads and cost in the cloud. Moreover, the cached data can be outdated for the user.

Mirroring: Mirroring is another cloud-centric approach employed to improve the application execution performance within MCC. The framework presented in (Zhao et al., 2012) employs the mirroring approach to reduce the delay by caching at the mirror for downloading and uploading. When a mobile device downloads a bulk of data from the Internet, a mirror server first caches the data and then the mobile device can download the data from the mirror cache. Similarly, to upload data, the mobile device merely sends a command to the mirror server that replays the command with the data. Even for multiple receivers, a mobile device needs only to send a command just for a single time. The performance of application can also be improved by deploying a mirror in the nearby telecommunication service providers (TSPs) network.

The mirror deployment reduces the operational overhead of a mobile device by transmitting only commands of operations from mobile device; actual operations on data are performed on the mirror server. The mirror-based frameworks alleviate the redundant traffic in the network. Moreover, mirror server minimizes the data access latency by caching the data on the server for subsequent users; thereby, optimizing the response time of applications. However, the mirroring approach increases storage cost at cloud. The mirror server requires management for updates, upgrades, new versions, and bug fixes. Furthermore, the mirror-based frameworks require synchronization between the mobile device and the mirror server.

Parallel Execution: Parallel execution is employed to minimize the application execution time in the cloud data center. We classify the parallel execution approaches into two classes. One class uses multiple VMs on a single cloud server and the other performs parallel execution on multiple servers. Using multiple VMs on a single server reduces the server identification and selection time; whereas, the VM deployment on multiple servers mandates the identification of multiple servers and to send two distinct requests to each server. Using the VM on multiple servers increases the network overhead and operational complexity. A framework proposed in (Kosta et al., 2012) uses the aforementioned approaches to improve the application execution performance. Elastic mobile applications (X. Zhang et al., 2010) leverage on parallel processing in cloud to improve the response time and load balancing by deploying weblets on multiple servers within the cloud.

The parallel execution reduces application execution time in cloud and facilitates in load balancing. The parallel execution support makes the framework highly scalable. However, the parallel execution consumes more power and requires more hardware. Moreover, the parallel execution mandates the identification of interdependency among tasks to avoid deadlocks.

Pre-installations: Pre-installation reduces the initiation and preparation delay of remote application execution by providing access to the already installed software packages. The pre-installations are classified into two categories. One category facilitates the mobile user by installing root partitioned images with the necessary applications, system boot-up scripts, and essential software packages on the cloud server. The researchers in (Goyal & Carter, 2004) encourage the usage of already installed applications, system boot-up scripts, and software packages to reduce the pre-execution delay. The other category provides a pre-installed run-time environment (Lee, 2012) that facilitates in the deployment of mobile applications into the cloud to reduce the remote application execution initialization delay.

The pre-installation of applications minimizes run-time transmission of data by elim-

inating the need of migrating the application from the mobile device to the cloud server. However, the pre-installation increases the resource consumption cost in the cloud if the applications and software packages are not frequently used. The pre-installation also increases the cloud service provider's services maintenance overhead.

Deploying Cloudlets: Deployment of cloudlets enhances application execution performance by exploiting the capabilities of trusted resource-rich local system either in the WLAN or in the TSP's network. We classify these cloudlets based on the deployment structure into three subclasses: (a) infrastructure-based/ static/ server-based cloudlet, (b) infrastructure-less/ mobile ad-hoc cloudlet, and (c) virtualized infrastructure-based/ elastic cloudlet. M. Satyanarayanan *et al.* (Satyanarayanan et al., 2009) present an infrastructure-based cloudlet that is deployed on the local server in the WLAN. Infrastructure-less or mobile ad-hoc cloudlet is proposed in (Huerta-Canepa & Lee, 2010) that does not require any local server for the deployment of cloudlet while all of the mobile devices with sufficient idle resources form a mobile ad-hoc cloudlet. An elastic cloudlet is presented in (Verbelen et al., 2012b), in which a component level application migration is performed instead of a whole VM migration. The approach provides flexible allocation of resources and facilitates in prioritizing the deployment of real-time components of an application in the cloudlet; whereas, delay tolerant components are offloaded in a remote cloud.

The cloudlet deployment closed to user reduces the WAN latency and minimizes the data transport cost. The cloudlet-based frameworks do not need to connect to the Internet. In spite of the aforementioned advantages provided by the cloudlet, the cloudlet requires network isolation for outside users to prevent them from accessing the private network resources. Moreover, the practical realization of the cloudlet deployment requires investigation of incentives for cloudlet service providers to deploy cloudlet. The cloudlet deployment also requires a well-defined charging policy as per usage basis. The cloudlet service providers need more resources in their local networks to provide the on-demand

services and resources to the MCC users.

Table 2.1: Comparison of frameworks based on application execution performance enhancement cloud-centric approaches

Application Execution Frameworks	Cloud Usage Overhead	Deploys Mirror	Pre-execution Delay	Caching Support	Parallel Execution Support
MAUI (Cuervo et al., 2010)	High	No	High	No	No
Virtual Mobile Cloud Computing (Huerta-Canepa & Lee, 2010)	High	No	High	No	No
Secure Cyber Foraging (Goyal & Carter, 2004)	High	No	Low	Yes	No
VM-based Cloudlets (Satyanarayanan et al., 2009)	Low	No	Medium	Yes	No
Optimized VM-based Cloudlets (Ha et al., 2013)	Low	No	Medium	Yes	No
Augmented Smartphone Application (Chun & Maniatis, 2009)	High	No	High	No	No
Cloudlets-based Network (Fesehaye et al., 2012)	High	No	High	Yes	No
Mirror Server-based Framework (Zhao et al., 2012)	High	Yes	Low	Yes	No
Pocket Cloudlets (Koukoumidis et al., 2012)	Low	No	Low	Yes	No
AIOLOS (Verbelen et al., 2012a)	High	No	High	Yes	Yes
ThinkAir (Kosta et al., 2012)	High	No	High	No	Yes
Hyrax (Marinelli, 2009)	High	No	High	No	No
Cloudlet (Verbelen et al., 2012b)	Low	No	Medium	No	No
COMET (Gordon et al., 2012)	Low	No	Low	Yes	Yes
MACS (Kovachev, Yu, & Klamma, 2012)	High	No	High	Yes	No

2.2.1.2 Hybrid Approaches

Hybrid approaches are implemented on both the mobile device and the cloud server for improving the application execution performance in the MCC. Hybrid approaches include provisioning of lightweight security mechanisms (Goyal & Carter, 2004), cloning (Chun & Maniatis, 2009; Chun et al., 2011), (Hung et al., 2012; Satyanarayanan et al., 2009), optimizing VM migration (Hung et al., 2012; Satyanarayanan et al., 2009), pre-installation of applications that require only states to be transferred (X. Zhang et al., 2010), and lastly, caching and compression of the data and meta-data (Mao, Xiao, Shi, & Lu, 2012). Table 2.2 shows the comparative summary of frameworks based on the application execution performance enhancement hybrid approaches.

Lightweight Security: Security is one of the main challenges that are obstructing the major deployment of clouds. Security issues in cloud are investigated in (Xiao & Xiao, 2013; Khan, Mat Kiah, Khan, & Madani, 2012) that are necessary to be resolved to rapidly increase the growth rate of the clouds. Aside from a number of security issues

in the cloud, existing security mechanisms induce delay in the application execution that is obstructing the aim of the seamless application execution. The computational delay of security mechanisms can be reduced by designing and employing lightweight security approaches. Lightweight security is classified into reputation-based trust establishment (Satyanarayanan et al., 2009), low overhead ciphers, and authentication mechanisms (Goyal & Carter, 2004). The lightweight approaches improve the application performance by reducing the delay involved in cipher execution, trust establishment, and authentication. The lightweight security mechanisms reduce the time taken by authentication process and minimize the resources consumption. Therefore, the application response time in MCC can be reduced. However, the security of the data can be compromised because of the employment of the lightweight security mechanisms and relying on reputation-based trust.

Cloning: The cloning improves the response time of an application by deploying the mobile device clone in the cloud. The mobile device clone that needs to be deployed in the cloud, requires only part of the application execution to be offloaded from mobile device into the cloud. The pre-installed mobile device clone alleviates the overhead involved in initiation and preparation of the application execution. During the application execution process, the states are migrated to the mobile device clone. The research works of (Chun & Maniatis, 2009) and (Chun et al., 2011) present the CloneCloud frameworks that execute an application on the nearby and remote servers. Moreover, an incremental checkpointing and opportunistic synchronization are used to improve the application execution performance in MCC. The cloning minimizes the application initiation and preparation delay; thereby, improving the application overall response time. However, the mobile device clone deployment increases the maintenance overhead for the cloud service provider. The cloning also increases the resource consumption cost for the cloud service provider if the applications are not frequently used. Moreover, providing the clones of all available mobile devices is not practical solution in the MCC.

Optimizing VM Migration: The VM migration incurs a significant overhead that is investigated in (Spata & Rinaudo, 2011). The VM-based frameworks employ three strategies to optimize the migration process: (a) automation of VM creation and migration (Hung et al., 2012), (b) transfer of only application specific states instead of the whole VM (Hung et al., 2012), and (c) the VM synthesis (Satyanarayanan et al., 2009). The optimized migration approaches alleviate the delay involved in the VM creation and migration. The transfer overhead is reduced by transferring only application relevant states and by replacing the full VM migration with the VM synthesis that requires low bandwidth and takes less transmission time. However, the VM creation, migration, and deployment in the cloud is time-intensive process. Although the VM synthesis reduces the migration overhead, the VM synthesis requires installation of VM base in the cloud, which arises the compatibility issues.

Applications Pre-installation: Elastic mobile application framework presented in (X. Zhang et al., 2010) employs an approach wherein the weblets are pre-installed on the remote server, and during the execution only states need to be transferred for execution of an application. The pre-installations of weblets reduce data transfer size, which lessens transmission delay, lowers the bandwidth consumption, and conserves the battery power. The application pre-installations reduce the application initiation and preparation time. However, application pre-installation increases the maintenance overhead on cloud server provider. Moreover, the pre-installed applications consume storage even if the applications are not frequently used.

Caching and Compression: The data caching and compression aid in locally caching the data and employing an adaptive compression to reduce the amount of transferred data (Mao et al., 2012). The caching and compression approach not only reduces the latency but also the storage and network cost. Moreover, the compression significantly reduces the data transfer size. However, the compression increases processing overhead

on resources-constrained mobile device; thereby, consuming a significant amount of battery power.

Intermediate States Exchange: To address the network disconnection issue that arises mainly because of user mobility, the execution frameworks exchange the intermediate states from the cloud server with the mobile device. Hence, the mobile device either can re-offload the only remaining portion of the execution or resumes the execution locally by itself. The successful exchange of intermediate states enables the mobile device to resume the application after the disconnection by itself, thereby minimizing the application execution time. Optimized VM-based cloudlet (Ha et al., 2013) and COMET (Gordon et al., 2012) exchange the intermediate results with the mobile device. The optimized VM-based cloudlet exchanges VM-overlay with the mobile device, which sends the received overlay to the newly visited cloudlet for execution. The newly visited cloudlet resumes the application from the point where the previously visited cloudlet left the execution. COMET also exchanges the intermediate states from the cloud to the mobile device. However, COMET cannot reconnect back with the cloud server on the connection re-establishment. Moreover, both solutions are based on VM-level information exchange.

Table 2.2: Comparison of frameworks based on application execution performance enhancement hybrid approaches

Application Execution Frameworks	**Security Overhead**	**Data Transfer Overhead**	**VM Migration Overhead**
COMET (Gordon et al., 2012)	N/A	Low	Low
Elastic Application Framework (X. Zhang et al., 2010)	High	Low	N/A
CloneCloud (Chun et al., 2011)	Medium	High	High
VM-based Cloudlets (Satyanarayanan et al., 2009)	Medium	Low	Low
Optimized VM-based Cloudlets (Ha et al., 2013)	Medium	Low	Low
Secure Cyber Foraging (Goyal & Carter, 2004)	Low	High	N/A
Mobile Application Execution Framework (Hung et al., 2012)	High	Medium	Medium

2.2.1.3 Mobile-centric Approaches

In mobile-centric approaches, the main functionality for improving the application execution performance is implemented on the mobile devices. The mobile-centric approaches consist of partitioning, offloading decisioning, optimizing application, pseudo-checkpointing, and fault tolerance approaches. Table 2.3 shows the comparative summary

of the frameworks based on the application execution performance enhancement mobile-centric approaches.

Table 2.3: Comparison of the frameworks based on the application execution performance enhancement mobile-centric approaches

Application Execution Frameworks	Partitioning Type	Partitioning Overhead	Offloading Overhead	Method Call Overhead	Data Transfer Overhead	Fault Tolerance	Transmission Delay
COMET (Gordon et al., 2012)	N/A	N/A	Med.	Low	Low	Yes	Low
Decentralized Computation Offloading Game (X. Chen, 2015)	N/A	N/A	Med.	N/A	Low	No	Low
Semi-Markovian Decision Process-Based Offloading (S. Chen et al., 2013)	N/A	N/A	Med.	N/A	Low	No	Low
Coalesced offloading framework (Xiang et al., 2014)	N/A	N/A	Low	Low	Low	No	Low
Replicated Application Execution Framework (Lee, 2012)	DyP	Med.	N/A	High	Med.	No	Med.
XMPP-based Middleware (Kovachev, Cao, & Klamma, 2012)		High	Low	High	High	No	Med.
Elastic Application Framework (X. Zhang et al., 2010)		High	High	High	High	Yes	High
Data Stream Application Partitioning Framework (Yang et al., 2012)		Low	High	High	High	No	High
AIOLOS (Verbelen et al., 2012a)		High	High	Low	High	Yes	Med.
ThinkAir (Kosta et al., 2012)		High	High	High	High	Yes	High
Dynamic Deployment & Quality Adaptation Framework (Verbelen, Stevens, Simoens, De Turck, & Dhoedt, 2011)		High	High	High	High	Yes	High
MACS (Kovachev, Yu, & Klamma, 2012)		Low	High	High	Low	No	Med.
CloneCloud (Chun et al., 2011)	StP	Low	High	High	Low	No	Low
MAUI (Cuervo et al., 2010)	SdP	Med.	High	Low	Low	Yes	Low
Calling the Cloud (Giurgiu et al., 2009)		Med.	High	High	High	No	Med.
Augmented Smartphone Application (Chun & Maniatis, 2009)		Med.	High	High	Low	Yes	Low

Note: Med.: Medium, N/A: Not Applicable, DyP: Dynamic Partitioning, StP: Static Partitioning, SdP: Semi-dynamic Partitioning

Partitioning: Partitioning is performed in three ways, static (Chun et al., 2011), dynamic (Chun & Maniatis, 2010; Lee, 2012; Kovachev, Cao, & Klamma, 2012; X. Zhang et al., 2010; Yang et al., 2012; Verbelen et al., 2012a; Kosta et al., 2012; Verbelen et al., 2011; Kovachev, Yu, & Klamma, 2012) and semi-dynamic (Giurgiu et al., 2009; Chun & Maniatis, 2009; Cuervo et al., 2010). The static partitioning requires developer support to annotate the remotely executable methods. Therefore, the static partitioning does not

incorporate the conditions of the run-time environment during the partitioning decision. However, in the dynamic partitioning, a dynamic and adaptive decision engine takes the context information as input, computes partitions, and adapts according to the new partitioning results. Although the static partitioning performs lesser run-time computations, the static partitioning approach does not optimally partition the application. The semi-dynamic partitioning takes the advantages of both static and dynamic partitioning. The semi-dynamic partitioning performs static annotation to reduce the run-time computations and also incorporates the dynamic environment parameters for run-time partitioning.

The lightweight application partitioning is vital for enhancing application execution performance in the MCC that alleviates the complexity of migration process (Ahmed, Akhunzada, et al., 2015; J. y. Liu et al., 2015). The lightweight partitioning approaches employed by the frameworks are reported in (Chun & Maniatis, 2010; Kovachev, Yu, & Klamma, 2012). The proposed approach in (Chun & Maniatis, 2010) employs a model that predicts the costs of different partitions. Thereafter, a simple approximate optimizer quickly solves the partitioning problem based on predicted costs. The authors transformed an integer linear programming (ILP) problem into a linear programming (LP) problem to attain speed up over the cost of accuracy. The lightweight application partitioning enhances the execution performance by reducing the complexity of the problem and performs a quick partitioning. The lightweight partitioning reduces consumption of mobile device battery power. The static partitioning does not require the profiling so it is relatively less complex in nature. On the other hand, the dynamic partitioning incorporates the dynamic conditions of the mobile device and network for partitioning, so performs optimal decision. Moreover, the dynamic partitioning suffers with the complexity of profiling. Similar to static partitioning, semi-dynamic partitioning is less complex, however, the semi-dynamic partitioning also incorporates the dynamic conditions of the environment.

Offloading Decisioning: An offloading decision algorithm is required to be lightweight with an objective function of minimizing the response time of an application in MCC. The researchers in (Kovachev, Yu, & Klamma, 2012; Kovachev, Cao, & Klamma, 2012) proposed an offloading middleware that incorporates network latency in the offloading decision with the objective of optimizing the overall application execution time. The XMPP-based middleware (Kovachev, Cao, & Klamma, 2012) reduces the offloading overhead by sharing the cloudlet-provided VM for all of the attached mobile users. The offloading decision algorithms aim to optimize the execution cost (Chun & Maniatis, 2010), to minimize the energy consumption (Cuervo et al., 2010), to reduce the execution time (Kosta et al., 2012; Kovachev, Yu, & Klamma, 2012; Giurgiu et al., 2009; Ou, Yang, & Zhang, 2007), to optimize the throughput (Yang et al., 2012), and to minimize the estimated network latency (Verbelen et al., 2012a). Such essential objectives make the offloading mechanism favorable to execute on a mobile device because of the cost-effective lightweight features. The offloading scheme that aims to minimize the execution time or to optimize the estimated network latency is a favorable choice for the optimized application execution.

The incorporation of the communication latency in the offloading objective function minimizes the application migration time and optimizes data transfer cost; thereby reducing the overall application execution cost and time. However, the run-time application offloading decision algorithms have high computational complexity; thereby consuming battery power of the mobile device.

Application Optimization: An application optimization focuses on the reduction in the amount of transferred data and method calls overhead. The amount of transferred data is reduced by implementing incremental state deltas and fine-grained code offloading support (Cuervo et al., 2010). The method call overhead is reduced either by reducing the number of method calls (Cuervo et al., 2010) or using callback functions (Verbelen et al., 2012a). The number of method calls are reduced by implementing a set of similar

operations in a single method. The callback function consumes less CPU resources and does not exchange much data. The method call overhead increases the delay especially if the method is called remotely. The huge data size increases the transmission delay; thereby, increasing the overall execution time. Therefore, the application optimization is also an important feature to enhance the performance in MCC. The application optimization minimizes the amount of data sent on network; thereby reducing the network bandwidth consumption and conserving the mobile device battery. However, optimizing the application design requires modification in application. Moreover, the optimization of the application requires programmer support.

Pseudo check-pointing: The pseudo-checkpointing does not need to explicitly save the states, which reduces the overall states storage and transmission overhead (Hung et al., 2012). The reduction in the states storage and transmission overhead enables the fast transmission and enhances the application execution performance. Pseudo-checkpointing minimizes the application states storage size and transmission overhead; thereby optimizing the consumption of network bandwidth and mobile device battery. Moreover, pseudo-checkpointing minimizes the network usage cost. In spite of aforementioned advantages, the pseudo-checkpointing has two limitations: (a) requires modification in applications and (b) needs programmer support.

Fault Tolerance Approaches: Fault tolerance approaches reduce the disruption time caused by either the link failures or server failures while an application is executed in MCC. The frameworks that support fault tolerance can execute the application locally on mobile device, when the cloud server is inaccessible. The fault tolerance approaches alleviate the disruption during execution and provide user transparent failure detection mechanism. The framework supported with fault tolerance approaches gracefully degrades the application performance. However, to provide the fault tolerance support, continuous monitoring mechanism is required to detect the failure. The proactive fault

tolerance approaches require more functionality; thereby inducing high complexity.

2.2.1.4 Network-centric Approaches

The network-centric approaches refer to a set of functions that utilize network related information for improving application execution management in MCC environment. The network-centric solutions are employed in a network to mitigate the network related issues, such as high WAN latency, jitter, and packet losses, to enhance the application execution performance in MCC. The network-centric solutions employ three types of approaches that include reducing number of hops to the cloud, using high bandwidth links, and on-demand bandwidth allocation. Table 2.4 shows the comparative summary of the frameworks based on the application execution performance enhancement network-centric approaches.

Table 2.4: Comparison of frameworks based on the application performance enhancement network-centric approaches

Application Execution Frameworks	**Reduction in Number of Hops**	**Usage of High Bandwidth Links**	**Guaranteed Bandwidth**
COMET (Gordon et al., 2012)	Yes	Yes	No
Cloudlets (Verbelen et al., 2012b)	Yes	Yes	No
MAUI (Cuervo et al., 2010)	Yes	Yes	No
Mirror Server-based Framework (Zhao et al., 2012)	Yes	No	No
AIOLOS (Verbelen et al., 2012a)	No	Yes	No
ThinkAir (Kosta et al., 2012)	No	Yes	No
Augmented Smartphone Application (Chun & Maniatis, 2009)	Yes	Yes	No
MACS (Kovachev, Yu, & Klamma, 2012)	Yes	Yes	No
Virtual Mobile Cloud Computing (Huerta-Canepa & Lee, 2010)	Yes	Yes	No
VM-based Cloudlets (Satyanarayanan et al., 2009)	Yes	Yes	No
Cloudlets-based Network (Fesehaye et al., 2012)	Yes	Yes	No
Secure Cyber Foraging (Goyal & Carter, 2004)	Yes	Yes	No
Hyrax (Marinelli, 2009)	Yes	Yes	No
Mobile Application Execution Framework (Hung et al., 2012)	No	Yes	Yes
Pocket Cloudlets (Koukoumidis et al., 2012)	Yes	Yes	No

Reducing Number of Hops: The number of hops to the cloud are reduced by bringing the cloud closer to mobile user either by deploying servers in the WLAN (Fesehaye et al., 2012) or in the TSP's network (Zhao et al., 2012). The reduction in the number of hops eliminates the issues of WAN high latency, jitter, and packet losses that significantly improves application response time. Moreover, reducing the number of hops to the cloud

minimizes the data transport network cost. The framework does not require Internet connection to access the cloud server if the cloud server is deployed inside the local network. However, the servers in local networks are not always accessible due to security policies. The cloud deployment in locally available servers also requires isolation from local networks to keep the outside users away from the private resources.

Using High Bandwidth Links: The high bandwidth links are deployed by using the WiFi (Fesehaye et al., 2012) and 3G networks (Zhao et al., 2012) for offloading the application. The high bandwidth links reduce the transmission time during the offloading of an application in MCC. However, the use of high bandwidth links in the cloudlet-based frameworks is more beneficial than the cloud-based frameworks because the path to the cloudlet has a direct link. Using the high bandwidth wireless links to access the cloud does not ensure that the same bandwidth will be allocated along the whole path. The path may have bottleneck links to the cloud. The data of cloud-based frameworks may have to go through such bottleneck link across the Internet; thereby reducing the overall application performance. Using the high bandwidth links provides high data transfer rate across the link. However, use of high bandwidth links increases monetary cost.

On-demand Bandwidth Allocation: The on-demand bandwidth is allocated to the users' traffic to ensure the bandwidth reservation according to the traffic requirements within the network (Hung et al., 2012). The on-demand bandwidth reservation approach helps in ensuring QoS for each application; therefore, it helps in improving the application execution performance by providing sufficient bandwidth to each of the flow according to the requirements. However, on-demand bandwidth allocation requires bandwidth negotiation with the network service provider. Moreover, the allocation of bandwidth to the network flows also requires fairness among the flows that requires additional functionality on the network elements. Such additional functionalities increase the overhead on the network elements.

The deployment of server in WLAN and in TSP's network copes with the WAN issues, such as high latency, jitter, and packet losses that degrade the application execution performance in MCC environment. The use of high bandwidth links reduces the transmission delay that also assists in optimizing the application performance in MCC. The on-demand allocation of bandwidth makes the execution framework scalable for varying demand loads of applications.

2.3 Open Challenges: Application Performance Enhancement

In this Section, we highlight some of the most important challenges in optimizing the application execution performance in MCC. The discussion on the research challenges provides research directions to the domain researchers for further investigations and improvements in MCC. The discussion also leads us to our research problem.

User-transparent Cloud Discovery: A mobile device must discover the cloud to access and utilize the services provided by the cloud service provider. Although the cloud service discovery is discussed in (Goscinski & Brock, 2010), no one has investigated the problem of the delay involved in discovering and selecting the cloud services. Existing frameworks also do not fully address the problem of cloud discovery and service selection. The cloud discovery process is required to be crisp and user-transparent for optimizing the application execution performance in terms of delay. The delay involved in discovering the cloud server can be reduced by pro-actively finding the server. However, the proactive server discovery consumes processor cycles and battery power of mobile device and network resources if the service is not used later. The service discovery solutions, such as the ones reported in (Di Modica, Tomarchio, & Vita, 2011; Bertolli et al., 2010) for the peer-to-peer networks can be used in effective design and implementation of user-transparent cloud discovery in MCC.

Unobtrusive Application Offloading: The unobtrusive offloading of application refers to the automated lightweight offloading process with the focus on reducing the offload-

ing time and user inference. The operational environment in MCC is highly dynamic in nature. The common causes are varying connection status, bandwidth, and attenuation distortion. Because of dynamic environmental conditions, an unobtrusive offloading of application is vital to sustain the usability of real-time applications and to provide the enhanced quality of experience (QoE) to the end user. However, the dynamic environment and run-time delay-inducing partitioning and offloading algorithms make it non-trivial to optimize the application execution performance. The unobtrusive offloading of applications is still an open research challenge due to the dynamic environment and complexities involved in offloading decision making.

Optimal Live VM Migration: Optimal live VM migration between cloud-based servers is vital to improve the performance of application execution in MCC, considering intermittent wireless network bandwidth and mobility limitations. The problem is more critical when the mobile device is leveraging the local available resources. When a mobile user moves to a place far from the offloaded application, the increase in distance induces the latency and degrades the user's QoE. Therefore, migrating the live VM along with the mobile service consumer without affecting the end-user's service becomes vital to alleviate QoE degradation. The optimal live VM migration can be achieved by employing automated low cost migration procedures with the objective of minimal downtime. However, the live VM migration in a seamless manner is still an open challenge due to intrinsic limitations of wireless technologies and huge overhead of VM migration that cause the significant delay and disruption in execution process. The optimal live VM migration challenges can be addressed by migrating only the live states of VM that reduces the size of data exchange at run-time.

The research efforts already done by researchers in similar domain (Takahashi, Sasada, & Hirofuchi, 2012) can be guidelines for optimizing the Live VM migration for frameworks in MCC. After successful migration and deployment of VM, it is required to ensure the user-transparent access of migrated VM via initial IP address in the case of VM mi-

gration to new physical machine. Further efforts similar to (Raad et al., 2013) are required to realize the vision of seamless access to migrated VM in MCC.

Seamless Computational Resources Handoff: Seamless computational resources handoff is an important concern to ensure smooth migration among varied cloud services when mobile service consumer is on the move. The unmanaged mobility in wireless environment causes communication disruption when mobile device moves across two different communication coverage areas (Zekri, Jouaber, & Zeghlache, 2012). In MCC, if a mobile device moves farther from associated cloud server, then the application performance is surely degraded due to the increase in the communication latency. The problem is more critical when the services are provided by local cloudlets; since the local service provider do not allow the mobile user to access the local resources from outside of the network because of security reasons. This situation results in large number of computation resources handoffs.

The seamless computational resources handoff mechanisms aim to discover the available resources and migrate the application across two clouds/cloudlets in a seamless manner. The seamless computational resources handoff requires user-transparent resources discovery, automated handoff process, and non-perceivable disruption during the execution.

However, the realization of the seamless computational resources handoff is a challenging task because of inherited limitations of wireless technologies and limited resources of the mobile device. The seamless computational resources handoff is an essential research problem that needs effective solutions for improving the application execution performance. The research efforts in emerging seamless handoff and mobility management solutions in wireless networks, such as one reported in (Mitra, 2010) can be used in designing effective seamless mechanisms for computational resources handoff in MCC environment. In this thesis, we address this research challenge.

Lightweight Fault Tolerance: Fault Tolerance is a critical aspect which enables the framework to continue its operation rather than failing completely, when cloud server is no more accessible. For meeting availability requirements and considering offloading dependency on cloud servers and wireless networks; it is vital to deal with failures and to provide a reliable service. Fault tolerance is more critical aspect in MCC than traditional cloud. Failure can occur due to user mobility as mobile device enters and leaves a network. Hardware failures, running out of battery power, and network signal loss are other common reasons. Hence, to keep the failure hidden from end user, fault tolerance mechanism is required to be lightweight and user-transparent, and implement the graceful degradation if access to remote server is no more possible.

Although fault tolerance is addressed by (X. Zhang et al., 2010; Kosta et al., 2012; Verbelen et al., 2012a; Chun & Maniatis, 2009; Verbelen et al., 2011; Gordon et al., 2012; Ha et al., 2013), the solution is required to be user-transparent, lightweight and adaptive. Therefore, designing and development of fast, user-transparent, and adaptive fault tolerance mechanism is still a research challenge due to low potential of mobile devices and intrinsic limitations of wireless medium. The lightweight fault tolerance mechanisms are proposed by different researchers in similar domains (Martins, Narasimhan, Lopes, & Silva, 2010; Ahn, 2009), which can provide a guideline for researchers in MCC to design lightweight fault tolerance mechanisms. In the proposed thesis, we address the issue of network faults.

Agile Security and Privacy Mechanisms: Security and privacy are important concerns that impede successful deployment of clouds across the Internet (Khan et al., 2012). A number of frameworks emphasis the need of security and privacy but very few of them actually implement the security solutions. To improve the application execution performance in MCC, security and privacy mechanisms are required to be agile to mitigate the intolerable delay caused by security provisioning (Satyanarayanan et al., 2009). The agile security and privacy mechanisms can be implemented by employing quick and cost-

effective trust establishment, by designing efficient security algorithms, and low-overhead encryption mechanisms.

Several solutions exist to provide security, such as hardware-based secure execution and steganography (J. Liu, Kumar, & Lu, 2010). The limitations of these existing security mechanisms, such as large encryption key size and dramatic increase in amount of data impede the practical realization of security solutions for improving the application execution performance in MCC. The research efforts, such as the one reported in (Al-Muhtadi, Mickunas, & Campbell, 2002) can be used to effectively design the agile security mechanisms for MCC.

2.4 Conclusion

We reviewed the state-of-the-art application execution frameworks employed in the MCC environment to execute the applications. We presented a thematic taxonomy of various approaches employed by the application execution frameworks to improve the application execution time in the cloud. The comparative summary of the application execution frameworks based on the thematic taxonomy was also presented by highlighting the similarities and differences on the basis of significant parameters. We highlighted the open challenges in optimizing the application execution time in MCC environment, which included the two problems of seamless network fault tolerance and seamless handover of the applications. In the subsequent chapters, we build upon the highlighted problems and further investigate the depth of the network disconnection problem and its impact on the application performance in terms of execution time.

The state-of-the-art execution frameworks are designed for four different cloud models: cloud, cloudlet, mobile ad hoc cloud, and hybrid. Several frameworks focus on migrating the application to the cloud for execution, some leverages the locally available resources of cloudlet, and the rest use the hybrid platform of the cloud and cloudlet. The frameworks that rely on the remote cloud to augment resources enjoy a diverse range of available services in the cloud, on-demand availability of resources, low computation

latency, and high computational power. The cloud-based frameworks have to exchange the data between the two ends using the wireless Internet resources. The communication across the Internet is affected due to the high WAN latency, non-guaranteed bandwidth reservation, interoperability among the underlying different networking technologies, excessive delays, and bursty losses.

The cloudlet-based frameworks mitigate the issues of accessing the remote cloud, such as packet losses, jitter, link failure, and WAN latency, by migrating the mobile device application execution to locally available devices using the WLAN technology. The cloudlet-based frameworks do not require the Internet connectivity to access and utilize the cloudlet resources. However, the cloudlet possesses less resources, provide limited services and suffers from scalability issues. Similar to cloudlet-based frameworks, mobile ad hoc cloud frameworks also do not require the Internet connectivity to access and utilize the mobile ad hoc cloud resources. However, the devices in mobile ad hoc cloudlet share the resources with each other instead of accessing the centralized server.

The hybrid frameworks overcome the limitations of above mentioned both type of MCC model. The hybrid platform provides the flexibility to a mobile user to run time-critical components of an application on the nearby low-latency cloudlet and delay-tolerant components of an application on the remote cloud servers. Therefore, the hybrid frameworks manage the high WAN latency for time-critical components of an application and resolve the scalability issue of cloudlets by migrating the delay-tolerant components to the cloud server.

CHAPTER 3 : PROBLEM ANALYSIS OF CLOUD-BASED MOBILE APPLICATION EXECUTION IN DISRUPTIVE NETWORKS

The literature review in the previous chapter highlighted multiple open research issues. In this thesis, we focus on the issue of the impact of disruptive networks on application execution performance, which is part of fault tolerance of cloud based mobile application execution frameworks. The purpose of this chapter is to establish the problem highlighted in Chapter 1. In this chapter, we perform in depth investigation of the problem and its severity. We conduct a set of experiments to show that under normal mobility profile, the mobile user gets disconnected from the network frequently. Consequently, tackling the problem of network disconnections is important with reference to cloud-based mobile application execution performance. Subsequently, we conduct formal analysis of application execution and show that in most of the cases, the problem is non-trivial.

The rest of the chapter is organized into two main parts. In the first part (Section 3.1), we present the findings of our empirical study to show the significant impact of network disconnections on application performance in terms of execution time. This part consists of four subsections. Section 3.1.1 presents experimental setup and data collection. Section 3.1.2 discusses the mobility impact on device connectivity and response time in 3G network. Section 3.1.3 highlights the types of network connectivity profiles that a mobile user can experience during cloud based mobile application execution. Data analysis is presented in Section 3.1.4 and Section 3.1.5. Conclusions based on the empirical data analysis are summarized in Section 3.1.7. In the second part, we formally define application execution and include the impact of mobility induced network disconnections on application execution (Section 3.2). We show that for majority of connectivity profiles, application execution time is severely affected by network disconnections (Section 3.2.2). We also consider the two most relevant application execution frameworks and show that both frameworks fail to adequately handle the temporary as well as permanent network

disconnections (Section 3.2.4). Section 3.3 summarizes the findings of the analysis conducted in this chapter.

3.1 Empirical Study: Network Disconnection and Application Execution

In this section, we present the empirical study conducted for problem establishment. We discuss the measurement of the network connectivity pattern by using mobile device inside a metro transit train. Different profile types are discussed in this section. Using the collected profile, we replicate the scenario of mobile user accessing the cloud during metro transit train in the lab and study the impact of network disconnections on cloud based mobile application execution. Finally, we summarize the findings of empirical study.

3.1.1 Experimental Setup and Data Collection

We conducted a series of experiments in the lab environment, that replicated the real user connectivity scenario while the mobile users move in general. In the experimental process, the first step was to replicate the real user mobility pattern while a cloud based mobile application is being executed. For this purpose, we collected the real data traces of mobile user connectivity in 3G network for intra-city metro train transit. Figure 3.1 shows the mobile user connectivity data trace collection path. An android application "Network Monitor" that was installed on Samsung Galaxy S-II, was used to collect the device connectivity parameters - connectivity status and signal strength. The collected network connectivity data traces were used to conduct the experiments of cloud-based mobile application execution through emulated 3G networks.

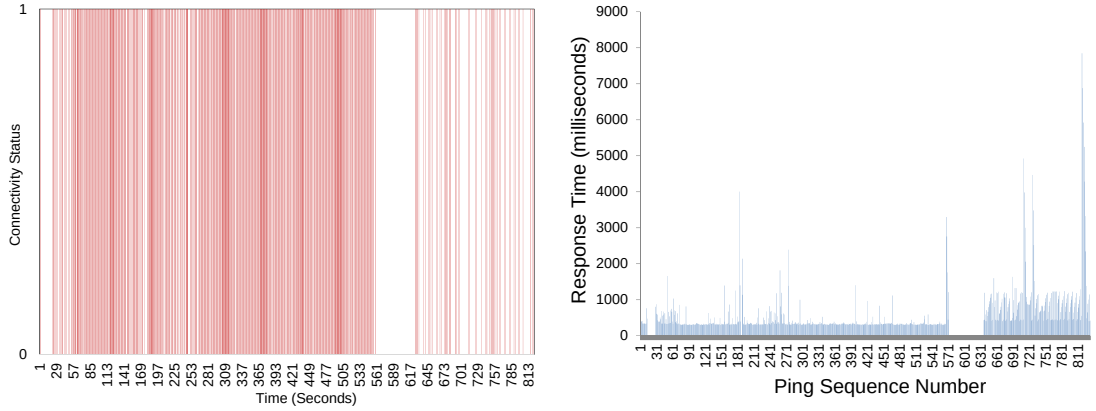
For emulation of the real implementation, we emulated the 3G connectivity traces in the lab environment over the LAN network. The disconnection has been emulated by dropping the packets between the computers during disconnection interval. The average connectivity and disconnection intervals were computed using the collected data traces and exponential distribution was used to generate a total of 30 data traces.

Figure 3.1: Data Collection Intra-city Metro Train Path Map (Courtesy: MYrapid http://www.myrapid.com.my)

A client server application provides communication between the emulated mobile device and the cloud server. Processing speed in terms of million instructions per second on the emulated mobile device has been matched with Samsung Galaxy II by generating background processes on the computer. On the cloud side, Openstack-based cloud is deployed and a 4 core virtual machine with 2.4 GHz processor is created for the mobile device. The virtual machine provides a processing speedup factor of 2.66 for the emulated mobile device.

The tested application is a simple algebraic computation repeated for the given input through automated interaction. Number of instructions of the application for use in the model has been computed using Lackey tool for valgrind (Developers, 2015). Number of other processes present in the system were kept uniform on the mobile and cloud side and were limited to automated background linux processes that remained in sleep state for entire experiment. Operating system overhead has been set to 7% of the total execution time and has been estimated using top command.

Using the above defined setup, we executed the mobile application by transferring it to the cloud server using communication module. The timing information was logged



(a) Intra-city Metro Transit Mobile Connectivity Graph (b) Ping Response Time in Intra-city Metro Transit

Figure 3.2: Mobility Effect on 3G Network Connectivity and Application Response Time

into text files during execution of the application by the cloud server as well as the mobile client. The text files were processed to generate the data that has been used for the analysis in the following sections. We start by showing the impact of mobility on device connectivity with the cloud while using a 3G cellular network.

3.1.2 Mobility Impact on Device Connectivity and Response Time in 3G Network

The graph in Figure 3.2 (a) shows the 3G device connectivity along the route where colored (red) space shows the device is connected and white space shows the device is disconnected. The study reveals that mobile device remains intermittently disconnected for 33% of the total sample time. We have also considered the effect of user mobility on response time of an application in 3G wireless network. Ping utility that is a basic tool to measure the response time on the Internet was used to collect the response time data. The graph in Figure 3.2 (b) shows the number of hikes in response time along the route. These hikes in the response time are due to extra delay added by retransmissions on 3G networks. More than 33% of the pings got either no response or their response time is greater than 400 ms. The response time greater than 400 ms is unacceptable for delay sensitive applications (Satyanarayanan et al., 2009).

These results suggest that there are significant disconnections and those disconnections can interrupt the cloud based mobile application execution. We now look into the

possible types of network connectivity profiles. Subsequently, we study the impact of each profile on application execution.

3.1.3 Types of Network Connectivity Profiles

With reference to cloud based mobile application execution, the network connectivity profiles can be one of the three types: (a) permanent disconnection, (b) intermittent disconnection and (c) always connected.

The permanent disconnection profile is the one where the mobile device never connects back with the cloud, once it gets disconnected from the cloud during the application execution. In such cases, the entire application needs to be re-executed, unless the state of the execution from the cloud is synchronized with the mobile device. On the other hand, intermittent disconnection profiles have short periods of disconnections during application execution; however, the device gets connected with the cloud. These profiles can be of two types depending on re-connection with the same cloud or another cloud service.

When the mobile device re-connects with the same cloud, there is a likely chance that the application state of the cloud based mobile application will be preserved on the cloud. The mobile device in this case is able to interact with partially executed application and can resume the application from the point of disconnection onwards. The adverse impact of disconnections in this case is relatively lesser in terms of application execution time. On the other hand, if the mobile device is unable to connect back with the same cloud/cloudlet service, the scenario of application execution changes. This is often the case when the mobile device user is on the move and is connected to different cloudlets on the way. In this case, the application execution on the previous cloudlet is lost unless a mechanism is devised for transfer of state to the new cloudlet. Multiple researchers have addressed this problem as discussed in the literature review. In the always connected profile, the mobile device never disconnects from the cloud server during the application execution.

The adverse impact of network disconnections on application execution time in case

of three profiles is different. The always connected scenario gives the least execution time as there is no disconnection during the application execution on the cloud server. The next less affected scenario is intermittent disconnections with connection to same cloud where the application execution time is generally increased by the time of disconnection. The most affected case is permanent disconnection where the entire execution state on the cloud is lost. Re-connection with a different cloud/cloudlet has intermediate affect on application execution time and depends on the efficient state transfer by the application execution framework. In the following sections, we present the experimental results for the three profiles.

3.1.4 Execution Performance in Different Connection Profiles

In this subsection, we analyze the performance of application in different connection profiles: (a) permanent disconnection, (b) intermittent disconnection, and (c) always connected.

3.1.4.1 Execution Time

We study the mobility effect on application execution time in MCC by executing a cloud-based interactive mobile application in an emulated MCC lab environment. In this study, the mobile device accesses the cloud server using emulated 3G wireless link over an Ethernet. Table 3.1 shows that the execution time when mobile device remains constantly connected is 995.17 $\pm$ 1.66 second with 99% confidence interval for the sample space of 30 values. This value is 69.53% and 31.91% less than the execution time of permanent and intermittent disconnection scenarios respectively.

Table 3.1: Execution Time Comparison in Different Connection Profiles

	Permanent Disconnection (second)	**Intermittent Disconnection (second)**	**Always Connected (second)**
Mean	3266.74	1461.74	995.17
Standard Deviation	103.34	119.65	3.52
Confidence Interval	3266.74 $\pm$ 48.68	1461.74 $\pm$ 56.36	995.17 $\pm$ 1.66

Figure 3.3 shows the execution time in case of permanent disconnection, intermittent disconnection, and always connected profiles for thirty different traces. The graph

presents the inconsistent trends for the execution time in case of permanent and intermittent disconnection. The reason for inconsistency in execution time of the permanent disconnection is the different time of disconnection occurrence. If the disconnection occurs earlier, the execution time is lesser because of less repetitive execution on the mobile device. In late occurrence of disconnection, the more instructions get wasted on the cloud server that results in higher number of repetitive execution on the mobile device. Hence, the execution time is higher in such cases. On the other hand, the inconsistency in case of intermittent disconnection is because of different sizes of disconnection intervals.

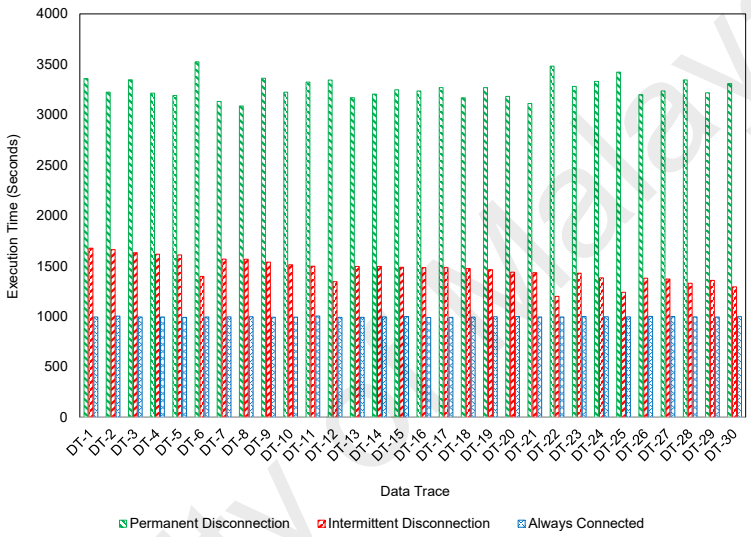


Figure 3.3: Execution Time in Different Connection Profiles

The higher execution time difference in case of permanent disconnection is because of the re-execution of whole application execution from the initial stage on the mobile device. That can be significantly reduced if the process execution on the remote cloud synchronizes with the mobile device during the execution to avoid the re-execution of the whole application on the mobile device. In case of intermittent disconnection, the execution time can be reduced by synchronizing the process states with the mobile device during the execution. These process states can be used to resume the application locally on the mobile device during the disconnection period. On the connection re-establishment, the mobile device states can be re-synchronized back with the cloud server.

3.1.4.2 Computation Wastage

Table 3.2 shows that computation wastage in case of permanent disconnection is 42.13 $\pm$ 4.60 percent of the total instructions, 2.25 * 10^{12}, with 99% confidence interval for the sample space of 30 values that is significantly higher than that of the intermittent and always connected scenarios. The computation wastage is zero in case of intermittent disconnection because the execution resumes on the cloud server after the connection re-establishment.

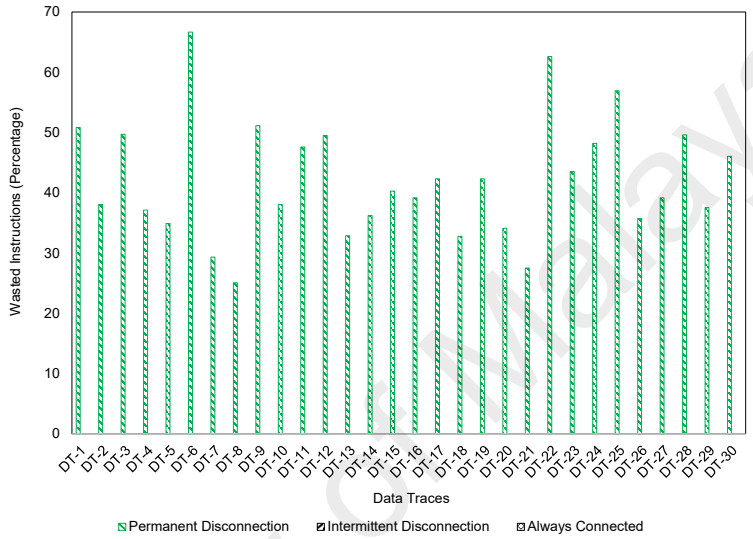


Figure 3.4: Computation Wasted For Different Connection Profiles

Figure 3.4 shows that the number of instructions wasted in case of permanent disconnection are presenting an inconsistent trend. The number of instructions wasted depends on the first event of disconnection. If the disconnection occurs earlier, only a small number of computed instructions are wasted because of limited computation by the cloud. In case of late disconnection, the number of instructions wasted is higher. The disconnection time in different data traces varies, hence the number of instructions wasted is also different.

Table 3.2: Computation Wastage Comparison in Different Connection Profiles

	Permanent Disconnection	**Intermittent Disconnection**	**Always Connected**
Mean (%age)	42.13	0	0
Std Deviation (%age)	9.77	0	0
Confidence Interval (%age)	42.13 $\pm$ 4.60	0	0

Table 3.3: Execution Time in Optimized VM-based Cloudlet, COMET, and Cloud Server Execution

	Optimized VM-based Cloudlet (second)	COMET (second)	Cloud Server Execution (second)
Sample Mean	1632.31	2080.93	995.17
Standard Deviation	295.32	172.26	3.52
Confidence Interval	1632.31 ± 139.11	2080.93 ± 81.14	995.17 ± 1.66

The results show that the interactive cloud based mobile applications are significantly affected by network disconnections. The execution time of the applications increases and the valuable computations are wasted, resulting in wastage of cloud resources that were used for the computation. We can safely say that network disconnection problem is a non-trivial challenge and needs to be addressed. We now consider the existing application execution frameworks and see how these frameworks manage the network disconnections.

3.1.5 State-of-the-art Frameworks Analysis

Two of the existing application execution frameworks deal with network disconnection scenarios. These frameworks are COMET and optimized VM-based cloudlet. COMET synchronizes the application state at thread level between the cloud and the mobile device. Upon network disconnection, the application can be executed by the mobile device from the last synchronized state. However, upon re-connection, the state cannot be transfered back to the cloud for remaining execution. The optimized VM-based cloudlet can generate VM overlay on demand. This on demand feature can be coupled with disconnection prediction to produce automated mechanism of application state transfer to the mobile device upon disconnection. However, this method remains dependent upon disconnection prediction, which is mostly inaccurate. We consider the two frameworks for analyzing the performance of execution time and computation wastage under disconnection profiles.

3.1.5.1 Execution Time

Table 3.3 shows the comparison of execution time in optimized VM-based cloudlet, COMET and uninterrupted cloud server execution. We are considering the cloud server execution time as a base-line for comparing the execution time of these two frameworks.

The average execution time for cloud server execution is 995.17 ± 1.66 with 99% confidence interval for the sample space of 30 values. However, the execution time minimum difference and execution time maximum difference in case of optimized VM-based cloudlet as compared to cloud server execution are 26% and 56%, respectively. The execution time minimum difference and execution time maximum difference in case of COMET as compared to cloud server execution is 40% and 58%, respectively. We can safely say that network disconnection significantly affects the execution time in case of existing frameworks, hence the existing frameworks fail to address the problem.

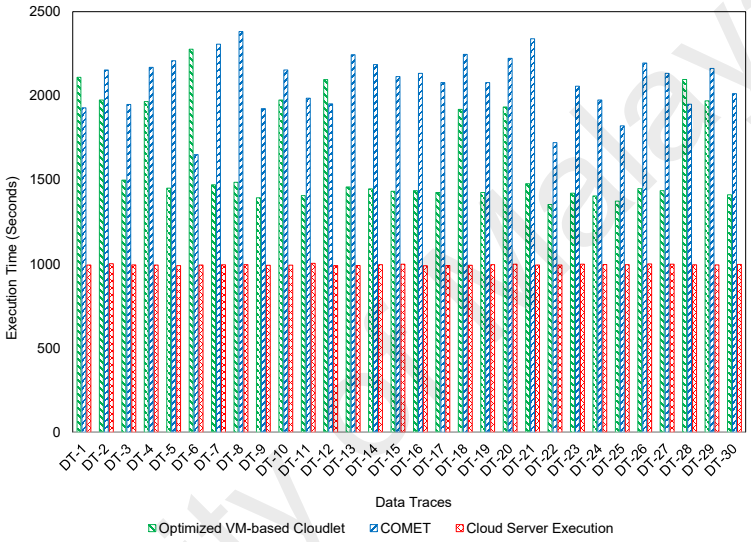


Figure 3.5: Execution Time in Optimized VM-based Cloudlet, COMET, and Cloud Server Execution

Figure 3.5 shows one another aspect of the study for different data traces where execution time in case of majority of the data traces is less for optimized VM-based cloudlet than that of the COMET. However, the execution time in case of data traces 1, 6, 12, and 28 is higher for optimized VM-based execution than that of COMET. In these data traces, mobile device disconnects from the network before getting back the VM-overlay in case of optimized VM-based cloudlet. Hence, the mobile device has to re-execute the whole application from initial stage that increases the application execution time of the optimized VM-based cloudlet.

Although COMET is able to synchronize the state of application, its execution time

remains high because of local execution of remaining task on the mobile device, which is significantly slower than the cloud.

3.1.5.2 Computation Wastage

Table 3.4 shows the comparison of the computation wastage in optimized VM-based cloudlet and COMET. The computation wastage is 0.76% of the total number of computed instructions, $2.25*10^{12}$ in case of COMET. This value is 95% less than that of the optimized VM-based cloudlet. COMET wastes significantly fewer instructions because of state synchronization. This highlights that the approach of state synchronization has merit, provided a light weight and more effective solution can be devised.

Table 3.4: Computation Wastage in Optimized VM-based Cloudlet and COMET

	Optimized VM-based Cloudlet	**COMET**
Sample Mean (%age)	14.4	0.76
Standard Deviation (%age)	2.29	0.40
Confidence Interval (%age)	14.4 ± 2.29	0.76 ± 0.40

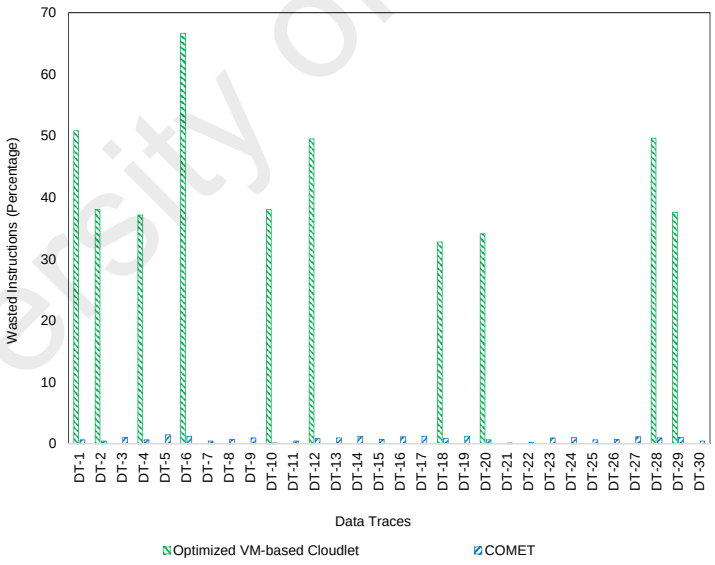


Figure 3.6: Computation Wastage in Optimized VM-based Cloudlet and COMET

Figure 3.6 presents the computation wastage comparison of optimized VM-based cloudlet and COMET-based execution for different connectivity data traces. The computation wastage for optimized VM-based cloudlet is significantly higher for some of the data traces where the mobile device is unable to get back the VM-overlay of cloudlet

executed process before the complete disconnection. Consequently, the computation performed on the previously visited cloudlet is wasted.

3.1.5.3 Energy Consumption

Table 3.5 shows the energy consumption in optimized VM-based cloudlet, COMET and mobile device execution. We have considered the energy consumption in mobile device execution as a base-line for comparison. The average energy consumption for mobile device execution is 72024 $\pm$ 2.67 with 99% confidence interval for the sample space of 30 values. However, the energy consumption for optimized VM-based cloudlet and COMET is 99.5% and 42% less than that of the mobile device execution, respectively. Although the optimized VM-based cloudlet consumes the very less amount of energy, the solution suffers a longer execution time and higher computation wastage.

Table 3.5: Energy Consumption in Optimized VM-based Cloudlet, COMET, and Mobile Device Execution

	Optimized VM-based Cloudlet (joule)	**COMET (joule)**	**Mobile Device Execution (joule)**
Sample Mean	235	41890	72024
Standard Deviation	112.25	7055.76	5.67
%age Standard Deviation	47.73	16.84	0.0078
Confidence Interval	235 $\pm$ 53	41891 $\pm$ 3323	72025 $\pm$ 3

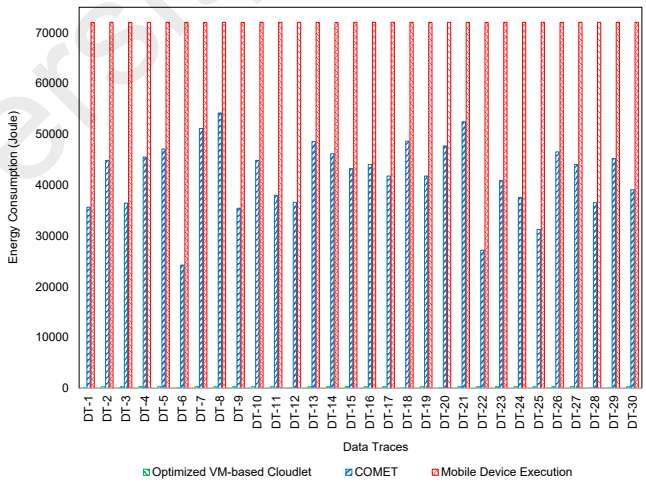


Figure 3.7: Energy Consumption in Optimized VM-based Cloudlet, COMET, and Mobile Device Execution

Figure 3.7 presents the energy consumption of optimized VM-based cloudlet, COMET and mobile device execution for various connectivity data traces. The energy consumption in case of COMET also varies for different data traces as COMET resumes the exe-

cution on the mobile device after the disconnection and complete it on the mobile device. The disconnection occurs earlier for some of the data traces and for the rest of the data traces the disconnection occurs relatively late.

Hence, the significant energy consumption in case of COMET arises the need to minimize the energy by resynchronizing back the process execution with the cloud.

3.1.6 Impact of Varying the Disconnection Parameters

In this section, we analyze the impact of varying the disconnection parameters on the application execution time and computation wastage.

3.1.6.1 Execution Time

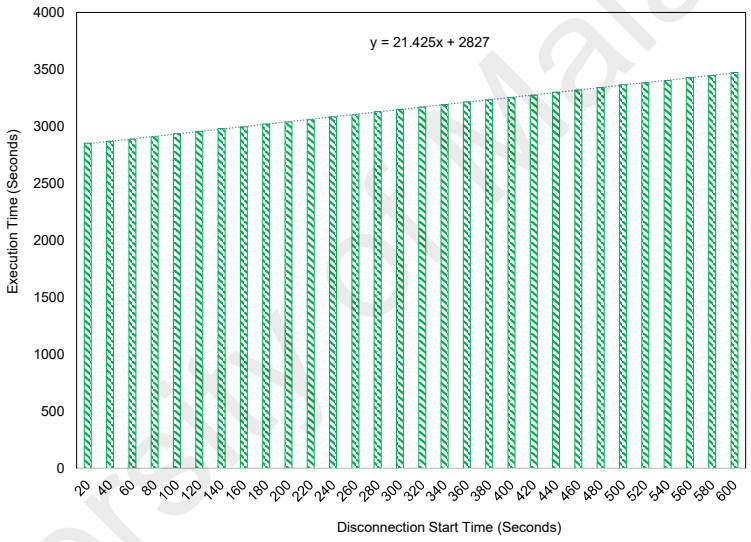


Figure 3.8: Impact of Disconnection Start Time on Cloud-based Application Execution Time

Figure 3.8 presents the impact of disconnection start time on the cloud-based mobile application execution time. The increase in execution time is due to wastage of large number of instructions wastage due to disconnection. The impact of the disconnection start time brings the inverse effects in case where the intermediate states or VM-overlay are successfully exchanged with the mobile device. The increase in disconnection start time reduces the execution time as the larger portion of the execution is performed on the cloud server.

3.1.6.2 Computation Wastage

Figure 3.9 shows the impact of the disconnection start time on number of instructions wasted on the cloud execution. The study is based on optimized VM-overlay cloudlet in the scenario where the mobile device is unable to get back the VM-overlay from the cloudlet before the complete disconnection. The graph shows that with the increasing disconnection start time, the number of computed instructions wastage on the cloud/cloudlet increases.

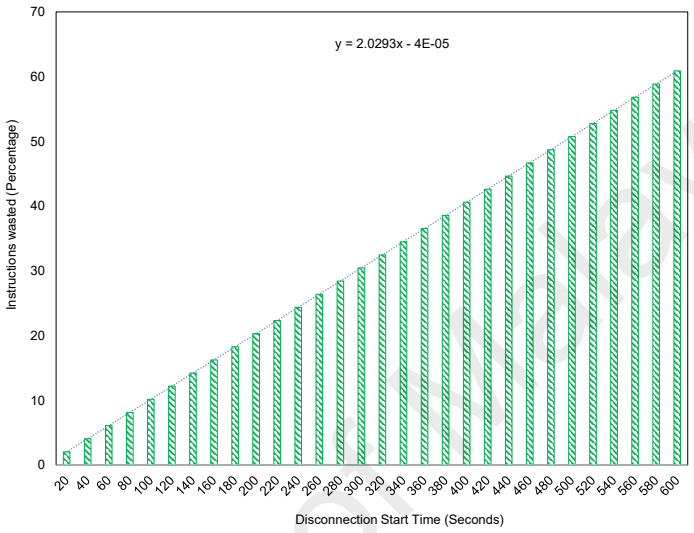


Figure 3.9: Impact of Disconnection Start Time on Computation Wasted

3.1.6.3 Cloud-based Application Execution Gain

Table 3.6 presents the impact of disconnection time on cloud-based application execution gain. The columns represent the expected disconnection time, execution time difference between the mobile device execution and cloud execution, and execution gain ratio.

Table 3.6: Impact of Disconnection Time on Cloud-based Application Execution Gain

Expected Disconnection Time (second)	Execution Time Difference (second)	Execution Gain (percentage)
10	1628	0.6195
20	1621	0.6168
30	1608	0.6119
40	1597	0.6077
50	1589	0.6047
60	1578	0.6005
70	1570	0.5974
80	1559	0.5933
90	1548	0.5888
100	1539	0.5856

Figure 3.10 shows the impact of expected disconnection interval size on the execution time gain ratio. The execution time gain ratio decreases with the increasing size of expected disconnection interval.

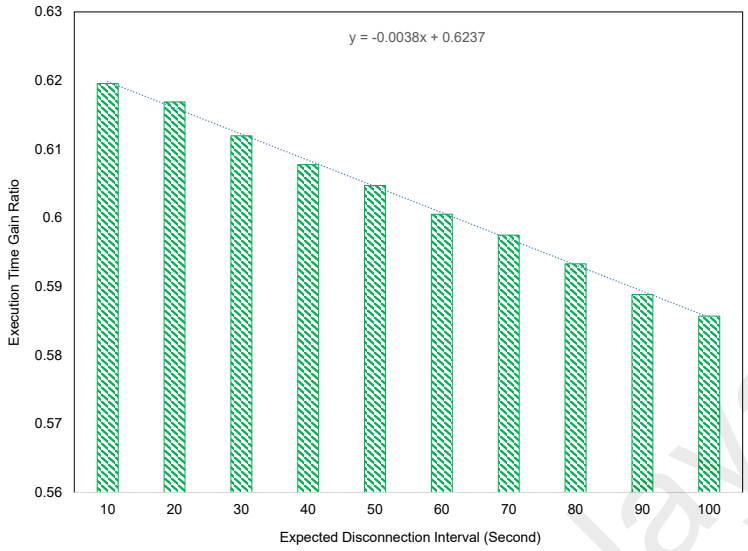


Figure 3.10: Impact of Expected Disconnection Interval Size on Cloud-based Application Execution Gain

3.1.7 Discussion

The empirical results highlight the following facts:

- Network disconnections are frequent during cloud based mobile application execution.
- The disconnections result in wasted computations and extended execution time. The increase in time is dependent on the connectivity profile and the pattern of disconnection.
- The existing frameworks do not completely manages the execution disruption caused by the network disconnections therefore, the execution time remains significantly high in case of network disconnections.
- The approach of state synchronization used by COMET has merit; however, COMET has significantly higher synchronization overhead. In addition, it cannot resume the computation on the cloud, in case of reconnection after intermittent disconnection.

This section clearly establishes the problem of adverse impact of network disconnections on cloud based mobile application execution as non-trivial. In the subsequent section, we formally analyze the problem to better understand the factors affecting the application execution in the event of disconnection.

3.2 Formal Analysis: Network Disconnection and Application Execution

Previous section showed that under usual mobility pattern of a mobile user, significant network disconnections can occur. The section also showed that the application execution performance in terms of execution time is severely affected in most of the connectivity profiles. In this section, we formally model the application execution time and incorporate the factor of network connectivity into execution time. The formal modeling has been used to facilitate in proofs of the impact of system parameters on performance metrics using formal theorems. We formally define the impact of disconnections under different connectivity profiles on application execution time. Finally, we show that the existing application execution frameworks do not handle network disconnections and majority of computation performed in the cloud is wasted in the event of network disconnection. The formal analysis has enabled us to obtain the insights by identifying the factors affecting the application execution because of disconnections.

3.2.1 Formal Definitions

We start this section by presenting the definitions of factors affecting the execution time under disruptive network connectivity such as instruction execution time and transfer time. Subsequently, we use these definitions to identify the conditions for execution time under disruptive network connectivity that affects the user experience. We use high order logic that is based on the simple typed theory of Church (Church, 1940). Table 3.7 presents the logical symbols and functions used in the paper and summarizes the interpretation.

Definition 1. (Execution Time) Given the processing speed of a mobile device as $P(m)$

Table 3.7: HOL Symbols used

$\wedge$	Logical conjunction, statement $A \wedge B$ is true only if both A and B are true
$\vee$	Logical disjunction, statement $A \vee B$ is true if at least A or B is true
$x_{(\mathbb{Z})}$	Variable x belongs to set Z or type of x is Z (set of integers)
$\forall$ x.	For all possible values of x
$\exists\, x.$	There exists at least one value of x
$\lambda\ x.\ f(x)$	Anonymous function of x with function definition given by expression $f(x)$
$(\lambda\ x.\ f(x))\ c$	Replace free occurrences of x in expression $f(x)$, resulting in $f[x := c]$
$sum(0,k)\ (\lambda\ x.\ f(x))$	$\sum_{x=0}^{k} f(x)$
$min(x,y)$	$if\ x \leq y\ then\ x\ else\ y$
$P\ x$	Function P applied on term x

and number of instructions of an application 'p' as $I(p)$, the execution time is given as $\frac{I(p)}{P(m)}$, ignoring all overheads and assuming the real-time automated user interaction.

The local execution time *Time_EL* can be modeled using the following equation:

$$Time\_EL = \frac{I(p)}{P(m)} \tag{3.1}$$

The remote execution time *Time_ER* can be modeled by dividing the local execution time with the speedup factor 'F' of the cloud server and adding the round trip delay for user interactions.

$$Time\_ER = \frac{I(p)}{P(m)} \times \frac{1}{F} + \frac{I(p)}{I(i)} \times 2d \tag{3.2}$$

The term $\frac{I(p)}{I(i)}$ represents the expected number of interactions during the execution of an application 'p'. The factor 2d represents the round trip delay taken by the single interaction.

The execution time can formally be defined using λ-abstraction as:

$$\vdash\ Time\_ER_{(\mathbb{R}_{\geq 0})} \stackrel{\text{def}}{=} (\lambda\ q_{(\mathbb{Z}_{\geq 0})}\ r_{(\mathbb{Z}_{>0})}.\ \frac{q}{r} \times \frac{1}{F}) + (\lambda\ a_{(\mathbb{Z}_{\geq 0})}\ b_{(\mathbb{Z}_{>0})}\ c_{(\mathbb{Z}_{\geq 0})}.\ (a/b)(\lambda sz.s(s(z)))c) \tag{3.3}$$

Where $(\lambda\ q\ r.\ E)$ indicates anonymous function of two variables q and r with the function defined by the expression E. The expression in this case is $\frac{q}{r}$. The variables q and r belong to the set of non-negative integers and positive integers respectively. The output of the function instructions execution time belongs to the set of positive real num-

bers. The variables q and r can be replaced by the number of instructions executed to perform the specific task $I(p)$ and the processor speed in instructions per seconds (IPS) $P(m)$ respectively, to give the execution time on the specific processor. $(\lambda\ a\ b\ c.\ E)$ indicates anonymous function of three variables a, b, and c with the function defined by the expression E. The expression in this case is $\frac{a}{b}(\lambda sz.s(s(z)))c$. The variables a, b and c belong to the set of non-negative integers, positive integers, and non-negative integers, respectively. The variables a, b, and c can be replaced by the total number of instructions $I(p)$, number of instructions after which the user interaction is required $I(i)$, and propagation delay d.

Definition 2. (Data Transfer Time)

The transfer time comprises of time required to transmit the data (known as transmission delay) and the network propagation delay. Transmission delay is the time taken to transmit the given amount of data using the available bandwidth. Transmission delay is a function of amount of data to be transferred and the data-rate of wireless/ wired link. Propagation delay is the time taken by the data to travel from the mobile device to the cloud server.

Data transfer time $Time\_DT$ can be mathematically written in terms of transmission delay $Time\_TD$ and propagation delay $Time\_PD$ as:

$$Time\_DT = Time\_TD + Time\_PD \tag{3.4}$$

For $|p|$ amount of the data, data-rate Rr, and propagation delay of the path d, the transfer delay $Time\_TD$ can be mathematically expressed as:

$$Time\_DT = \frac{|p|}{Rr} + d \tag{3.5}$$

Table 3.8: System variables

$\|p\|$	Size of program p	$\mathbb{Z}_{>0}$
$\|s\|$	Size of process states	$\mathbb{Z}_{>0}$
R_r	Data rate of the bottlenecked link along the path	$\mathbb{Z}_{>0}$
d	Network propagation delay from mobile device to cloud	$\mathbb{R}_{\geq 0}$
$I(p)$	Number of instructions executed of a program p	$\mathbb{Z}_{\geq 0}$
$I(p_m)$	Number of instructions executed on mobile device	$\mathbb{Z}_{\geq 0}$
$I(p_{c1})$	Number of instructions executed on cloudlet c1	$\mathbb{Z}_{\geq 0}$
$I(p_{c2})$	Number of instructions executed on cloudlet c2	$\mathbb{Z}_{\geq 0}$
T_{dc}	Disconnection time	$\mathbb{Z}_{>0}$
F	Speed up factor of cloud server with respect to mobile device	$\mathbb{Z}_{\geq 0}$
$0 :\iff \lambda$ sz.z	Natural number zero	
$2 :\iff \lambda$ sz.s(s(z))	Natural number two	

The data transfer time can formally be defined using λ-abstraction as:

$$\vdash \mathit{Time\_DT}_{(\mathbb{R}_{\geq 0})} \stackrel{\text{def}}{=} (\lambda\ e_{(\mathbb{Z}_{\geq 0})}\ f_{(\mathbb{Z}_{>0})}.\ \frac{e}{f}) + d \tag{3.6}$$

where $(\lambda\ e\ f\ E)$ indicates anonymous function of two variables e and f with the function defined by the expression E. The expression in this case is $\frac{e}{f}$. The variable e belongs to the set of non-negative integers. However, the variables f belongs to a set of positive integers. The output of the function (transmission time) belongs to the set of positive real numbers. The variables e and f can be replaced by the amount of the data to be transmitted and the links transmission rates, respectively.

3.2.2 Connectivity Profiles and Execution Time

A number of research efforts in the form of cloud-based mobile application execution frameworks have been made to enable the execution in MCC. In case of network connection disruption, the frameworks start execution either on the same computing device or has to move to other device. Considering the selection of execution location after network connection disruption by the frameworks, we define three main categories.

- Same Cloud
- Cloudlet-to-cloudlet
- Cloudlet-to-mobile Device

There are two main cases of network connection disruption: a) permanent disconnection and b) intermittent disconnection. In “permanent disconnection" scenario, the mobile device may not be able to reconnect back with the cloud due to failure of permanent network connectivity. In “intermittent disconnection" scenario, the mobile device may be able to reconnect back with the cloud. This category has two further types: (a) reconnection with the same cloud and (b) reconnection with different cloud. The three disconnection scenarios can be modelled for the three types of cloud, cloudlet and mobile based reconnections.

3.2.2.1 Same Cloud

This scenario occurs when the application can only be executed in the cloud. The total application execution time comprises of remote execution time *Time_ER*, propagation delay ‘d’, disruption time *Time_dis.*, interaction delay *Time_I*, and result transfer time *Time_Rtr*. Total application execution time can be mathematically given by Equation 3.7.

$$Time\_totE = d + Time\_ER + Time\_dis. + Time\_I + Time\_Rtr \tag{3.7}$$

where $Time\_totE$ represent total execution time. In this scenario, we are assuming that application is already installed in the cloud and only application start command is needed to be sent from mobile device to cloud server.

$$Time\_totE = 2 \times d + \frac{I(p)}{P(m)} \times \frac{1}{F} + \frac{I(p)}{I(i)} \times 2d + k \times T_{dc} \tag{3.8}$$

The terms F, $I(p)$, $P(m)$, I_i, k, and T_{dc} represent the cloud speedup factor, number of instructions executed, mobile device speed, expected number of instructions to be executed between consecutive interactions, number of disconnections, and expected disruption time, respectively. Total application execution time can formally be defined using

λ-abstraction as given in Expression 3.9.

$$
\begin{aligned}
Time\_totE_{(\mathbb{R}_{\geq 0})} \stackrel{\text{def}}{=} \forall d\ & I(p)\ P(m)\ I(i)\ T_{dc} \\
& \vdash \lambda d_{\mathbb{Z}_{>0}} sz.s(s(z))d + (\lambda q_{(\mathbb{Z}_{\geq 0})}\ l_{(\mathbb{Z}_{>0})}.\frac{q}{l})I(p)P(m)\frac{1}{F} \\
& + (\lambda\ a_{(\mathbb{Z}_{\geq 0})}\ b_{(\mathbb{Z}_{>0})}\ .\ (a/b)I(p)I(i))(\lambda c_{(\mathbb{Z}_{\geq 0})} sz.s(s(z))c)d + (\lambda\ y.ky)T_{dc}
\end{aligned}
\tag{3.9}
$$

3.2.2.2 Cloudlet-to-cloudlet

This scenario occurs when the mobile device moves to and connects with another cloudlet during the execution of the application. The application state of the application being executed in the previous cloudlet needs to be transfered to the new cloudlet using mobile device or an alternate mechanism. The execution is transferred from the previously visited cloudlet to the newly visited cloudlet through the mobile device. The mobile device takes the intermediate states from the previously visited cloudlet when it starts moving from the cloudlet. On successfully getting the intermediate states, the mobile device can migrate the previously computed process states along with the application to the newly visited cloudlet. The rest of the execution will be performed on the newly visited cloudlet after resuming the process. Assuming the cloudlet execution speed for the mobile device to be equal and ignoring the suspend resume time, the total execution time can be mathematically modelled as follows:

$$
\begin{aligned}
Time\_totE = {} & Time\_DTMC1 + Time\_DTMC2 + Time\_DTCM + Time\_Ec1 + Time\_Ec2 \\
& + Time\_dis. + Time\_I + Time\_Rtr
\end{aligned}
\tag{3.10}
$$

where $Time\_DTMC1$, $Time\_DTMC2$, $Time\_DTCM$, $Time\_Ec1$, $Time\_Ec2$, $Time\_dis.$, $Time\_I$ and $Time\_Rtr$ represent application transfer time from mobile device to first cloudlet server, application and states transfer time from mobile device to second cloudlet server, intermediate results transfer time from cloudlet to mobile device, application execution time on first cloudlet, application execution time on second cloudlet, the disrup-

tion time, user interaction time, and result transfer time, respectively. That can further be rewritten as follows:

$$Time\_totE = (\frac{|p|}{Rr} + d) + (\frac{(|p|+|s|)}{Rr} + d) + (\frac{(|s|)}{Rr} + d) + \frac{I(p_{c1})}{P(m)} \times \frac{1}{F_1} + \frac{I(p_{c2})}{P(m)} \times \frac{1}{F_2} + \frac{I(p)}{I(i)} \times 2d + k \times T_{dc} + d \tag{3.11}$$

The terms |p|, Rr, d, |s|, $I(p_{c1})$, $I(p_{c2})$, F_1, F_2, k, T_{dc} represent the application size, data rate, propagation delay, state size, number of instructions to be executed on previously visited cloudlet, number of instructions to be executed on newly visited cloudlet, speedup factor for previously visited cloudlet, speedup factor for newly visited cloudlet, number of disconnections, and expected disconnection interval respectively.

Total application execution time for this scenario can formally be defined using λ-abstraction as:

$$\begin{aligned} Time\_totE_{(\mathbb{R}_{\geq 0})} \stackrel{\text{def}}{=} & \forall |p|\ d\ |s|\ Rr\ I(p)\ F_1\ F_2\ k\ I(p_{c1})\ I(p_{c2})\ P(m)\ I(i)\ T_{dc} \\ & \vdash (\lambda\ e_{(\mathbb{Z}_{\geq 0})}\ f_{(\mathbb{Z}_{>0})}\ .\frac{e}{f})|p|Rr + (\lambda\ m_{(\mathbb{Z}_{\geq 0})}\ n_{(\mathbb{Z}_{>0})}\ .\frac{m}{n}\ (|p|+|s|)\ Rr) + \\ & (\lambda d_{(\mathbb{Z}_{>0})}\ sz.s(s(s(s(z))))d\ ) + (\lambda\ m_{(\mathbb{Z}_{\geq 0})}\ n_{(\mathbb{Z}_{>0})}\ .\frac{m}{n}|s|Rr) + \\ & (\lambda\ q_{(\mathbb{Z}_{\geq 0})}\ r_{(\mathbb{Z}_{>0})}.\frac{q}{r}I(p_{c1})P(m))\frac{1}{F_1} + \\ & (\lambda\ q_{(\mathbb{Z}_{\geq 0})}\ r_{(\mathbb{Z}_{>0})}.\frac{q}{r}I(p_{c2})P(m))\frac{1}{F_2} + \\ & (\lambda\ a_{(\mathbb{Z}_{\geq 0})}\ b_{(\mathbb{Z}_{>0})}\ c_{(\mathbb{Z}_{\geq 0})}.\ (a/b)(\lambda sz.s(s(z)))c)I(p)I(i)d + (\lambda\ y.ky)T_{dc} \end{aligned} \tag{3.12}$$

3.2.2.3 Cloud/Cloudlet-to-mobile

When the mobile device finds that connection to the cloudlet/cloud is unavailable, the mobile device starts the application execution locally if no other cloudlet is accessible.

Total execution time for this scenario can be mathematically modeled as:

$$
\begin{aligned}
Time\_totE &= Time\_ELAD + Time\_DTMC + Time\_DTCM + Time\_EC + Time\_I + Time\_dis. \\
&= \frac{I(p_m)}{P(m)} + (\frac{|p|}{Rr} + d) + (\frac{|s|}{Rr} + d) + \frac{I(p_c)}{P(m)} \times \frac{1}{F} + \frac{I(p_c)}{I(i)} \times 2d + T_{dc} \\
&= \frac{I(p_m)}{P(m)} + \frac{|p| + |s|}{Rr} + \frac{I(p_c)}{P(m)} \times \frac{1}{F} + 2 \times d + \frac{I(p_c)}{I(i)} \times 2d + T_{dc}
\end{aligned}
\tag{3.13}
$$

where $Time\_ELAD$, $Time\_DTMC$, $Time\_DTCM$, $Time\_EC$, and $Time\_I$ represent the local execution time after the disconnection, application transfer time from cloud to mobile device, state transfer time from mobile device to cloud, execution time on cloud, and interaction time, respectively. The total application execution time in case of application execution on mobile device after disruption in network connection can formally be defined using λ-abstraction as:

$$
\begin{aligned}
Time\_totE_{(\mathbb{R}_{\geq 0})} \stackrel{\text{def}}{=} & \forall |p|\ |s|\ d\ Rr\ I(p_m)\ I(p_c)\ I(i)\ P(m)\ I(i)\ T_{dc} \\
& \vdash (\lambda\ q_{(\mathbb{Z}_{\geq 0})}\ r_{(\mathbb{Z}_{>0})}.\frac{q}{r} I(p_m)P(s_m)) + \\
& (\lambda\ e_{(\mathbb{Z}_{\geq 0})}\ m_{(\mathbb{Z}_{\geq 0})}\ f_{(\mathbb{Z}_{>0})}\ .\frac{(e+m)}{f}|p||s|Rr) + \\
& \lambda x_{(\mathbb{Z}_{\geq 0})} y_{(\mathbb{Z}_{\geq 0})} xy(\lambda\ a_{(\mathbb{Z}_{\geq 0})}\ b_{(\mathbb{Z}_{>0})}.\frac{a}{b} I(p_c)P(s_m))(\frac{1}{F}) + \\
& (\lambda d_{(\mathbb{Z}_{>0})}\ sz.s(s(z))d\ ) + \\
& (\lambda\ a_{(\mathbb{Z}_{\geq 0})}\ b_{(\mathbb{Z}_{>0})}\ c_{(\mathbb{Z}_{\geq 0})}.\ (a/b)(\lambda sz.s(s(z)))c)I(p)I(i)d + T_{dc}
\end{aligned}
$$

3.2.3 Disruptive Remote Execution Comparison with Non-disruptive Execution

In this section, we present the comparison of application execution time with and without network disconnections. We consider two types of applications to highlight the significance of the problem. The cloud based mobile applications can be interactive, requiring user interaction during execution or the applications can be non-interactive. Although this thesis is about interactive applications, we present the formal analysis for both types to highlight why interactive applications are different and more complex compared to non-interactive applications.

3.2.3.1 Same Cloud

In case of same cloud, for interactive applications, it is trivial to show that the application execution time in disruptive networks is higher than that of the non-disruptive network. We give the formal theorem for the sake of completeness. Mathematically, we want to prove that remote execution time with disconnections is higher than the time without disconnections. This is represented by the Equation 3.14, which is the simplified form of the inequality.

$$\begin{aligned} Time\_totE_{DC} &> Time\_totE_{NoDC} \\ k \times T_{dc} &> 0 \end{aligned} \tag{3.14}$$

Formally, the above equation can be represented as follows.

Theorem 1.

$$\begin{aligned} &\vdash \forall\, k\, T_{dc}.\, k > 0 \wedge T_{dc} > 0 \\ &\implies (\lambda\ y.ky) T_{dc} > 0 \end{aligned}$$

Proof. This is trivial to prove because this expression is always true, given that $T_{dc} > 0$. Hence the theorem is a universal truth if the network is disruptive. □

In case of non-interactive applications, it is again trivial to show that the application execution time in disruptive networks with intermittent network connection is unaffected by the network disconnections. This is because the application execution continues even when the mobile device is disconnected from the network. This is mathematically represented in Equation 3.15.

$$(Time\_ERb + Time\_ERa + Time\_Rtr) = (Time\_ERb + Time\_ERa + Time\_Rtr) \tag{3.15}$$

Both sides of the equations are equal as non-interactive application execution is unaffected by the network disconnections.

3.2.3.2 Cloudlet-to-Cloudlet

In case of disconnection from one cloud/cloudlet and reconnection with another cloud/cloudlet, there are two scenarios. If the state of application is not migrated to the new cloudlet, the entire application needs re-execution. This is the worst case scenario with reference to execution time under disruptive networks. The execution time for disruptive networks is always higher than non-disruptive case. Therefore, we do not consider this scenario. The better scenario is where the state is transferred through mobile device to the new cloudlet. In this case, irrespective of interactive or non-interactive application, the execution time of application under disruptive network connection is higher as shown in equation below.

$$(\frac{|p|}{Rr} + d) + (\frac{(|p|+|s|)}{Rr} + d) + (\frac{(|s|)}{Rr} + d) + \frac{I(p_{c1})}{P(m)} \times \frac{1}{F_1} + \frac{I(p_{c2})}{P(m)} \times \frac{1}{F_2} + \frac{I(p)}{I(i)} \times 2d$$
$$+ T_{dc} + Time\_Rtr > (\frac{|p|}{Rr} + d) + \frac{I(p)}{P(m)} \times \frac{1}{F} + \frac{I(p)}{I(i)} \times 2d + Time\_Rtr \tag{3.16}$$

If we consider the case of equal speedup factor for both cloudlets then the simplified form of the equation will be as below:

$$(\frac{(|p|+|s|)}{Rr} + d) + (\frac{(|s|)}{Rr} + d) + T_{dc} > 0 \tag{3.17}$$

This is formally given in theorem below.

Theorem 2.

$$\vdash \forall\, |p|\ |s|\ Rr\ d\ T_{dc}\ |p| > 0 \wedge |s| > 0 \wedge Rr \geq 0\ d > 0; T_{dc} > 0$$
$$\implies (\lambda\, m_{(\mathbb{Z}_{\geq 0})}\, n_{(\mathbb{Z}_{>0})}\, . \frac{m}{n}\, (|p|+|s|)\, Rr) + (\lambda d_{(\mathbb{Z}_{>0})}\, sz.s(s(z))d\ )$$
$$+ (\lambda\, m_{(\mathbb{Z}_{\geq 0})}\, n_{(\mathbb{Z}_{>0})}\, . \frac{m}{n} |s| Rr) + T_{dc} > 0$$

Proof. This is trivial to prove because this expression is always true, given that $T_{dc} > 0$, $|s| > 0$, and $d > 0$. Hence the theorem is a universal truth if the network is disruptive. □

3.2.3.3 Cloud/Cloudlet-to-mobile

This scenario arises in case of permanent disconnection. We again ignore the worst case of no state received. In this case, the application is executed by the mobile device locally after disconnection. In this scenario, the interactive and non-interactive applications behave differently.

For non-interactive applications, the following theorem shows that the execution time in disruptive networks is higher. The theorem is a universal truth for all non-negative values of the state, even for arbitrarily large values of Rr.

Theorem 3.

$$\vdash \forall\, |s|\ Rr.\ |s| > 0 \wedge Rr > 0$$
$$\implies (\lambda\ m_{(\mathbb{Z}_{\geq 0})}\ f_{(\mathbb{Z}_{>0})}\ .\frac{m}{f}|s|Rr) > 0$$

The case of interactive applications is complicated and needs special conditions to hold for the non-disruptive case to give better execution time compared to the disruptive case. We observe that the conditions before permanent network disconnection are same for the case of disruptive as well as non-disruptive network execution of the cloud based interactive mobile application. However, in case of non-disruptive case, the computation continues on the cloud/cloudlet, which requires remote interaction with the user through mobile device. On the other hand, the local execution on the mobile device does not require remote interaction; however, the execution is slower in this case. In this case, a specific relationship between interaction time, number of interactions and the local execution speed of the mobile device is required to hold for the disruptive case to have higher time. We first derive the relationship mathematically. Subsequently we present the formal theorem for this relationship to hold.

Ignoring the state transfer time in case of disruptive network, the following expression must be satisfied for the remote execution under disruptive network to have remote execution gain.

$$\frac{I(p_m)}{P(m)} > \frac{I(p_m)}{P(m)} \times \frac{1}{F} + \frac{I(p_m)}{I(i)} \times 2d \tag{3.18}$$

The expression states that the time taken by the mobile device to compute the remaining instructions after disconnection must be higher than the aggregate time taken by interactions and cloud/cloudlet based execution of the remaining instructions after presumed disconnection. The expression can be simplified to the following form.

$$P(m) * F \times 2d < I(i)(F-1) \tag{3.19}$$

Using the minimum value of $F = 2$, this expression states that the instructions executed by the mobile device during two round trip propagation delays of the network should be less than the instructions after which interaction takes place. To ensure that the above expression holds, following formal theorem should hold, where the desired condition appears as a pre-condition. In general, the network interaction is slower than twice the round trip time, because the user response is one round trip time, hence the condition always holds. With pre-condition satisfied, the theorem remains a universal truth.

Theorem 4.

$$\vdash \forall\, I(i)\ F\ d\ P(m)\ .F > 2 \wedge P(m) < I(i)/4d$$

$$\implies (\lambda\ a_{(\mathbb{Z}_{>0})}\ .a(\lambda d_{(\mathbb{Z}_{>0})}\ sz.s(s(s(s(z))))d\ )P(m) <$$

$$(\lambda\ \ b_{(\mathbb{Z}_{>0})}\ c_{(\mathbb{Z}_{>0})} b(c-1)I(i)F)$$

3.2.4 Formal Analysis of Application Execution Frameworks in Disruptive Network Conditions

In this section, we analyze the remote application execution for application execution frameworks in disruptive network conditions of MCC. There are two frameworks that can be used for network disconnection cases. First framework is optimized VM based cloudlet, where the application along with the virtual machine is resumed on a different cloudlet, after disconnection. This framework suggests using mobile device to carry overlay VM to the next cloudlet. A number of variations of the idea has been proposed where VM is carried to the next cloudlet through intermediate cloud or using mobility prediction and direct transfer. All such cases can be considered as special case of the VM based cloudlet, as far as this analysis is concerned. This is because the mobile device can get the VM on next cloudlet only after it has connected with the next cloudlet. Therefore, the time taken for reconnection is at least equal to the time of the disconnection.

The second framework is COMET, where application is resumed on the mobile device after disconnection. The application continues to be executed on the mobile device and cannot be transfered to the cloud/cloudlet upon reconnection. We formally analyze both frameworks and compare them with one another, showing one to be superior compared to the other. Subsequently, we also show how this analysis sets the objective for our own proposed algorithms.

3.2.4.1 Optimized VM based Cloudlet

This mechanism has been proposed by Satyanarayanan et al.(Satyanarayanan et al., 2009). In case the mobile device predicts a network disconnection, the VM overlay is created and transfered to the mobile device. Upon reconnection with the next cloudlet, the state is used to create new virtual machine for the mobile device on the next cloudlet. For best case analysis of the execution time of the application, we make following set of assumptions.

- Irrespective of the mechanism for mobility prediction, we assume that the mobile

device is always able to predict the network disconnection well in time.

- The time for overlay creation and creation of VM for the mobile device to resume computation can be ignored.

- Irrespective of the mechanism for state transfer, the state is always available on the next cloudlet, when the mobile device connects with the next cloudlet.

Under these assumptions, we can analyze the best case application execution time of VM based cloudlet. Based on the above stated assumptions, irrespective of interactive or non-interactive application, the execution time of application under disruptive network connection is higher as shown in Equation 3.16. The simplified form of the equation for the case of equal speedup factor for both cloudlets is shown in Equation 3.17.

$$\mathscr{L} = \{|p| + |s| = 570 \times 10^6 bytes,\ |s| = 500 \times 10^6 bytes,\ Rr = 9.5\ Mbps,\ T_{dc} = 2\ seconds\}$$

The condition (L.H.S. = 903.15 > R.H.S. = 0) holds. This is trivial to prove because this expression is always true as all factors involved cannot be negative. Hence the theorem 2 is a universal truth if the network is disruptive for optimized VM-based cloudlet.

To summarize, the optimized VM based cloudlet increases the application execution time by at least the time of the network disconnection, in the best case scenario. In general, the situation is worst, specifically, if the network disconnection is not predicted in time. In such cases, the application execution on the cloudlet can be wasted and the mobile device needs to restart from the point of second to last disconnection or in the worst case, from the start of the application. It will be more practical, if the mobile device can perform useful computations during disconnection time. COMET works based on this idea as explained in next section.

3.2.4.2 COMET

COMET (Gordon et al., 2012) improves on optimized VM based cloudlet in three ways. First, instead of exchanging the overlay VM, COMET synchronizes thread level

state between cloudlet and the mobile device. This has significantly lower overhead, although the data is exchanged more frequently. Second, the state between cloud/cloudlet and the mobile device is continuously synchronized, eliminating the possibility of total state loss at the time of disconnection. Third, COMET resumes the application on the mobile device, after network disconnection. This results in performing useful computations during disconnection compared to wasting the time of disconnection. However, there is a negative side of this execution. Even when the mobile device is able to reconnect with same or a different cloud, the mobile device cannot take advantage of the cloudlet and must continue the computation locally.

The case of COMET is equivalent to the cloudlet-to-mobile class that has been discussed above. COMET execution of application can be considered as interactive because of continuous state synchronization. Ignoring the state transfer time in case of disruptive network, the Equation 3.19 must be satisfied for the remote execution under disruptive network to have higher execution time. The simplified form of the equation is as follows:

$$\frac{P(m) * F \times 2d}{(F-1)} < I(i) \tag{3.20}$$

Using the minimum value of $F = 2$, to ensure that the above expression holds, the Theorem 4 should hold, where the desired condition appears as a pre-condition. In general, the network interaction is slower than twice the round trip time, because the user response is one round trip time, hence the condition always holds. With pre-condition satisfied, the theorem remains a universal truth.

$$\mathscr{L} = \{P(m) = 856 \times 10^6,\ F = 2,\ d = 50 \times 10^{-3},\ I(i) = 3.42 \times 10^8,\ RTT = 100ms\}$$

The condition (L.H.S. = 1.7×10^8 < R.H.S. = 3.42×10^8) holds for the Theorem 4 with the constraint that the interaction interval must be greater than the RTT. It is worth noting that unlike general application execution where user interaction cannot be controlled,

COMET based cloud/cloudlet interaction with the mobile device can be controlled and intentionally set such that the above theorem's precondition is not satisfied. COMET synchronize the process states whenever the distributed shared memory is modified. That increases the overhead. Although COMET outperforms the optimized VM-based cloudlet execution, COMET is still an impractical solution in case of highly dynamic environment as COMET does not support the resumption of execution on the cloud after the connection re-establishment.

3.3 Conclusion

In this chapter, we showed that under very realistic user mobility scenarios, the mobile device frequently gets disconnected from the cloud/cloudlet during the execution of cloud based mobile applications. We also showed that such disconnections adversely impact the performance of the applications. We presented extensive empirical analysis to show that the existing application execution frameworks are unable to mitigate or reduce the adverse impact of network disconnections on mobile application execution.

The findings of the empirical analysis lead us to the formal analysis of the application execution time under disruptive networks. We derived the expressions for execution time and showed that under different network connectivity scenarios, the application execution time is always affected by the network disconnections for interactive applications. For non-interactive applications, the situation remains the same with exception of reconnection with the same cloud/cloudlet, in which case the performance is unaffected.

We formally analyzed the application execution frameworks and their handling of network disconnections. The formal analysis revealed that under best case, the VM based cloudlet experiences a delay equal to the aggregate time of network disconnections. On the other hand, the COMET may experience lesser or higher than this delay, depending upon the remaining computation at the time of disconnection. We note that continuous synchronization of COMET helps it match the best case of optimized VM based cloudlet; however, its inability to re-synchronize upon reconnection can lead to slower application

execution in case of network disconnection. This observation gives us hint about our proposed solution.

In the subsequent chapter, we propose process state synchronization algorithms that aim at minimizing the application execution delay from the best case of equivalent to the aggregate of network disconnections. We propose the execution on the mobile device during network disconnections. However, unlike COMET, our proposed solution aims at transferring the computation on the cloud/cloudlet upon reconnection.

The empirical and formal analysis presented in the previous chapter establishes the problem of the impact of network disconnections on the cloud based mobile application execution as non trivial. The purpose of this chapter is to present the algorithms proposed for addressing the problem. We propose a solution for significantly improving the cloud-based mobile application execution performance in disruptive networks. We propose the use of process state synchronization as a solution to the problem. The problem is mathematically modeled to compute the preferred and upper-bound synchronization intervals for process state synchronization, such that the loss of computation can be minimized for cloud based mobile application execution.

The chapter is organized into five sections. Section 4.1 presents the proposed algorithms for process state synchronization to support mobility in MCC. Section 4.2 elaborates the proposed process state synchronization algorithm with the help of a sequence diagrams illustrating the execution and synchronization of a simple application. Section 4.3 presents the mathematical model and drives upper bound on synchronization interval. The model will be used in next chapter for the validation of the emulated results of the undertaken research. Section 4.4 highlights the distinctive features of the proposed algorithm. Section 4.5 summarizes the chapter with conclusive remarks.

4.1 Process State Synchronization (PSS)

We propose the use of periodic process state synchronization as a solution to the problem of application state loss because of network disconnections during cloud based execution of mobile applications. This section presents the proposed process state synchronization (PSS) algorithm. Synchronization of execution state of the process contains three components. Firstly, the application execution environment (hardware architecture, operating system and libraries support) in the cloud/cloudlet and the application execution environment on the mobile devices should be compatible in such a way that the execution

state captured on one computational element can be used to resume the process on the other. This requirement is explained in Section 4.1.1. Second requirement is a mechanism to capture the state on the given environment and resume the process from the captured state on the environment of the second computational element. This component is explained in the Section 4.1.2. Finally, the mechanism of actually synchronizing the state across computational elements is required, which is explained in Section 4.1.3.

4.1.1 Cloud-based Application Execution

Three application execution models exist for execution of mobile applications in clouds. (a) Client server model where application server performs complex and extensive computations in the cloud while lightweight client side is executed on mobile device. (b) VM based application execution where mobile application is executed inside a virtual machine that is loaded into the cloud server on the mobile user demand. (c) Application offloading to the clouds with the application programmed using Java and executing inside Java virtual machine. The application execution state requirements and size varies for all the three models. For the purpose of this research study, we have tested the PSS mechanism for application offloading model. However, our mechanism is not dependent on Java runtime environment support.

We suggested that the mobile application should be executed inside the mobile device emulator on the cloud server. The use of emulator on the cloud side enables any mobile application to be offloaded to the cloud server without requiring any modification in the application. Furthermore, the overhead of executing the application inside a virtual machine on the mobile device can be eliminated. This approach also allows the mobile applications to be programmed in any programming language. The primary advantage of this approach is that properly captured application execution state can be used to resume the application either on the mobile device or any other cloud/cloudlet where the emulator support is available. Consequently, the complex task of developing a unified method for cross platform application execution state capture and resume is reduced to a significantly

simpler task of state capture and resume over uniform platform.

Keeping above advantages in view, this research recommends and uses the emulator-based execution of mobile application in the cloud/cloudlet, although the proposed PSS mechanism can work for any execution model. It is assumed throughout the rest of the paper that such an execution environment is available on the cloud server.

4.1.2 Process State Capture and Process Resumption

For ensuring PSS mechanism to work, a key prerequisite is an ability to successfully capture the execution state of application processes from the remote server, transfer the state over the network and resume the application processes on the local computation element (Mobile device). In this section, we briefly explain the process state components that are required for successful application migration across uniform architectures. We also briefly explain the procedure for capturing the process execution state and resuming the process from the captured state.

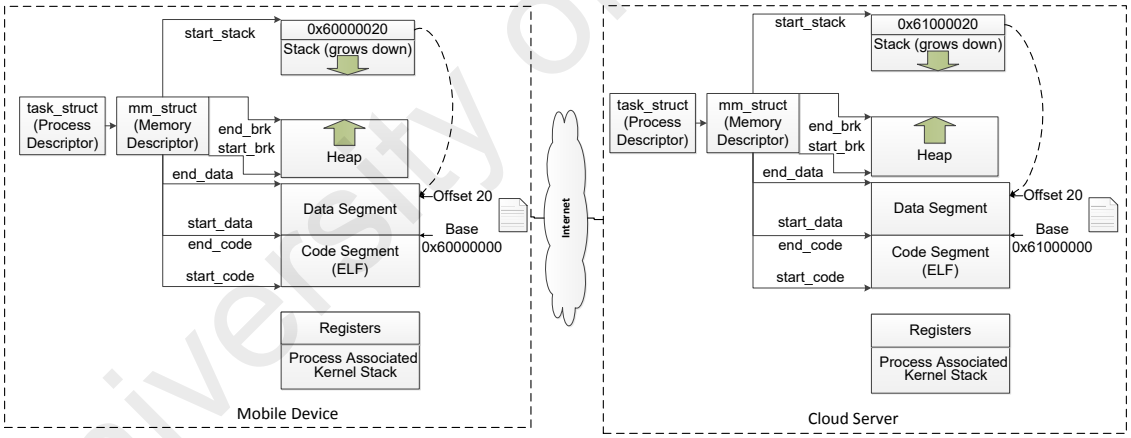


Figure 4.1: Example Process State, highlighting the reference update required while resuming the process

The state of a process comprises of memory segments (Heap, Data, Stack and Code/-Text) for all threads and child processes, File references to the files on the permanent storage or I/O modules and the CPU state. On linux based systems, the references to the memory segments as well as other details of the process are saved in the linux kernel structure of *task_struct*. The CPU register state of every process can be acquired from top of the process-wise kernel stack that is associated with each process. To capture the

process state, a process can be suspended through a kernel level module. The process *task_struct* is used to acquire the detail of all memory segments as well as file references of the process. The segments are mapped on kernel logical address space. The data from segments as well as the process associated kernel stack is copied for transfer over the network to the computational element where the process is expected to be resumed.

To resume the process, a skeleton process is created by executing the mobile application on the machine where process is to be resumed. This requires the availability of the application executable on the system. The newly created process is immediately suspended. This saves the synchronization module from performing the complex task of creating the process state in the linux kernel. The skeleton state is populated with the captured process state by kernel level copy of state data into process memory. Figure 4.1 highlights that the reference update is required while resuming the process on a different computing platform. These references are updated by separating the offset part of the reference and combining the acquired offsets with the segment addresses of the memory segments on the local machine. Multiple mechanisms are available for process state migration over uniform architectures (Litzkow, Tannenbaum, Basney, & Livny, 1997; Osman, Subhraveti, Su, & Nieh, 2002). We used kernel module on the local and remote server to perform the state capture and resume operations.

The size of the process state can be reduced by taking the difference of two consecutive states and transmitting only the updated parts. Similarly, the code segment needs not be transfered because a page fault on code segment can result in loading the required code segment locally at the time of process resumption. With process state migration issue resolved, we turn our attention to the process state synchronization.

4.1.3 State Synchronization

The PSS mechanism must fulfill a number of requirements in order to be effective. Firstly, the synchronization mechanism should be lightweight requiring minimum data transfer between the mobile device and the cloud, keeping in view the limited bandwidth

Table 4.1: Symbols and Their Descriptions

Symbol	Description
A	Application ID
M	Mobile device ID
SyncInterval	Synchronization interval
AppState	Application state
PCID	Previously connected cloud ID
$Dis_{.int}$	Disconnection time interval
T_{dc}	Disconnection time instance
T_{sync}	Synchronization time instance
$\mathscr{F}$	Speedup factor
$T_{sus.}$	Suspend time
$T_{trans.}$	State transfer time
$T_{res.}$	State resume time

of last hop wireless connection. Secondly, the computational overhead of the synchronization mechanism, specifically on the mobile device side should be minimum. Thirdly, in case of disconnection, the mobile device shall be able to resume the computation from a reasonably up to date execution state to ensure minimum loss of computation. Finally, upon reconnection, the mobile device shall be able to synchronize back with the cloud server or resume the computation on an alternate cloud by transferring its current computational state. In case of reconnection with the same cloud server, the mobile device shall also be able to decide on resynchronization or continuation from the last cloud state, depending upon which state is more recent. Note that although Gordon *et al.* (Gordon et al., 2012) proposed the use of synchronization, they have not analyzed if the synchronization provides advantage in computation by verifying the third requirements above. Similarly, their synchronization mechanism does not fulfill fourth requirement and a user, once disconnected, cannot resume the computation back on cloud server in any form.

We present the proposed algorithms for PSS that fulfill the listed requirements. The proposed mechanism requires a module on the mobile device as well as on the cloud side. Table 4.1 presents description of the symbols used in the proposed algorithms.

4.1.3.1 Cloud-side Module

The cloud-side synchronization module is presented in the form of pseudo-code in the Algorithm 1.

The application ID ‘A’, mobile device ID ‘M’, and SyncInterval are inputs to the algorithm. The two IDs help the cloud server from distinguishing multiple applications

Algorithm 1: Process State Synchronization Algorithm (Cloud Server to Mobile Device)

```
   Input: [A, M, SyncInterval]
 1 AppState ← NULL
 2 do
 3 sleep(SyncInterval)
 4 if getNetworkState() = CONNECTED then
 5 |   setAppStatus(A, SUSPEND)
 6 |   CloudAppState ← getAppState(A)
 7 |   sendAppState(CloudAppState, M)
 8 |   setAppStatus(A, RESUME)
 9 end
10 else if getNetworkState()=RECENTLYCONNECTED then
11 |   waitforstate()
12 |   if receivedAppState(MobileAppState, A, M) then
13 |   |   setAppStatus(A, SUSPEND)
14 |   |   setAppState(A, MobileAppState)
15 |   |   setAppStatus(A, RESUME)
16 |   end
17 |   else
18 |   |   setAppStatus(A, RESUME)
19 |   end
20 end
21 while(execApp)
```

of same mobile device as well as same application being executed on the cloud from different devices. When the application starts its execution on the cloud server, cloud synchronization module sets a synchronization timer with the value of synchronization interval, which is a tunable parameter. At the expiry of synchronization timer, the module checks if the network status with the particular device is still connected. If yes, the cloud synchronization module suspends the application, captures its execution states from memory, and resumes the application (Lines 5-8 of Algorithm 1). After capturing the state, the entire state is transmitted to the mobile device. If the network state has recently changed from disconnected to connected (recently connected) the module waits for the state from the mobile device (Lines 10-11). If the cloud server receives the state, the cloud server updates its state to the recently received state and resumes the application (Lines 12-15). Alternatively, the application is resumed from the last available state (Line 18).

4.1.3.2 Mobile-side Module

Algorithm 2 presents the pseudo-code for the module on the mobile device. The primary task of the module is receive and save the state, when it is transmitted by the

cloud. In addition, it checks the network status periodically. If the network status is disconnected and the local instance of application is suspended, this means the network recently got disconnected. In this case, the last received state from the cloud is used to update the application state on the mobile device and the application is resumed (Lines 3-8 of Algorithm 2). In case the application is not suspended, it indicates the application is already executing locally while the network is disconnected and no action is taken (Lines 10-12).

Algorithm 2: Process State Re-Synchronization Algorithm (Mobile Device to Cloud/Cloudlet)

Input: $[A, M, PCID, Dis._{int.}, T_{dc}, T_{sync.}, \mathscr{F}, T_{sus.}, T_{trans.}, T_{res.}]$

```
1  AppState ← NULL do
2  x ← getNetworkState()
3  if x=DISCONNECTED then
4      y ← getAppStatus()
5      if y=SUSPEND then
6          setAppState(A, CloudAppState)
7          resumeApp(A)
8          setAppStatus(A, LOCALEXEC)
9      end
10     else if y=LOCALEXEC then
11         continue
12     end
13 end
14 else if x=CONNECTED then
15     y=getAppStatus()
16     if y=LOCALEXEC then
17         if ResyncRequired(A, PCID, MobileAppState, Dis._int., T_dc,
           T_sync., F, T_sus., T_trans., T_res.) then
18             setAppStatus(A, SUSPEND)
19             MobileAppState ← getAppState(A)
20             sendAppStatetoServer(MobileAppState, A, PCID)
21         end
22         else
23             setAppStatus(A, PCID, SUSPEND)
24         end
25     end
26 end
27 if receivedAppState(CloudAppState, A, PCID) then
28     saveAppState(CloudAppState, A)
29 end
30 while(execApp)
```

If the network status is connected and the application status shows local execution, it indicates that the network connection with the cloud server (same cloud or an alternate cloud/cloudlet) recently got established. In this case, the procedure listed as Procedure

1: ResyncRequired is used. The procedure takes as input the application ID, cloud ID, disconnection interval, disconnection time, synchronization timestamp, speedup factor, suspend time, state transfer time and resume time. Return value is the possible action to be taken by the mobile device. In case of re-connection with same cloud/cloudlet, the mobile device checks if disconnection interval was long enough to have advanced the application on mobile device beyond the expected last state of the application on the cloud server (Line 4 of Procedure 1). If so, the action flag is set to re-synchronize. If the disconnection is short, the flag is set to simple reconnect (Line 7 of Procedure 1).

In case of a new cloud/cloudlet connection, the flag is set to migrate and re-synchronize whereby the mobile device must transfer its application along with the existing execution state of the application for successful resumption on the cloud server (Line 11 of Procedure 1). Depending upon the action required, the mobile device responds to the cloud server either by requesting application resumption without state update or application resumption with state update (state is also transmitted by the mobile device) or application start using execution state (state as well as application code is transmitted by mobile device).

Procedure 1: ResyncRequired($A, PCID, Dis._{int.}, T_{dc}, T_{sync.}, \mathscr{F}, T_{sus.}, T_{trans.}, T_{res.}$)

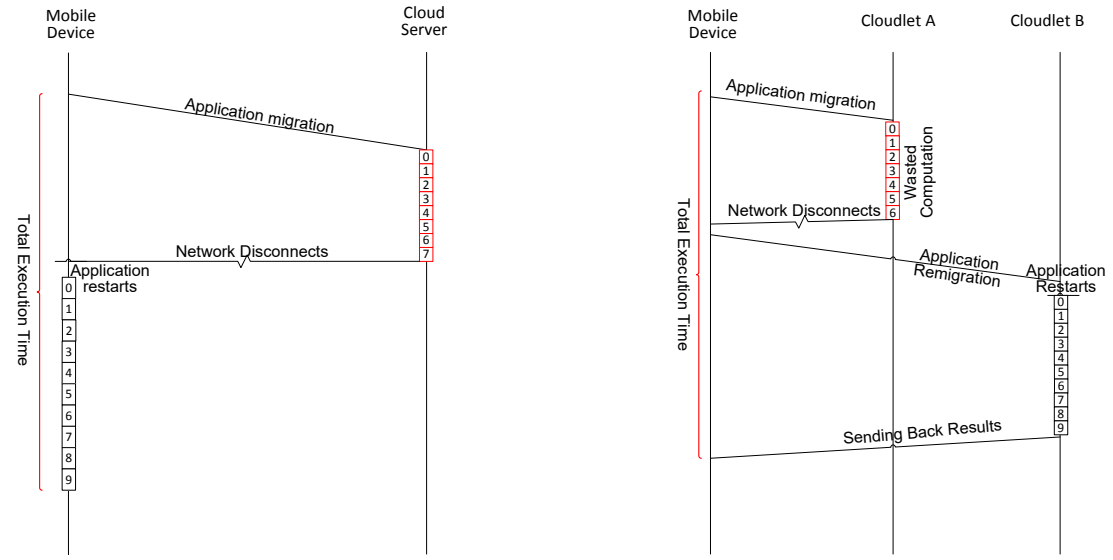
```
1  if A is INTERACTIVE then
2      if getCloudID() = PCID then
3          if Dis._int. > (T_dc - T_sync.)*F + (T_sus. + T_trans. + T_res.) then
4              ReSyncFlag ← RESYNC
5          end
6          else
7              ReSyncFlag ← NORESYNC
8          end
9      end
10     else
11         ReSyncFlag ← RESYNCWITHMIGRATION
12     end
13 end
14 else
15     ReSyncFlag ← NORESYNC
16 end
   Output: [ReSyncFlag]
```

4.2 Example Illustration

In order to further elaborate the concept of application execution process, we illustrate the application execution process in various scenarios. Figure 4.2 (a) illustrates a scenario where the execution restarts on mobile device when the network disconnection occurs. Initially the application is migrated to the cloud server for execution. However, during the execution on the cloud server, the network is disconnected. On the network disconnection detection, the mobile device restarts the application execution by itself. The total execution time is the sum of the application migration time to the cloud, cloud execution time before the network disconnection, and the whole application execution time on the mobile device.

Figure 4.2 (b) illustrates the application re-offloading to another cloudlet after the occurrence of network disconnection. In this scenario, two cloudlets, cloudlet 'A' and cloudlet 'B', have been used. Initially, the application is migrated to the cloudlet 'A' where the execution is started. However, during the execution on the cloudlet 'A', the network disconnection occurs. Consequently, the computation which was performed on the cloudlet 'A' has lost. Meanwhile, the mobile device discovers another cloudlet 'B' and re-offloads its application for execution to the cloudlet 'B'. The execution from the initial stage is again performed on the cloudlet 'B'. On the successful completion of the application execution, the results are sent back to the mobile device. The total execution time is the sum of the application migration time to the cloudlet 'A', cloudlet execution before the network disconnection, application migration time to the cloudlet 'B', whole application execution time on the cloudlet 'B', and the result transfer time to the mobile device.

In these two scenarios, the mobile device did not get intermediate results from the cloud server. Hence, the mobile device has to either re-execute the whole task from initial stage as shown in Figure 4.2 (a) or has to re-offload the application to another cloudlet for execution as shown in Figure 4.2 (b). The network disconnections in these two scenarios



(a) Application Execution on Mobile Device After Network Disconnection

(b) Application Reoffloading on Another Cloudlet After Network Disconnection

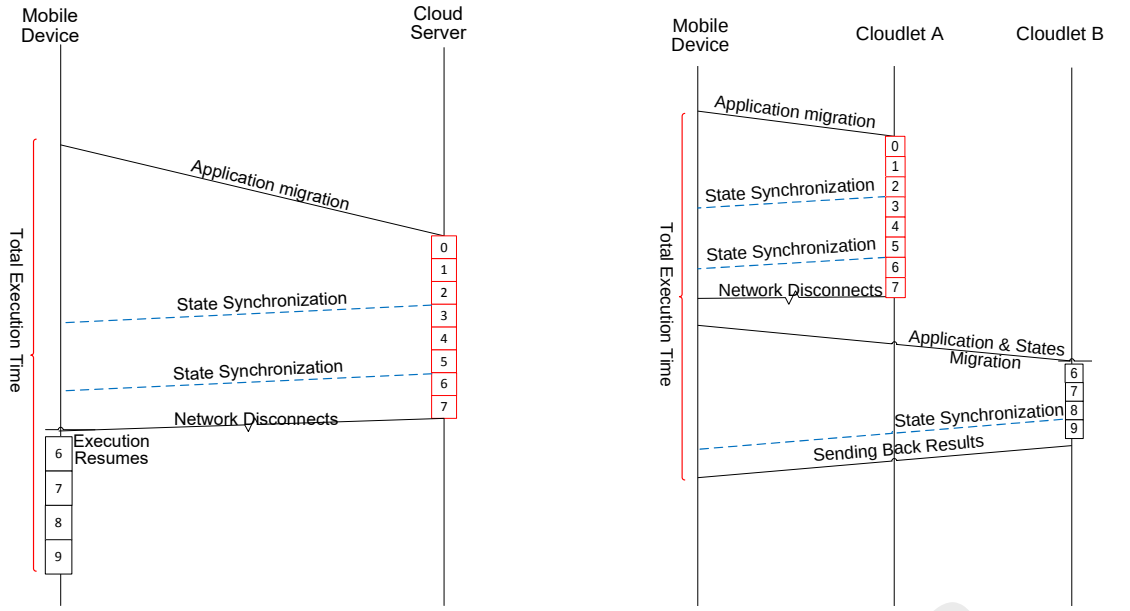
Figure 4.2: Non-Synchronized Application Execution Sequence Diagram for MCC

have resulted into wastage of computation. The re-execution of application from initial stage or re-offloading of the application induces additional delay, consumes more energy, and inefficiently utilizes the bandwidth.

Our proposed solution PSS reduces the computation wastage and the re-execution cost by enabling the cloud server to exchange the intermediate results with the mobile device. Figure 4.3 (a) and Figure 4.3 (b) illustrate the application resumption on the mobile device and on the cloudlet, respectively, in network disconnection scenario.

Figure 4.3 (a) shows that application is initially migrated from the mobile device to the cloud server for the execution. During the execution, cloud server synchronizes the process execution states with the mobile device. When the network disconnection occurs, the mobile device resumes the application execution from the last synchronized states. In this case, the total execution time is the sum of the application migration time to the cloud, cloud execution time before the network disconnection, and the remaining application execution time on the mobile device.

Figure 4.3 (b) shows that application is initially migrated from the mobile device to the cloudlet 'A' where the execution is started. During the execution, the cloudlet 'A' synchronizes application states with the mobile device. Upon the network disconnection,



(a) Application Execution Resumption on Mobile Device After Network Disconnection

(b) Application Execution Resumption on Another Cloudlet After Network Disconnection

Figure 4.3: Synchronized Application Resumption Sequence Diagram for MCC

the mobile device discovers another cloudlet that is cloudlet 'B', to execute the rest of the task. When the mobile device discovers another cloudlet, it migrates the application and last synchronization states from cloudlet 'A' to the cloudlet 'B'. On receiving the application and last synchronization process states, the application execution is resumed on cloudlet 'B'. The rest of the execution is performed on cloudlet 'B' and the results are sent back to the mobile device on the completion of execution.

In these two scenarios, the mobile device gets the intermediate results from the cloud server. Hence, the mobile device can resume the application execution from the last synchronization point either on the mobile device as shown in Figure 4.3 (a) or on the newly discovered cloudlet as shown in Figure 4.3 (b). Hence, the synchronization of the process states between the mobile device and the cloud server reduces the computation wastage after the network disconnection. Moreover, the synchronization minimizes the execution time and energy consumption after the network disconnections.

Figure 4.4 (a) illustrates the interactive application execution scenario of state resynchronization on reconnect. The figure shows that the application is initially migrated to the cloud server where the execution is performed. During the execution, the process

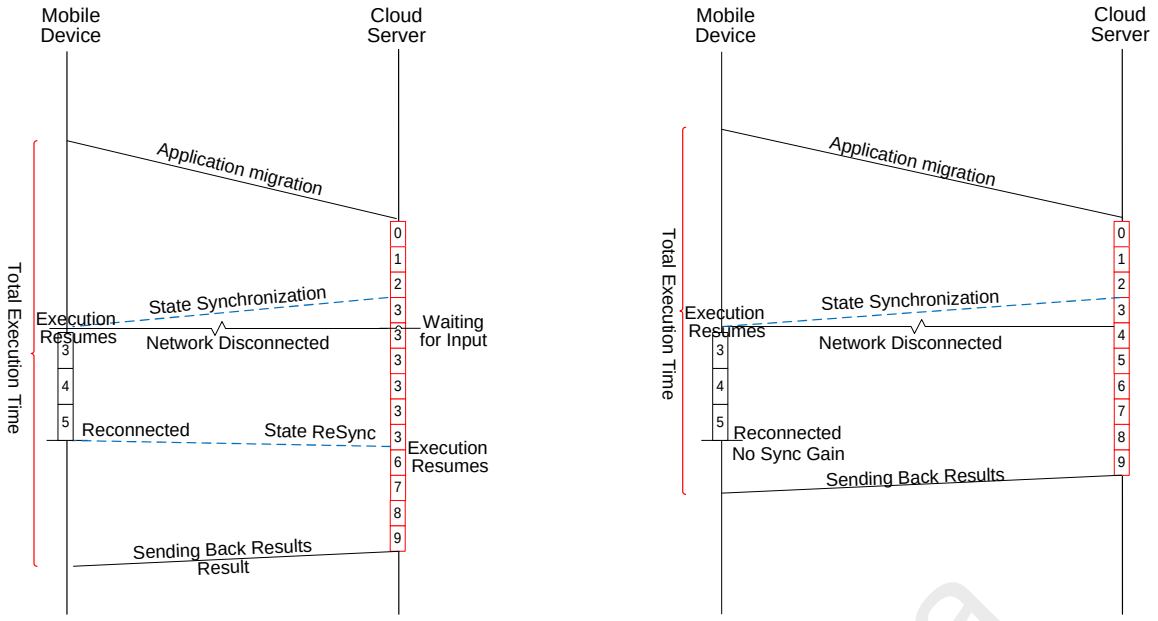
states are synchronized with the mobile device. Meanwhile, the network disconnection occurs and mobile device device starts the execution locally by itself. Execution on the cloud server is suspended due to waiting for the input from the mobile device. When the connection is re-established, the mobile device resynchronizes back the process states with the cloud server. Thereafter, the application execution is resumed on the cloud server from the resynchronization point. The rest of the application is executed on the cloud server and results are sent back to the mobile device.

Figure 4.4 (b) illustrates the non-interactive application execution scenario where no state re-synchronization gain on reconnect is shown. The figure shows that the application is initially migrated to the cloud server where the execution is performed. During the execution, the process states are synchronized with the mobile device. Meanwhile, the network connection is lost and on the disconnection detection the mobile device resumes the application execution locally by itself from the last synchronization point. When the network connection is re-established, the mobile device computes the resynchronization gain. As the cloud server has no gain to resynchronize the states therefore the process states are not synchronized back with the cloud server. As the application is already executing on the cloud server during the disconnection it continues and sends back the results to the mobile device when network connection re-established.

4.3 Mathematical Model of PSS

The overhead and the efficiency of the algorithms presented in the previous section are dependent on proper selection of the synchronization interval. The synchronization interval is dependent upon disconnection frequency and disconnection intervals. Although both frequency of disconnections and the duration of disconnections is unpredictable, the average value of duration can be computed by observing the disconnection pattern in the time proceeding the application execution. This can easily be achieved using already available mechanisms within the mobile devices.

In this section, we develop a mathematical model to establish a correlation between



(a) State Re-Synchronization on Reconnect

(b) No State Re-Synchronization on Reconnect

Figure 4.4: Re-Synchronization Decision on Reconnect Sequence Diagram for MCC

synchronization interval, disconnection interval, cloud to mobile processing speedup ratio and the cloud-based mobile application execution time under intermittent network connectivity. We derive the sufficient condition on synchronization interval for PSS to ensure reduced execution time of mobile application. We also derive the upper bound on synchronization interval such that a higher value of synchronization interval results in negligible reduction in execution time and leads to significant overhead on mobile device in terms of wasted computations. We only present the case of local resumption on mobile device after disconnection using PSS because the worst case execution time is observed for PSS in this case compared to remote resumption, which is significantly more beneficial. The variables used in this section are summarized in Table 4.2.

4.3.1 Application Execution Time

The execution time of an application depends on many factors including the CPU time required by the application, operating system overhead, number of other applications requiring the CPU time, direct memory access time and the wait time on resources such as files, hardware peripherals *etc*. For the purpose of analysis, we assume that the application only requires CPU time and automated access of a set of resources where it does not have

Table 4.2: Symbols and descriptions

Symbol	Description
P_m	Mobile device speed (Million Instructions per Second (MIPS))
P_c	Cloud processing speed offered for mobile device (MIPS)
f	Ratio of Cloud to mobile device processing speed
r	Data Rate (Mbps)
l	Propagation delay
I	Expected number of application instructions
I_{mu}	Expected number of mobile useful instructions
I_{mw}	Expected number of mobile wasted instructions
$\mathscr{M}$	Code Migration Size (Bits)
T_{sync}	Synchronization interval
T_{dc}^{j}	jth disconnection interval
t_{dc}^{j}	Time of jth disconnection
t_{sync}^{k}	Time of kth synchronization

to wait. Under this assumption the execution time of application being executed on a local resource can be given in terms of expected number of instructions, processing speed in million instructions per second (MIPS), operating system overhead and the wait time because of time sharing with other applications. This expression for local execution time is given in Equation 4.1 where I is the expected number of executed machine instructions, P_m is the execution speed of processor in MIPS, O is the overhead instructions because of operating system and n is the number of other processes in the system.

$$T_{le} = \frac{I}{P_m} + \frac{O}{P_m} + \sum_{n} \frac{I}{P_m} \tag{4.1}$$

If the same application is executed on a remote computational element, then the overhead because of interaction with the local resources will add to the execution time of application. Similarly, if offloading of code is required to transfer the application from local to remote computational element, the offloading time will be added to the application. Furthermore, in case of intermittent network connectivity, the execution time for interactive applications increases by the total duration of disconnection. Assuming the interaction is required after every I_i machine instructions and assuming that the data transfered during interaction is negligible, the remote execution time of interactive application under intermittent network connectivity is given by Equation 4.2. In the equation,

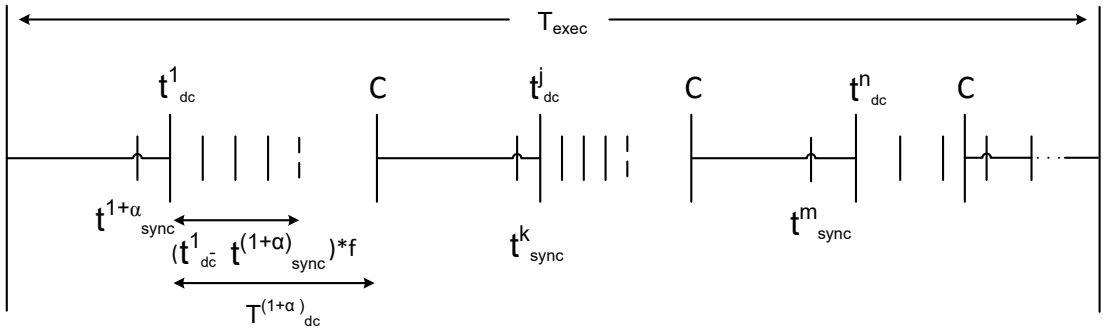


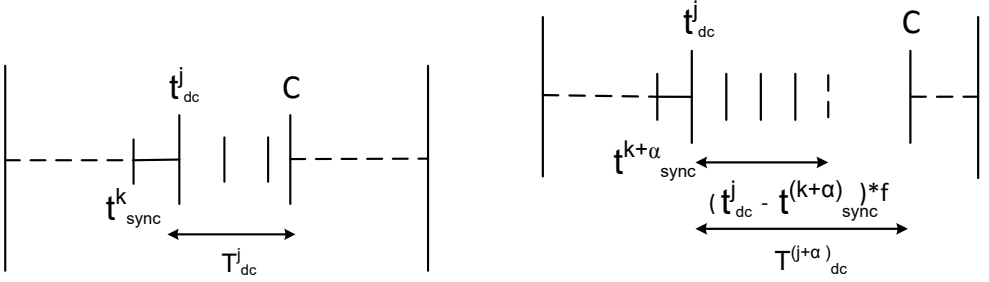
Figure 4.5: Illustration of useful and useless computation disconnection intervals

r is the data transfer rate, M is size of migration state in bits and l is propagation delay while T_{dc}^{j} is the duration of jth disconnection interval.

$$T_{re}^{dc} = \frac{I}{P_c} + \frac{O}{P_c} + \sum_{n} \frac{I}{P_c} + \frac{I}{I_i} * 2 * l + \frac{M}{r} + l + \sum_{j} T_{dc}^{j} \tag{4.2}$$

4.3.2 Application Execution Time with PSS

If PSS is enabled, in case of disconnection, the mobile device can locally resume the computation from the last received state. With reference to the execution being performed by the mobile device, the disconnection intervals can be divided into two sub-intervals, as shown in Figure 4.5. During first interval, mobile device will re-perform the execution that was performed on the cloud but not synchronized with the mobile device. The duration of this interval is dependent upon the interval between last synchronization and disconnection as well as on the processing speed of the mobile device and is given by the expression $(t_{dc}^{j} - t_{sync}^{k}) * f$ where f is the ratio of processing speed of the cloud and the mobile device. Any computation performed during this interval by the mobile device is wasted if the mobile device reconnects with the same cloud after some time. The number of wasted instructions execution performed on the mobile device is given by Equation 4.3. During the second interval, mobile device will execute the part of application that has not yet been executed by the cloud and resynchronization by mobile device with the cloud will take place upon reconnection. The computation performed during the second inter-



(a) DC interval with Wasteful Comp. (b) DC interval with Useful and wasteful comp.

Figure 4.6: Illustration of DC interval for useful and wasteful computations

val of the disconnections is useful computation. The number of instructions executed during this interval are given by Equation 4.4 where k is last synchronization event before disconnection event j. Note that the interval $t^j_{dc} - t^k_{sync}$ is dependent upon the disconnection profile and the synchronization interval T_{sync}. The value of this interval varies between 0 and $T_{sync} - 1$.

$$I_{mw} = \sum_j min((t^j_{dc} - t^k_{sync}) * f, T^j_{dc}) * P_m \tag{4.3}$$

$$I_{mu} = \sum_j max(T^j_{dc} - (t^j_{dc} - t^k_{sync}) * f, 0) * P_m \tag{4.4}$$

Using the useful computations performed by the mobile device, the execution time of application under intermittent connectivity with PSS is given by Equation 4.5.

$$T^{sync}_{re} = \frac{I - I_{mu}}{P_c} + \frac{O}{P_c} + \sum_n \frac{I - I_{mu}}{P_c} + \frac{I - I_{mu}}{I_i} * 2 * l + \frac{M}{r} + l + \sum_j T^j_{dc} \tag{4.5}$$

4.3.3 Sufficient Condition for Usefulness of PSS

Process state synchronization will be beneficial only if the application execution time with synchronization is lesser than the execution time without synchronization. For permanent disconnections or reconnection to a different cloud/cloudlet upon reconnection, it is trivial to show that $T^{sync}_{re} < T^{dc}_{re}$. Therefore, we focus on the case of intermittent connectivity with same cloud and derive the sufficient condition that ensures T^{sync}_{re} is less

than T_{re}^{dc}.

We observe that $T_{re}^{sync} < T_{re}^{dc}$ holds as long as $I_{mu} > 0$. I_{mu} will be positive if there exists at least one disconnection interval j such that expression 4.5 holds. This can be achieved in multiple ways. First, if synchronization interval $T_{sync} = \frac{1}{f}$ then even in the worst case of $t_{dc}^{j} - t_{sync}^{k} = T_{sync}$, we get $(t_{dc}^{j} - t_{sync}^{k}) * f - 1 = 0$; however, such a small synchronization interval will lead to very high synchronization overhead and is not a suitable option. Second, if synchronization is performed just prior to disconnection then $(t_{dc}^{j} - t_{sync}^{k}) * f - 1 = 0$. This requires accurate disconnection event prediction, which is impractical. Most practical option is to estimate the average disconnection interval $\bar{T}_{dc}$ and set the synchronization interval as given in Equation 4.6. Under this condition, all disconnection intervals greater than the average disconnection interval will fulfill condition listed in Expression 4.5 even in worst case scenario of $t_{dc}^{j} - t_{sync}^{k} = T_{sync} - 1$ resulting in $I_{mu} > 0$. Therefore, Equation 4.6 gives the sufficient condition for PSS to be useful.

$$T_{dc}^{j} - (t_{dc}^{j} - t_{sync}^{k}) * f - 1 > 0 \tag{4.5}$$

$$T_{sync} = \frac{\bar{T}_{dc}}{f} \tag{4.6}$$

4.3.4 Upper-bound on Synchronization Interval

We now compute the upper bound on synchronization interval such that further increase in T_{sync} does not lead to reduced application execution time under PSS. Experiments show that a particular value of synchronization interval results in equal number of useful and wasteful computations being performed by the mobile device. Experiments also show that further increase in synchronization interval beyond this value causes the sharp decline in useful computations and a sharp increase in wasteful computations as shown in Figure 4.7. We select the synchronization interval as upper bound where useful and wasteful computations performed by mobile device are equal, i.e., $I_{mu} = I_{mw}$.

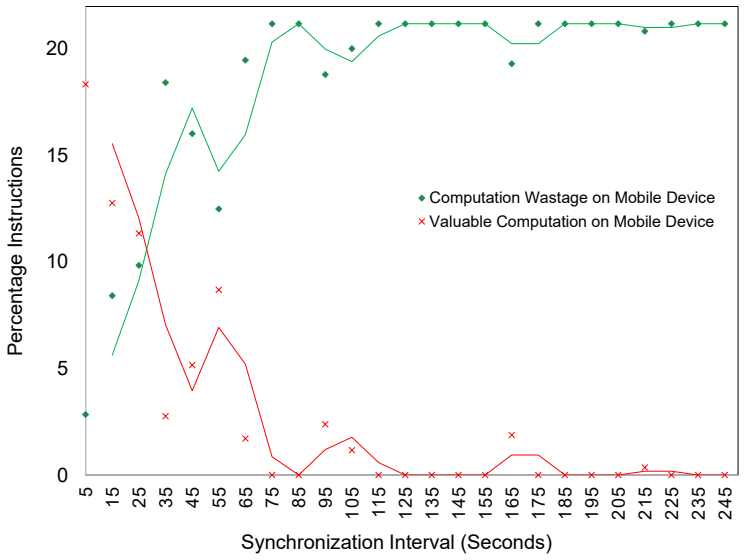


Figure 4.7: Synchronization Interval For Balancing Trade-off between Mobile Device Useful and Useless Computation

Useful and wasteful instructions are dependent on: (a) the interval between last synchronization time t^k_{sync} and the disconnection time t^j_{dc} and (b) the duration of disconnection T^j_{dc}. We have $0 \leq (t^j_{dc} - t^k_{sync}) < T_{sync}$. For computation of upper bound, we use the average value of interval $(t^j_{dc} - t^k_{sync})$ which is $\frac{T_{sync}-1}{2}$. The results show that the computation of upper-bound is very accurate even with this simplifying assumption. Assuming the interval $(t^j_{dc} - t^k_{sync})$ to be equal to $\frac{T_{sync}-1}{2}$, the disconnection intervals fall in two groups. If $T^j_{dc} \leq \frac{T_{sync}-1}{2} * f$, no useful computation is performed by the mobile device during this interval as shown in Figure 4.6 (a). Alternatively, if $T^j_{dc} > \frac{T_{sync}-1}{2} * f$, then $\frac{T_{sync}-1}{2} * f$ amount of time is spent in wasteful computation while $T^j_{dc} - \frac{T_{sync}-1}{2} * f$ amount of time is spent in useful computation as shown in Figure 4.6 (b). Suppose a disconnection intervals out of a total of n intervals satisfy $T^j_{dc} \leq \frac{T_{sync}-1}{2} * f$, then using above assumption $I_{mu} = I_{mw}$ can be written as Equation 4.7.

$$\sum_{i=a+1}^{n} \left[T^i_{dc} - \frac{T_{sync}-1}{2} * f \right] = \sum_{i=1}^{a} T^i_{dc} + \sum_{i=a+1}^{n} \frac{T_{sync}-1}{2} * f \tag{4.7}$$

After simplification and replacing $\sum_{i=1}^{n} T^i_{dc} = n\bar{T_{dc}}$ and $T^i_{dc} = \frac{T_{sync}-1}{2} * f - T^i_{\sigma}$ for $T^j_{dc} \leq \frac{T_{sync}-1}{2} * f$, we get the synchronization interval as given by Equation 4.8. The value of T_{sync} can be computed iteratively for given value of average disconnection interval and

distribution of the disconnection intervals. This value gives the upper bound on synchronization interval for a given average value of disconnection interval.

$$T_{sync} = 1 + \frac{1}{f}\left[\bar{T_{dc}} + \frac{2}{n}\sum_{i=1}^{a} T_{dc}^{i}\right] \tag{4.8}$$

4.4 Distinguishing Features of Proposed Algorithm

We discussed in the literature review that a number of application execution frameworks have been proposed by the researcher. Some of these frameworks including COMET address the issue of network disconnections; however, the solutions proposed in the literature are not adequate enough to mitigate the adverse impact of network disconnections. In this section, we highlight how the proposed algorithms can address the shortcomings of the existing solutions. Following are the distinguishing features of the proposed PSS-based execution, which make it a distinct execution solution for interactive cloud-based mobile applications in MCC.

4.4.1 Adaptive Synchronization Interval

The PSS-based execution is adaptive that enables the cloud server to tune the synchronization frequency interval with the changes in the MCC environment. The advantage of making the synchronization interval adaptive is that when the network is stable, the synchronization frequency will be minimized that results in low synchronization overhead. However, in case of highly dynamic environment, synchronizing the process states after a long interval increases the computation wastage. Hence, the synchronization interval should be adjusted to smaller value to frequently exchange the process states.

4.4.2 Lightweight Synchronization Mechanism

The synchronization is mainly performed by the cloud server during the application execution to exchange the process states with the mobile device. The mobile device has to receive the process states during the execution which consumes less energy as compared to transmitting the process states. However, the resynchronization process that is

implemented on the mobile device exchanges the mobile executed process states with the cloud server only on the re-establishment of the connection. Hence, the resynchronization process also incur less overhead because of the synchronization on only connection establishment. That is less frequent than the periodic synchronization of process states. Therefore, the PSS-based execution is lightweight in terms of energy consumption.

4.4.3 Distributed Algorithm

The synchronization algorithm is implemented and run in the distributed form, thereby reducing the information exchange overhead for selecting the synchronization interval. The operational complexity of the algorithm is also divided between two ends, mobile device and the cloud server. The distributed nature of the algorithm aids in implementing the only specific functionality required for the computing platform. As on the cloud server, periodic synchronization mechanism is required to periodically exchange the process states with the mobile device during the execution. Hence, the synchronization algorithm implements the periodically checks for the updated states, collects the process states and exchanges the information with the mobile device. However, the synchronization requirements on the mobile device are different as the mobile device has to exchange only the process states of mobile computed execution after the reconnection establishment. Most importantly, the mobile device is resource constrained so a lightweight resynchronization procedure is required.

4.4.4 Two-way Synchronization

The most prominent feature of PSS that sets it distinctly apart from the existing solutions is the use of two-way synchronization. Unlike COMET where application state in cloud is synchronized with the mobile device, in case of PSS, the mobile device can also send its state to the cloud/cloudlets. This reverse synchronization helps in case of re-connections, which is often the case in network connectivity profile. The resynchronization algorithm is implemented on the mobile device to exchange the locally executed process states with the cloud server on the reconnection establishment. This enables

PSS based execution to take advantage of the faster cloud based computations upon reconnection, which is not the case with COMET. The faster cloud based execution upon reconnection significantly improves the application execution time.

This feature further liberates the mobile user because the user can have the saved state on the mobile device and suspend the application execution during significantly longer disconnections. The user can restart the application by transferring its state to the cloud server, when it gets connected with the server, following an extended period of time. Therefore, PSS provides significantly higher level of flexibility and improved computation time for cloud based mobile applications, compared to the existing solutions.

4.5 Conclusion

In this chapter, we proposed a distributed synchronization algorithm for mobile application execution in disruptive network conditions of MCC. Majority of the state-of-the-art application execution frameworks start the application execution from initial stage on mobile device or re-offload the whole task execution to nearby available cloudlets when the network disconnection occurs. The proposed synchronization algorithm enables the cloud server to exchange the intermediate execution states with the mobile device. The execution can be resumed on the mobile device from the last synchronized point when the network disconnection occurs. Upon reconnection, the execution state from mobile device is synchronized back with the cloud server. In case of same cloud, the most current of the two states is used to resume the computation on the cloud server.

The synchronization of the process states between cloud server and mobile device is expected to reduce the overall execution time in disruptive network conditions with an objective to minimize the synchronization overhead, energy consumption, and computation wastage. The computation wastage is minimized through the synchronization of the intermediate states with the mobile device. The energy consumption is minimized by utilizing the previously received synchronization states of a process on mobile device instead of executing from initial stage. The additional execution delay is decreased by

resuming the application from last sync point. In the subsequent chapters, we present the implementation details of the proposed algorithm, collect data using mathematical model and the implementation, verify the model and present the performance analysis of the proposed solution.

This chapter discusses the data collection method for the evaluation of proposed process state synchronization algorithm in MCC environment. The purpose of the chapter is to explain the experimental setup used for testing the performance of the proposed algorithm, mathematical model parameters, data collection technique, and the statistical method used for examining the accuracy of the data. The chapter also presents the data collected for evaluating the performance of the proposed algorithm.

The chapter is organized into seven sections. Section 5.2 explains the experimental setup, connection execution profiles, prototype application and performance metrics, and data gathering and data processing. Section 5.3 presents the data collected to validate the correctness of developed mathematical model by comparing the results obtained from mathematical model with results of the experiments. Section 5.4 reports the collected data for execution time and total computation wastage in case of different disconnection profiles. Section 5.5 reports the data collected to analyze the impact of synchronization interval on the following parameters: a) execution time, b) number of resynchronization, c) synchronization overhead, d) mobile device computation wastage, e) cloud computation wastage, f) valuable computation performed on the mobile device, g) number of instructions computed between last sync point and disconnection, and h) energy consumption. Section 5.6 presents the data collected for the performance comparison of PSS-based execution with optimized VM-based cloudlet and COMET. Finally, Section 5.7 extracts conclusive remarks.

5.1 Introduction

Process state synchronization algorithm is designed to enable the application execution frameworks for executing the application with minimum execution overhead in disruptive network conditions of MCC. The significance of PSS is evaluated in emulated mobile cloud computing setup and with mathematical modeling. The performance of PSS

is evaluated by varying the synchronization interval, for different connectivity data traces, and comparing with the benchmarked solutions.

A prototype application was developed to run on the mobile device as well as on the cloud server which is tested in mobile cloud computing environment for five different types of disconnection-execution profiles: a) permanent disconnection execution, b) permanent disconnection execution with synchronization, c) intermittent disconnection execution, d) intermittent disconnection execution with synchronization, and e) always connected. The performance of an application is evaluated with respect to execution time, computation wastage in form of number of instructions, energy consumption, and synchronization overhead. The sample mean for the sample space of 30 values is determined that is signified by finding the error estimate for 99% confidence interval. The results of testing the prototype application are also compared to validate the significance of the proposed solution.

5.2 Performance Evaluation

This section presents the methodology used for the evaluation of the process state synchronization algorithm. We discuss the experimental setup, five types of disconnection-execution profiles used for comparison, prototype application used for the evaluation, and data gathering and processing method.

5.2.1 Experimental Setup

We implemented process state synchronization algorithm in distributed form on a mobile device and on the cloud server at linux kernel level. For conducting the experiments, we emulated 3G connectivity traces in the lab environment over the LAN network. The disconnection has been emulated by dropping the packets between the emulated cloud server and emulated mobile device. The execution state capture and resume has been implemented as linux kernel module while a simple client server application provides communication between the emulated mobile device and the cloud server. Processing speed in terms of million instructions per second on the emulated mobile device

Table 5.1: Systems specifications

Specification	System 1 (Emulated Cloud Server)	System 2 (Emulated Mobile Device)
Processor	2GHz	2GHz
RAM	8GB	4GB
BogoMIPS	5984	4655
Operating System	Ubuntu 12.04.4 LTS	Ubuntu 12.04.4 LTS

has been matched with Samsung Galaxy II by generating background processes on the computer. On the cloud side, Openstack based cloud is deployed and a 4 core virtual machine with 2.4 GHz processor is created for the mobile device. The virtual machine on the cloud server provides a processing speedup factor of 2.66 for the emulated mobile device.

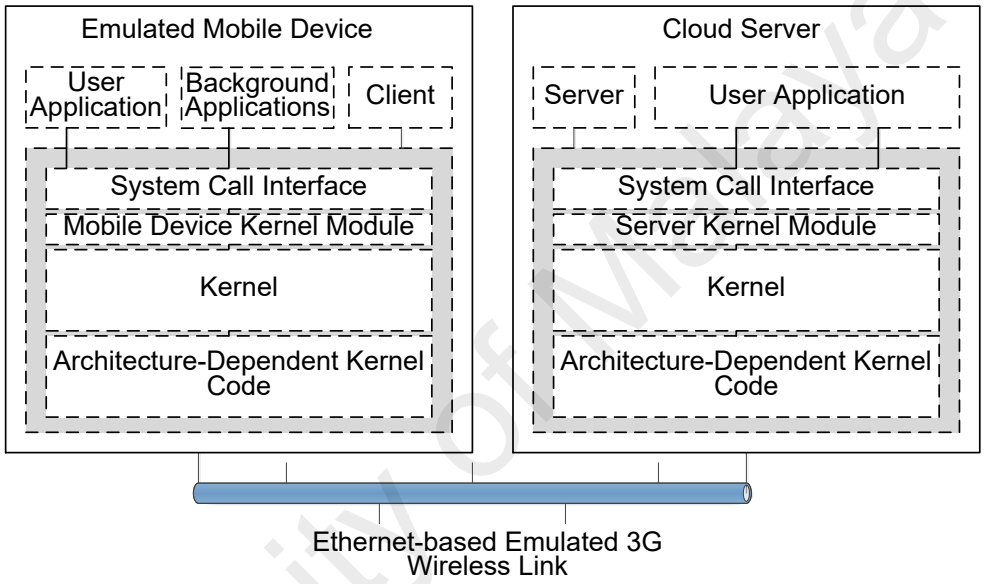


Figure 5.1: Experimental Setup Illustration

The tested application is a simple algebraic computation repeated for the given input through automated interaction. We computed the number of instructions of the applications for use in the model using Lackey tool for valgrind (Developers, 2015). Number of other processes present in the system were kept uniform on the emulated mobile device and emulated cloud server and were limited to automated background linux processes that remained in sleep state for entire experiment. Operating system overhead has been set to 7% of the total execution time and has been estimated using top command. Table 5.1 presents the specifications of the systems used for the conducted experiments. Figure 5.1 presents the experimental setup illustration of the emulated MCC environment.

5.2.2 Connection Execution Profiles

We conducted a comparative study for five different scenarios: a) permanent disconnection, b) permanent disconnection with synchronization, c) intermittent disconnection, d) intermittent disconnection with synchronization, and e) always connected. In permanent disconnection scenario, the mobile device never connects back with the cloud server once disconnected from the cloud. Hence, the execution that is performed on the cloud server is wasted and after the disconnection the mobile device starts the execution from the initial stage on the mobile device.

The execution time in case of permanent disconnection can be reduced by synchronizing the process states with the mobile device. In this scenario, the mobile device can resume the execution from the last synced point after the disconnection, thereby minimizing the wastage of previously performed computations on the cloud server. In intermittent disconnection scenarios, the disconnections occur more than one time during the execution of the application in the cloud. However, with intermittent connectivity the execution in the cloud depends on the nature of the application. In case of interactive applications, the execution on the cloud may suspend for input from mobile device when the mobile device is disconnected from the network. For non-interactive applications, the cloud server continues the execution during the disconnection period. On the connection reestablishment, the mobile device can get back the results from the cloud server if the computation is complete. We are considering a simple algebraic interactive application.

Similar to execution time of permanent disconnection execution, the execution time in case of intermittent disconnections can be minimized by synchronizing the process states with the mobile device. The mobile device can resume the application execution locally during the network disconnection. On the network connection reestablishment, the mobile device synchronizes back the process states with the cloud server. Thereafter, the cloud server resumes the process. On the completion of execution, the results are sent back to the mobile device. In case of always connected scenario, an offloaded application

will complete its execution on the cloud server, which gives less application execution response time.

5.2.3 Prototype Application and Performance Metrics

To evaluate the performance of the process state synchronization algorithm in MCC environment, we designed and developed a simple algebraic computation repeated for the given input through automated interaction. The interactive component is added to emulate the real interaction with the mobile device. We analyze the performance of our proposed solution by studying the following performance measuring metrics.

1. Application Execution Time
2. Computation Wastage
3. Synchronization Overhead
4. Energy Consumption

The application execution time does not consider the offloading decision or application migration time. We are assuming that application is already offloaded into the cloud. Hence, the execution time is the time taken to perform computation on the cloud server, disconnection time, and time taken to perform the computation on the mobile device or results sent back time. The application execution time is measured in seconds.

The computation wastage represents the number of computed instructions wasted on a computing device. The computation wastage is studied on both sides: the cloud server and the mobile device. In case of permanent disconnection with synchronization, the instructions are wasted on the cloud server but no instruction is wasted on the mobile device. The instructions wasted on the cloud server are the instructions computed between the last sync point and disconnection point. The reason of no computation wastage on the mobile device is that mobile device either complete the execution by itself or offload it to other cloudlet server. In case of intermittent disconnection, the computation is only wasted on the mobile device and no computation is wasted on the cloud server. The

Table 5.2: Mathematical model parameters and their corresponding values

Parameter	Value
P_m	$856 * 10^6$ Instructions Per Second
f	2.667
P_c	f* P_m
l	10 ms
I	$2.25 * 10^{12}$
$\mathscr{M}$	3 KB
T_{sync}	[5, 25, 45, 65, 85, 105, 125, 145, 165, 185, 205, 225, 245]

mobile device computation wastage is due to shorter network disconnection during which the mobile device resumes the application execution locally but after reconnection the mobile device does not resynchronize back with the cloud server. The re-computation of instructions from last sync point to disconnection point, which is performed by mobile device before the useful computation is also considered as computation wastage on the mobile device. The reason is that the cloud server has already computed the instructions.

The synchronization overhead represents the number of bytes that needs to be exchanged during the execution of the application on the cloud server for synchronizing the process states with the mobile device and for synchronizing back from the mobile device with the cloud server. The synchronization overhead is measured in number of bytes. The energy consumption is a parameter that measures the consumption of energy on the process states synchronization from cloud server to mobile device and vice versa. The energy consumption is measured in joules.

5.2.4 Data Gathering and Data Processing

The performance measuring parameters are investigated in diverse conditions by varying the system variables such as synchronization interval. These variables effects have been investigated on the performance measuring parameters. First of all, we compared the results of the mathematical model with the results of execution in the emulated MCC environment for validating the mathematical model. Thereafter, we used the mathematical model to collect the results for different performance measuring parameters. The parameter values used for the mathematical model are presented in Table 5.2.

The primary data is collected by testing the prototype application in the emulated MCC environment for different scenarios. The data is collected for application execu-

tion time in case of thirty different mobile user connectivity data traces. The impact of synchronization interval on execution time is analyzed by varying the synchronization interval. The data for application execution time and total computation wastage is collected for thirty different connectivity traces in case of different disconnection profiles.

Similarly, the data is collected for valuable computation performed on process state synchronization enabled mobile device during the disconnection period. We also evaluated the impact of synchronization interval on: a) execution time, b) number of resynchronizations, c) synchronization overhead, d) cloud computation wastage, e) mobile device computation wastage, f) valuable computation performed on mobile device, and g) number of instructions executed between last sync point and disconnection point. Further, we collect the data for the comparison of our proposed solution with the state-of-the-art application execution frameworks in case of application execution time, process states exchange overhead, and energy consumption.

5.3 Data Collected For Model Validation

The correctness of developed mathematical model is validated by comparing the results obtained from mathematical model with empirical results. The execution time, useful computation performed on mobile device, wasted computation performed on mobile device are parameters which we studied for model validation.

Table 5.3: Application Execution Time Computed Through Mathematical Model and Experiments

Synchronization Interval (seconds)	**5**	**15**	**25**	**35**	**45**	**95**	**145**	**195**	**245**
Experiment Data Trace-1	1341	1355	1352	1360	1368	1398	1478	1437	1437
Mathematical Model Data Trace-1	1291	1293	1313	1358	1369	1382	1347	1382	1382
Experiment Data Trace-2	1287	1301	1332	1323	1354	1420	1386	1386	1386
Mathematical Model Data Trace-2	1271	1287	1318	1321	1329	1358	1357	1358	1358
Experiment Data Trace-3	1376	1401	1413	1397	1446	1447	1511	1522	1532
Mathematical Model Data Trace-3	1330	1349	1391	1377	1423	1440	1440	1438	1440

[1]Note: Data presented in the table is in seconds.

Table 5.3 presents the comparison of execution time computed through mathematical model and experiments. The first row and the first column of the table represent the synchronization intervals for which the execution time is studied and the experiment type respectively. The execution time is presented for different synchronization intervals of

three mobile user connectivity data traces. The average difference in execution time of mathematical model and experiments is 2.9%, 1.8% and 3% for the three traces. This small amount of difference validates the model results.

Table 5.4: Useful Instructions Computed Through Mathematical Model and Experiments

Synchronization Interval (seconds)	**20**	**23**	**27**	**31**	**35**
Experiments Run-1	6.75	8.25	10.25	13.25	10.92
Experiments Run-2	6.67	8.33	9.67	11.75	12.33
Experiments Run-3	7.67	8.33	8.83	12.00	10.92
Mathematical Model	6.75	8.40	10.40	12.44	11.46

[2]Note: Data presented in the table is percentage of useful instructions out of the total instructions.

Table 5.4 and Table 5.5 present the comparison of useful instructions computed and wasted instructions through mathematical model and experiments, respectively.

The first column and the first row of the tables represent the experiment type and synchronization interval respectively. Each column other than first column also represents data for separate trace. The selected synchronization interval is an ideal interval for each data trace where useful and useless instructions are equal in case of mathematical model. The presented data for useful and wasted instructions computed on the mobile device are shown in the form of percentage out of the total number of instructions. The average percentage differences in useful instructions computed on mobile device through mathematical model and experiments are 0.33%, 0.096%, 0.82%, 0.65%, 0.65% for five different data traces. Similarly, the average percentage differences in computed wasted instructions on mobile device through mathematical model and experiments are 0.15%, 0.26%, 0.41%, 0.88%, and 1.02% for five different data traces. This small amount of differences validate the model results for useful and wasted instructions computed on the mobile device.

Table 5.5: Wasted Instructions Computed Through Mathematical Model and Experiments

Synchronization Interval (seconds)	**20**	**23**	**27**	**31**	**35**
Experiments Run-1	6.75	8.54	10.54	11.63	12.00
Experiments Run-2	6.96	8.54	10.17	11.42	10.38
Experiments Run-3	7.00	8.92	9.54	11.63	12.92
Mathematical Model	6.75	8.40	10.40	12.44	11.46

[3]Note: Data presented in the table is percentage of the wasted instructions out of the total instructions.

5.4 Data Collected for Different Disconnection-Execution Profiles

We study the execution time and computation wastage for different disconnection-execution profiles. This section presents the collected data for the conducted studies.

5.4.1 Execution Time

Table 5.6 presents the collected data for execution time of an application in different disconnection-execution profiles.

Table 5.6: Application Execution Time for Different Disconnection-Execution Profiles

Synchronization Interval (seconds)	Perm. Disc.	P. D. W. Sync	Int. Disc.	I. D. W. Sync	Al. Con.
Data Trace-1	3358	1931	1677	1458	994
Data Trace-2	3223	2168	1662	1458	1002
Data Trace-3	3346	1962	1632	1431	994
Data Trace-4	3214	2172	1617	1435	994
Data Trace-5	3190	2220	1610	1426	990
Data Trace-6	3525	1656	1396	1292	994
Data Trace-7	3131	2318	1568	1405	995
Data Trace-8	3086	2387	1566	1397	996
Data Trace-9	3361	1935	1539	1301	992
Data Trace-10	3223	2168	1512	1368	993
Data Trace-11	3324	1997	1498	1302	1004
Data Trace-12	3344	1960	1344	1253	990
Data Trace-13	3169	2256	1496	1357	991
Data Trace-14	3204	2191	1496	1320	996
Data Trace-15	3247	2120	1487	1345	999
Data Trace-16	3235	2136	1483	1353	989
Data Trace-17	3268	2084	1483	1236	991
Data Trace-18	3168	2255	1473	1338	993
Data Trace-19	3268	2084	1464	1234	996
Data Trace-20	3182	2226	1438	1315	999
Data Trace-21	3112	2342	1434	1309	994
Data Trace-22	3482	1727	1198	1155	994
Data Trace-23	3281	2068	1428	1286	998
Data Trace-24	3330	1989	1382	1259	997
Data Trace-25	3422	1824	1239	1188	996
Data Trace-26	3199	2200	1379	1272	1000
Data Trace-27	3235	2136	1371	1278	999
Data Trace-28	3345	1961	1329	1249	995
Data Trace-29	3218	2176	1358	1269	994
Data Trace-30	3308	2024	1293	1189	997
Mean	3267	2089	1462	1316	995
Standard Deviation	103	172	120	81	4
Confidence Interval	3267 ± 49	2089 ± 81	1462 ± 56	1316 ± 38	995 ± 2

[4]Note: Perm. Disc.: Permanent Disconnection, P. D. W. Sync: Permanent Disconnection With Synchronization, Int. Disc.: Intermittent Disconnection, I. D. W. Sync: Intermittent Disconnection With Synchronization, Al. Con.: Always Connected

[5]Note: Data presented in the table is in seconds.

These executions were performed in permanent disconnection, permanent disconnection with synchronization, intermittent disconnection, intermittent disconnection with synchronization and always connected scenarios. The results are collected for 30 different connectivity data traces. The column permanent disconnection represents the execution time for scenarios where the execution on the cloud server is disrupted due to permanent disconnection. The execution time for permanent disconnection is comprised of execution

Table 5.7: Computation Wastage for Different Disconnection-Execution Profiles

Synchronization Interval (seconds)	Perm. Disc.	P. D. W. Sync	Int. Disc.	I. D. W. Sync	Al. Con.
Data Trace-1	50.83	0.10	0.00	2.23	0.00
Data Trace-2	38.05	0.51	0.00	2.03	0.00
Data Trace-3	49.72	0.51	0.00	1.72	0.00
Data Trace-4	37.14	0.10	0.00	2.54	0.00
Data Trace-5	34.90	0.41	0.00	2.12	0.00
Data Trace-6	66.66	0.20	0.00	2.03	0.00
Data Trace-7	29.32	0.41	0.00	2.64	0.00
Data Trace-8	25.06	0.20	0.00	1.93	0.00
Data Trace-9	51.14	0.41	0.00	2.09	0.00
Data Trace-10	38.05	0.51	0.00	2.38	0.00
Data Trace-11	47.59	0.41	0.00	1.62	0.00
Data Trace-12	49.51	0.30	0.00	1.52	0.00
Data Trace-13	32.87	0.41	0.00	2.32	0.00
Data Trace-14	36.22	0.20	0.00	1.72	0.00
Data Trace-15	40.28	0.20	0.00	2.71	0.00
Data Trace-16	39.17	0.10	0.00	2.64	0.00
Data Trace-17	42.31	0.20	0.00	1.42	0.00
Data Trace-18	32.77	0.30	0.00	1.83	0.00
Data Trace-19	42.31	0.20	0.00	1.80	0.00
Data Trace-20	34.09	0.10	0.00	1.83	0.00
Data Trace-21	27.50	0.10	0.00	1.42	0.00
Data Trace-22	62.60	0.20	0.00	0.81	0.00
Data Trace-23	43.53	0.41	0.00	2.13	0.00
Data Trace-24	48.20	0.51	0.00	1.59	0.00
Data Trace-25	56.92	0.10	0.00	1.52	0.00
Data Trace-26	35.72	0.20	0.00	2.31	0.00
Data Trace-27	39.17	0.10	0.00	2.21	0.00
Data Trace-28	49.62	0.41	0.00	1.93	0.00
Data Trace-29	37.54	0.51	0.00	2.74	0.00
Data Trace-30	46.06	0.41	0.00	1.32	0.00
Mean	42.16	0.29	0.00	1.97	0.00
Standard Deviation	9.80	0.15	0.00	0.46	0.00
Confidence Interval	42.16 $\pm$4.62	0.29 $\pm$0.07	0.00	1.97 $\pm$0.22	0.00

[6]Note: Perm. Disc.: Permanent Disconnection, P. D. W. Sync: Permanent Disconnection With Synchronization, Int. Disc.: Intermittent Disconnection, I. D. W. Sync: Intermittent Disconnection With Synchronization, Al. Con.: Always Connected

[7]Note: Data presented in the table is percentage of the wasted instructions out of the total instructions.

time before the network disconnection and execution time of the task from initial stage after the network disconnection on the mobile device. The total execution time in case of the permanent disconnection with synchronization is comprised of execution time before the disruption on the cloud server and the execution time after the resumption on the mobile device. The execution time in case of the intermittent disconnection is comprised of time during the connection interval and suspension period due to interaction during the execution in disconnection interval. The intermittent disconnection with synchronization support attribute represents sum of the execution time during the connectivity period and local execution time for the portion of execution during the disconnection period. The synchronization support enables the mobile device to resume the application on the mobile device from last sync point when the disconnection occurs. The always connected attribute represents the only execution time on the cloud server as all the execution is performed on the cloud server.

The data in Table 5.6 shows that the execution time for permanent disconnection is significantly higher than others because of the computation wastage and local execution for longer portion of the task. The always connected execution profile gives the lowest execution time as complete execution is performed on the cloud. Table 5.6 also presents the mean, standard deviation, and confidence interval for each of the disconnection-execution profile.

5.4.2 Computation Wastage

Table 5.7 presents the collected data for computation wastage in different disconnection-execution profiles. The table also contains the computation wastage sample mean, standard deviation, and confidence interval for the permanent disconnection, permanent disconnection with synchronization, intermittent disconnection, intermittent disconnection with synchronization and always connected profiles.

5.5 Data Collection for Analyzing the Impact of Synchronization Interval

We studied the impact of synchronization interval on following parameters: a) execution time, b) number of resynchronizations, c) synchronization overhead, d) computation wastage, e) valuable computation performed on mobile device, f) number of instructions executed between last sync point and disconnection point, and g) energy consumption.

5.5.1 Impact of Synchronization Interval on Execution Time

Table 5.8 shows the collected data for analyzing the impact of synchronization interval on execution time.

The execution time is measured in seconds. The execution time represents the total time taken by cloud server and the mobile device to complete the execution of the task. The columns attribute represent the synchronization interval for which the execution time is computed and the rows represent the mobile user connectivity data traces for which the execution time was studied.

Table 5.8: Impact of Synchronization Interval on Execution Time

Synchronization Interval (seconds)	5	25	45	65	85	105	125	145	165	185	205	225	245
Data Trace-1	1476	1531	1584	1644	1676	1649	1676	1676	1649	1676	1676	1676	1676
Data Trace-2	1459	1531	1606	1613	1662	1662	1662	1647	1662	1662	1662	1662	1662
Data Trace-3	1439	1499	1545	1583	1632	1632	1632	1566	1621	1632	1632	1632	1632
Data Trace-4	1431	1495	1581	1614	1617	1617	1617	1617	1617	1617	1617	1617	1617
Data Trace-5	1430	1522	1533	1610	1544	1610	1610	1610	1610	1610	1610	1610	1610
Data Trace-6	1435	1555	1540	1602	1602	1602	1602	1602	1602	1602	1602	1602	1602
Data Trace-7	1404	1456	1543	1546	1550	1557	1568	1559	1568	1568	1568	1568	1568
Data Trace-8	1395	1495	1557	1512	1522	1566	1566	1566	1566	1552	1566	1566	1566
Data Trace-9	1388	1428	1471	1510	1502	1539	1530	1539	1539	1510	1539	1539	1539
Data Trace-10	1367	1471	1487	1493	1512	1512	1498	1477	1455	1512	1512	1512	1512
Data Trace-11	1360	1480	1486	1487	1493	1498	1480	1498	1485	1498	1498	1498	1491
Data Trace-12	1357	1443	1455	1497	1471	1497	1497	1497	1491	1469	1497	1497	1497
Data Trace-13	1356	1403	1463	1451	1467	1467	1467	1452	1467	1467	1467	1467	1467
Data Trace-14	1355	1394	1479	1455	1486	1465	1486	1496	1496	1496	1496	1496	1496
Data Trace-15	1360	1409	1421	1454	1436	1487	1487	1487	1487	1487	1487	1487	1487
Data Trace-16	1348	1412	1483	1483	1477	1481	1483	1483	1479	1483	1483	1483	1483
Data Trace-17	1344	1442	1461	1450	1423	1483	1471	1450	1428	1483	1483	1483	1483
Data Trace-18	1340	1460	1457	1434	1463	1473	1453	1473	1473	1473	1473	1473	1473
Data Trace-19	1330	1391	1423	1426	1433	1440	1440	1440	1440	1440	1440	1436	1440
Data Trace-20	1320	1396	1378	1404	1438	1438	1438	1438	1438	1422	1438	1438	1438
Data Trace-21	1311	1372	1404	1434	1408	1391	1434	1411	1434	1434	1434	1434	1434
Data Trace-22	1311	1377	1369	1417	1425	1432	1422	1432	1431	1432	1432	1432	1432
Data Trace-23	1314	1392	1402	1399	1428	1422	1428	1418	1423	1428	1428	1428	1428
Data Trace-24	1291	1348	1395	1390	1398	1361	1398	1385	1398	1398	1366	1398	1398
Data Trace-25	1291	1313	1369	1377	1382	1382	1356	1347	1382	1361	1382	1382	1382
Data Trace-26	1288	1326	1357	1378	1378	1345	1379	1379	1379	1359	1379	1379	1379
Data Trace-27	1279	1347	1353	1371	1371	1371	1371	1371	1371	1371	1371	1354	1321
Data Trace-28	1273	1311	1355	1350	1358	1358	1357	1358	1358	1358	1358	1358	1358
Data Trace-29	1271	1318	1319	1338	1347	1358	1343	1357	1358	1358	1358	1358	1358
Data Trace-30	1261	1329	1329	1330	1346	1346	1346	1346	1346	1346	1346	1346	1346
Mean	1353	1422	1453	1468	1475	1481	1483	1479	1482	1483	1487	1487	1486
Standard Deviation	60	72	82	90	93	97	97	94	93	96	96	96	97
Confidence Interval	1353 ± 28	1422 ± 34	1453 ± 39	1468 ± 42	1475 ± 44	1481 ± 46	1483 ± 46	1479 ± 44	1482 ± 44	1483 ± 45	1487 ± 45	1487 ± 45	1486 ± 46

[8]Note: Data presented in the table is in seconds.

The table shows that the execution time increases with the increasing size of synchronization interval. The data traces in the table are sorted in descending order with respect to the total disconnection time. The execution time decreases from top to the bottom of the columns because of shorter disconnection time. The shorter disconnection time helps the mobile device in executing the larger portion of the execution on the cloud server that takes relatively less time to execute the same application.

5.5.2 Impact of Synchronization Interval on Number of Resynchronizations

Table 5.9 shows the data collected for analyzing the impact of synchronization interval on number of resynchronizations performed by the mobile device.

The mobile device performs the resynchronization if the disconnection interval time is greater than the time taken to perform the computation between the last sync point and disconnection point, the state transfer time, suspend and resume time of the application

Table 5.9: Impact of Synchronization Interval on Number of Resynchronizations

Synchronization Interval (seconds)	5	25	45	65	85	105	125	145	165	185	205	225	245
Data Trace-1	8	6	5	1	1	1	1	1	1	1	1	1	1
Data Trace-2	8	7	4	2	0	0	0	1	0	0	0	0	0
Data Trace-3	7	6	4	2	0	0	0	2	1	0	0	0	0
Data Trace-4	8	6	1	1	0	0	0	0	0	0	0	0	0
Data Trace-5	7	6	4	0	4	0	0	0	0	0	0	0	0
Data Trace-6	8	3	1	1	0	0	0	0	0	0	0	0	0
Data Trace-7	8	5	2	1	1	2	0	1	0	0	0	0	0
Data Trace-8	8	5	1	5	2	0	0	0	0	1	0	0	0
Data Trace-9	7	5	3	3	1	0	2	0	0	1	0	0	0
Data Trace-10	7	1	1	1	0	0	1	1	1	0	0	0	0
Data Trace-11	8	3	2	2	1	0	1	0	1	0	0	0	1
Data Trace-12	8	7	4	0	3	0	0	0	2	2	0	0	0
Data Trace-13	7	5	2	2	1	1	1	1	1	1	1	1	1
Data Trace-14	8	5	2	3	1	1	1	0	0	0	0	1	0
Data Trace-15	7	5	4	4	2	0	0	0	0	0	0	0	0
Data Trace-16	8	4	0	0	1	1	0	0	1	0	0	0	0
Data Trace-17	8	2	1	2	1	1	1	1	1	0	0	0	0
Data Trace-18	7	4	1	3	1	0	2	0	0	0	0	0	0
Data Trace-19	7	3	3	2	2	1	1	1	1	1	1	2	1
Data Trace-20	8	4	3	2	1	0	0	0	0	1	0	0	0
Data Trace-21	8	5	3	1	2	1	0	1	0	0	0	0	0
Data Trace-22	8	6	3	1	1	0	1	0	1	0	0	0	0
Data Trace-23	8	4	5	1	0	1	0	1	2	0	0	0	0
Data Trace-24	7	3	1	1	0	1	0	1	0	0	1	0	0
Data Trace-25	6	4	3	2	1	1	2	3	1	1	1	1	1
Data Trace-26	6	5	3	1	1	2	1	0	0	1	0	0	0
Data Trace-27	7	4	4	0	0	0	0	0	0	0	0	3	4
Data Trace-28	7	4	2	1	0	0	1	0	0	0	0	0	0
Data Trace-29	8	6	2	1	2	0	1	1	0	0	0	0	0
Data Trace-30	8	3	1	2	0	0	0	0	0	0	0	0	0
Mean	7.50	4.53	2.50	1.60	1.00	0.47	0.57	0.53	0.47	0.33	0.17	0.30	0.30
Standard Deviation	0.63	1.43	1.36	1.16	0.98	0.63	0.68	0.73	0.63	0.55	0.38	0.70	0.79
Confidence Interval	7.50± 0.30	4.53± 0.67	2.50± 0.64	1.60± 0.55	1.00± 0.46	0.47± 0.30	0.57± 0.32	0.53± 0.34	0.47± 0.30	0.33± 0.26	0.17± 0.18	0.30± 0.33	0.30± 0.37

on the cloud server. The column attribute represents the synchronization interval for which the resynchronization count has been computed and row represents the mobile user connectivity data trace.

5.5.3 Impact of Synchronization Interval on Synchronization Overhead

Table 5.10 shows the collected data for analyzing the impact of synchronization interval on synchronization overhead.

The synchronization overhead is the number of bytes exchanged for performing the synchronization task between the cloud server and the mobile device. The data presented in the table is in kilobytes. The column attribute presents the synchronization interval for which the synchronization overhead is computed and the rows represent the mobile user connectivity data trace. The table shows that synchronization overhead decreases with the increasing synchronization interval. The table also presents the mean, standard deviation, and confidence interval for the collected data of synchronization overhead.

Table 5.10: Impact of Synchronization Interval on Synchronization Overhead

Synchronization Interval (seconds)	5	25	45	65	85	105	125	145	165	185	205	225	245
Data Trace-1	468	90	51	33	21	18	15	12	12	9	9	6	6
Data Trace-2	474	93	48	33	21	15	12	12	9	9	6	6	6
Data Trace-3	477	90	48	30	24	21	15	12	12	9	6	6	6
Data Trace-4	483	96	54	36	27	21	18	15	12	12	9	9	9
Data Trace-5	483	93	54	36	27	21	12	12	9	9	6	6	6
Data Trace-6	486	102	51	36	24	21	18	12	9	9	6	6	6
Data Trace-7	492	99	54	36	27	18	15	12	9	6	6	6	6
Data Trace-8	492	96	51	39	24	15	15	12	12	9	6	6	6
Data Trace-9	498	93	51	36	21	18	15	9	9	9	6	6	6
Data Trace-10	504	99	51	33	21	18	15	12	12	9	6	6	6
Data Trace-11	507	105	51	33	24	18	18	12	12	9	9	9	9
Data Trace-12	507	102	57	30	27	15	12	12	9	9	6	6	3
Data Trace-13	507	99	51	33	18	18	15	12	6	6	6	3	3
Data Trace-14	507	99	54	36	24	18	12	9	6	6	6	6	3
Data Trace-15	510	99	51	36	27	12	9	9	6	6	6	6	3
Data Trace-16	510	99	51	33	24	15	15	12	12	6	6	6	6
Data Trace-17	510	99	51	33	24	18	12	12	9	6	6	6	3
Data Trace-18	510	105	54	33	24	15	15	12	9	9	9	6	6
Data Trace-19	513	99	54	30	21	15	12	9	9	9	6	6	3
Data Trace-20	519	99	51	33	24	15	15	12	12	9	6	6	6
Data Trace-21	522	99	51	33	21	15	12	12	6	6	6	3	3
Data Trace-22	519	102	51	30	24	18	18	12	12	6	6	6	6
Data Trace-23	522	102	54	33	21	18	15	12	9	6	6	6	6
Data Trace-24	528	99	51	36	21	21	18	12	6	6	6	3	3
Data Trace-25	528	102	51	36	21	18	18	12	6	6	3	3	3
Data Trace-26	531	102	54	33	24	21	12	9	6	6	3	3	3
Data Trace-27	534	105	57	33	27	18	12	12	12	9	9	9	9
Data Trace-28	537	102	54	33	21	15	15	12	9	6	3	3	3
Data Trace-29	537	105	51	36	24	18	15	9	6	6	6	3	3
Data Trace-30	540	105	54	36	24	18	9	9	9	6	3	3	3
Mean	509	99	52	34	23	18	14	11	9	8	6	6	5
Standard Deviation	19.88	4.27	2.17	2.25	2.42	2.37	2.58	1.45	2.35	1.71	1.67	1.78	1.98
Confidence Interval	19.88± 9.36	4.27± 2.01	2.17± 1.02	2.25± 1.06	2.42± 1.14	2.37± 1.12	2.58± 1.21	1.45± 0.68	2.35± 1.11	1.71± 0.81	1.67± 0.79	1.78± 0.84	1.98± 0.93

[9]Note: Data presented in the table is in Kilo Byte.

5.5.4 Impact of Synchronization Interval on Valuable Computation Performed by Process State Synchronization Enabled Mobile Device

Table 5.11 presents the collected data for analyzing the impact of synchronization interval on valuable computation performed by process state synchronization enabled mobile device.

The presented data represent the percentage of valuable computation out of the total computation performed on the mobile device in intermittent disconnection profile. The columns show the valuable instructions computed on the mobile device during the disconnection for different values of synchronization interval. The rows present the mobile user connectivity data trace number.

5.5.5 Impact of Synchronization Interval on Cloud Computation Wastage

Table 5.12 presents the data collected for the study of impact of synchronization interval on cloud computation wastage.

Table 5.11: Impact of Synchronization Interval on Valuable Computation Performed by Process State Synchronization Enabled Mobile Device

Synchronization Interval (seconds)	5	25	45	65	85	105	125	145	165	185	205	225	245
Data Trace-1	19.15	13.83	8.85	3.21	0.16	2.70	0.16	0.16	2.70	0.16	0.16	0.16	0.16
Data Trace-2	19.33	12.45	5.27	4.73	0.00	0.00	0.00	1.47	0.00	0.00	0.00	0.00	0.00
Data Trace-3	18.30	12.59	8.22	4.67	0.00	0.00	0.00	6.19	1.05	0.00	0.00	0.00	0.00
Data Trace-4	17.73	11.56	3.37	0.38	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Data Trace-5	17.21	8.43	7.37	0.00	6.35	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Data Trace-6	15.88	4.50	5.92	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Data Trace-7	15.57	10.60	2.38	2.11	1.70	1.10	0.00	0.82	0.00	0.00	0.00	0.00	0.00
Data Trace-8	16.24	6.85	0.96	5.19	4.29	0.00	0.00	0.00	0.00	1.47	0.00	0.00	0.00
Data Trace-9	14.38	10.46	6.40	2.80	3.51	0.00	0.81	0.00	0.00	2.82	0.00	0.00	0.00
Data Trace-10	13.84	3.91	2.38	1.88	0.00	0.00	1.37	3.40	5.43	0.00	0.00	0.00	0.00
Data Trace-11	13.10	1.76	1.18	1.06	0.51	0.00	1.81	0.00	1.31	0.00	0.00	0.00	0.80
Data Trace-12	13.37	5.08	4.03	0.00	2.50	0.00	0.00	0.00	0.67	2.70	0.00	0.00	0.00
Data Trace-13	13.34	8.89	3.14	4.31	2.79	2.79	2.79	4.21	2.79	2.79	2.79	2.79	2.79
Data Trace-14	13.43	9.65	1.64	3.94	0.99	3.02	0.98	0.00	0.00	0.00	0.00	0.03	0.00
Data Trace-15	12.15	7.48	6.34	3.14	4.95	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Data Trace-16	12.87	6.70	0.00	0.00	0.68	0.23	0.00	0.00	0.47	0.00	0.00	0.00	0.00
Data Trace-17	13.28	3.88	2.14	3.16	5.69	0.05	1.13	3.16	5.19	0.00	0.00	0.00	0.00
Data Trace-18	12.73	1.32	1.61	3.77	1.10	0.00	1.97	0.00	0.00	0.00	0.00	0.00	0.00
Data Trace-19	12.72	6.90	3.93	3.58	2.97	2.27	2.27	2.27	2.27	2.27	2.27	2.73	2.27
Data Trace-20	11.27	3.97	5.73	3.28	0.04	0.00	0.00	0.00	0.00	1.56	0.00	0.00	0.00
Data Trace-21	11.73	5.90	2.92	0.02	2.49	4.08	0.00	2.19	0.00	0.00	0.00	0.00	0.00
Data Trace-22	11.55	5.24	6.05	1.48	0.63	0.00	0.98	0.00	0.10	0.00	0.00	0.00	0.00
Data Trace-23	10.93	3.50	2.62	2.90	0.00	0.61	0.00	1.08	0.56	0.00	0.00	0.00	0.00
Data Trace-24	10.31	4.86	0.37	0.81	0.00	3.55	0.00	1.38	0.00	0.00	3.04	0.00	0.00
Data Trace-25	9.52	7.34	2.08	1.28	0.75	0.75	3.31	4.11	0.75	2.78	0.75	0.75	0.75
Data Trace-26	8.70	5.15	2.16	0.14	0.18	3.33	0.04	0.00	0.00	1.97	0.00	0.00	0.00
Data Trace-27	8.80	2.28	1.77	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.62	4.68
Data Trace-28	8.08	4.49	0.33	0.72	0.00	0.00	0.14	0.00	0.00	0.00	0.00	0.00	0.00
Data Trace-29	8.22	3.79	3.70	1.99	1.06	0.00	1.48	0.13	0.00	0.00	0.00	0.00	0.00
Data Trace-30	8.20	1.65	1.70	1.52	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Mean	13.06	6.50	3.49	2.07	1.44	0.82	0.64	1.02	0.78	0.62	0.30	0.27	0.38
Std Deviation	3.28	3.47	2.42	1.68	1.88	1.34	0.96	1.65	1.47	1.07	0.83	0.75	1.05
Confidence Interval	13.06 ±1.54	6.50 ±1.63	3.49 ±1.14	2.07 ±0.79	1.44 ±0.88	0.82 ±0.63	0.64 ±0.45	1.02 ±0.78	0.78 ±0.69	0.62 ±0.50	0.30 ±0.39	0.27 ±0.35	0.38 ±0.49

[10]Note: Data presented in the table is percentage of valuable instructions out of total instructions computed by the mobile device.

The table shows the computation wastage only in case of permanent disconnection profile. In permanent disconnection profile, the computation is only wasted on cloud server. However, in case of intermittent disconnection the computation is only wasted on the mobile device. The presented data represent the percentage of cloud computation wastage out of the total computation performed on the cloud server. The columns attribute represent the synchronization interval for which the cloud computation wastage is computed and the rows represent the mobile user connectivity data traces for which the cloud computation wastage was studied.

5.5.6 Impact of Synchronization Interval on Mobile Device Computation Wastage

Table 5.13 shows the collected data for analyzing the impact of synchronization interval on mobile device computation wastage.

The presented data in the table represent the mobile device computation wastage in

Table 5.12: Impact of Synchronization Interval on Cloud Computation Wastage

Synchronization Interval (seconds)	5	25	45	65	85	105	125	145	165	185	205	225	245
Data Trace-1	0.1	0.1	0.6	4.7	7.7	8.2	0.1	6.7	0.6	13.3	9.2	5.2	1.1
Data Trace-2	0.5	2.5	1.5	5.1	3.6	6.1	12.7	8.6	4.6	0.5	17.2	15.2	13.2
Data Trace-3	0.5	1.5	4.1	3.6	6.6	7.1	11.7	5.6	16.2	12.2	8.1	4.1	24.9
Data Trace-4	0.1	1.6	0.6	4.2	2.6	5.2	11.8	7.7	3.7	18.4	16.3	14.3	12.3
Data Trace-5	0.4	1.9	2.9	1.9	0.4	2.9	9.5	5.5	1.4	16.1	14.1	12.1	10.0
Data Trace-6	0.2	0.7	2.7	0.7	6.3	2.7	3.2	7.8	16.4	10.3	4.3	21.0	16.9
Data Trace-7	0.4	1.4	1.9	2.9	3.4	8.0	4.0	14.6	12.6	10.6	8.5	6.5	4.5
Data Trace-8	0.2	2.2	2.2	5.3	7.8	3.8	12.4	10.3	8.3	6.3	4.3	2.2	0.2
Data Trace-9	0.4	0.4	0.9	5.0	8.0	8.5	0.4	7.0	0.9	13.6	9.5	5.5	1.4
Data Trace-10	0.5	2.5	1.5	5.1	3.6	6.1	12.7	8.6	4.6	0.5	17.2	15.2	13.2
Data Trace-11	0.4	1.9	1.9	1.4	4.5	5.0	9.5	3.4	14.1	10.0	6.0	1.9	22.7
Data Trace-12	0.3	1.3	3.9	3.3	6.4	6.9	11.5	5.4	16.0	12.0	7.9	3.9	24.7
Data Trace-13	0.4	2.4	0.9	6.5	7.0	0.9	7.5	3.4	16.1	14.1	12.1	10.0	8.0
Data Trace-14	0.2	0.7	4.3	3.2	1.7	4.3	10.9	6.8	2.7	17.5	15.4	13.4	11.4
Data Trace-15	0.2	2.2	3.8	0.7	5.8	8.3	2.2	10.9	6.8	2.7	19.5	17.5	15.4
Data Trace-16	0.1	1.1	2.6	6.2	4.7	7.2	1.1	9.7	5.7	1.6	18.4	16.3	14.3
Data Trace-17	0.2	1.7	1.2	2.7	7.8	10.3	4.3	12.9	8.8	4.8	0.7	19.5	17.5
Data Trace-18	0.3	2.3	0.8	6.4	6.9	0.8	7.4	3.3	16.0	14.0	12.0	9.9	7.9
Data Trace-19	0.2	1.7	1.2	2.7	7.8	10.3	4.3	12.9	8.8	4.8	0.7	19.5	17.5
Data Trace-20	0.1	1.1	2.1	1.1	8.2	2.1	8.7	4.7	0.6	15.3	13.3	11.3	9.2
Data Trace-21	0.1	2.1	0.1	1.1	1.6	6.2	2.1	12.8	10.8	8.7	6.7	4.7	2.6
Data Trace-22	0.2	1.7	3.2	3.2	2.2	9.3	11.9	3.8	12.4	6.3	0.2	16.9	12.9
Data Trace-23	0.4	0.4	2.4	4.0	0.4	0.9	5.5	14.1	10.0	6.0	1.9	20.7	18.7
Data Trace-24	0.5	2.5	2.5	2.0	5.1	5.6	10.1	4.1	14.7	10.7	6.6	2.5	23.3
Data Trace-25	0.1	1.1	2.1	4.2	5.2	3.7	6.2	12.8	6.7	0.6	15.3	11.3	7.2
Data Trace-26	0.2	0.2	3.8	2.7	1.2	3.8	10.3	6.3	2.2	16.9	14.9	12.9	10.9
Data Trace-27	0.1	1.1	2.6	6.2	4.7	7.2	1.1	9.7	5.7	1.6	18.4	16.3	14.3
Data Trace-28	0.4	1.4	4.0	3.4	6.5	7.0	11.6	5.5	16.1	12.1	8.0	4.0	24.8
Data Trace-29	0.5	2.0	1.0	4.6	3.0	5.6	12.2	8.1	4.1	18.8	16.7	14.7	12.7
Data Trace-30	0.4	0.4	0.4	6.5	2.9	3.4	8.0	1.9	12.6	8.5	4.5	0.4	21.2
Mean	0.3	1.5	2.1	3.7	4.8	5.6	7.5	7.8	8.7	9.6	10.3	11.0	13.2
Standard Deviation	0.2	0.7	1.2	1.8	2.4	2.7	4.2	3.6	5.5	5.7	5.9	6.3	7.2
Confidence Interval	0.3 ±0.1	1.5 ±0.4	2.1 ±0.6	3.7 ±0.8	4.8 ±1.2	5.6 ±1.3	7.5 ±2.0	7.8 ±1.7	8.7 ±2.6	9.6 ±2.7	10.3 ±2.8	11.0 ±3.0	13.2 ±3.4

[11]Note: Data presented in the table is percentage of wasted instructions out of the total instructions computed on the cloud server.

percentage out of the total computation performed on the mobile device. The computed instructions wastage occurs because the mobile device has to re-execute the already cloud computed instructions between the last sync point and disconnection. The columns attribute represent the synchronization interval for which the mobile device computation wastage is computed and the rows represent the mobile user connectivity data traces for which the mobile device computation wastage was studied.

5.5.7 Impact of Synchronization Interval on Number of Instructions Executed Between Last Sync Point and Disconnection

Table 5.14 presents the collected data for analyzing the impact of synchronization interval on number of instructions executed between last sync point and disconnection in cloud.

The collected data is presented as a computed instructions percentage out of total instructions computed on cloud server. The columns attribute represent the synchroniza-

Table 5.13: Impact of Synchronization Interval on Mobile Device Computation Wastage

Synchronization Interval (seconds)	5	25	45	65	85	105	125	145	165	185	205	225	245
Data Trace-1	2.8	8.2	13.1	18.8	21.8	19.3	21.8	21.8	19.3	21.8	21.8	21.8	21.8
Data Trace-2	2.1	9.0	16.2	16.7	21.5	21.5	21.5	20.0	21.5	21.5	21.5	21.5	21.5
Data Trace-3	2.1	7.8	12.2	15.7	20.4	20.4	20.4	14.2	19.3	20.4	20.4	20.4	20.4
Data Trace-4	2.1	8.3	16.5	19.5	19.9	19.9	19.9	19.9	19.9	19.9	19.9	19.9	19.9
Data Trace-5	2.4	11.2	12.3	19.6	13.3	19.6	19.6	19.6	19.6	19.6	19.6	19.6	19.6
Data Trace-6	3.4	14.8	13.4	19.3	19.3	19.3	19.3	19.3	19.3	19.3	19.3	19.3	19.3
Data Trace-7	2.5	7.5	15.7	16.0	16.4	17.0	18.1	17.3	18.1	18.1	18.1	18.1	18.1
Data Trace-8	1.8	11.2	17.1	12.9	13.8	18.1	18.1	18.1	18.1	16.6	18.1	18.1	18.1
Data Trace-9	2.7	6.6	10.7	14.3	13.6	17.1	16.3	17.1	17.1	14.3	17.1	17.1	17.1
Data Trace-10	2.3	12.2	13.7	14.3	16.1	16.1	14.8	12.7	10.7	16.1	16.1	16.1	16.1
Data Trace-11	2.5	13.9	14.5	14.6	15.1	15.6	13.8	15.6	14.3	15.6	15.6	15.6	14.8
Data Trace-12	2.2	10.5	11.6	15.6	13.1	15.6	15.6	15.6	14.9	12.9	15.6	15.6	15.6
Data Trace-13	2.2	6.7	12.4	11.2	12.8	12.8	12.8	11.3	12.8	12.8	12.8	12.8	12.8
Data Trace-14	2.1	5.9	13.9	11.6	14.6	12.5	14.6	15.6	15.6	15.6	15.6	15.5	15.6
Data Trace-15	3.1	7.8	8.9	12.1	10.3	15.3	15.3	15.3	15.3	15.3	15.3	15.3	15.3
Data Trace-16	2.2	8.4	15.1	15.1	14.4	14.9	15.1	15.1	14.6	15.1	15.1	15.1	15.1
Data Trace-17	1.8	11.2	13.0	11.9	9.4	15.1	14.0	11.9	9.9	15.1	15.1	15.1	15.1
Data Trace-18	2.0	13.4	13.2	11.0	13.7	14.8	12.8	14.8	14.8	14.8	14.8	14.8	14.8
Data Trace-19	1.7	7.5	10.5	10.8	11.5	12.1	12.1	12.1	12.1	12.1	12.1	11.7	12.1
Data Trace-20	2.2	9.5	7.8	10.2	13.5	13.5	13.5	13.5	13.5	11.9	13.5	13.5	13.5
Data Trace-21	1.6	7.5	10.4	13.3	10.9	9.3	13.4	11.2	13.4	13.4	13.4	13.4	13.4
Data Trace-22	1.7	8.0	7.2	11.8	12.6	13.3	12.3	13.3	13.2	13.3	13.3	13.3	13.3
Data Trace-23	2.2	9.7	10.5	10.3	13.2	12.6	13.2	12.1	12.6	13.2	13.2	13.2	13.2
Data Trace-24	1.8	7.2	11.7	11.3	12.1	8.5	12.1	10.7	12.1	12.1	9.1	12.1	12.1
Data Trace-25	2.3	4.5	9.7	10.5	11.0	11.0	8.5	7.7	11.0	9.0	11.0	11.0	11.0
Data Trace-26	2.7	6.3	9.3	11.3	11.2	8.1	11.4	11.4	11.4	9.4	11.4	11.4	11.4
Data Trace-27	2.3	8.8	9.3	11.1	11.1	11.1	11.1	11.1	11.1	11.1	11.1	9.5	6.4
Data Trace-28	2.6	6.2	10.3	9.9	10.7	10.7	10.5	10.7	10.7	10.7	10.7	10.7	10.7
Data Trace-29	2.4	6.9	6.9	8.7	9.6	10.7	9.2	10.5	10.7	10.7	10.7	10.7	10.7
Data Trace-30	2.0	8.6	8.5	8.7	10.2	10.2	10.2	10.2	10.2	10.2	10.2	10.2	10.2
Mean	2.3	8.8	11.9	13.3	13.9	14.5	14.7	14.3	14.6	14.7	15.0	15.1	15.0
Standard Deviation	0.4	2.5	2.8	3.2	3.5	3.8	3.7	3.6	3.5	3.6	3.6	3.5	3.7
Confidence Interval	2.3 ±0.2	8.8 ±1.2	11.9 ±1.3	13.3 ±1.5	13.9 ±1.7	14.5 ±1.8	14.7 ±1.7	14.3 ±1.7	14.6 ±1.6	14.7 ±1.7	15.0 ±1.7	15.1 ±1.7	15.0 ±1.8

[12]Note: Data presented in the table is percentage of wasted instructions out of the total instructions computed on the mobile device.

tion interval for which the percentage instructions executed is computed and the rows represent the mobile user connectivity data traces for which the percentage of executed instructions is measured.

5.6 Data Collected For Performance Comparison of PSS with Optimized VM-based Cloudlet and COMET

This section presents the data collected for verification of PSS through comparison with optimized VM-based cloudlet and COMET.

5.6.1 Execution Time

Table 5.15 presents the data collected for comparing the proposed solution with the state-of-the-art application execution frameworks, optimized VM-based cloudlet and COMET, which provide support for intermediate information exchange.

The column attribute represents the type of approach for execution and row attribute

Table 5.14: Impact of Synchronization Interval on Executed Instructions Between Last Sync Point and Disconnection

Synchronization Interval (seconds)	5	25	45	65	85	105	125	145	165	185	205	225	245
Data Trace-1	2.8	10.2	15.4	26.5	36.0	33.5	36.0	36.0	33.5	36.0	36.0	36.0	36.0
Data Trace-2	2.1	9.5	21.3	25.6	44.0	45.1	45.1	41.7	45.1	44.2	45.1	45.1	45.1
Data Trace-3	2.2	10.1	17.9	25.4	27.9	23.8	32.4	30.3	43.1	44.2	44.9	44.9	44.9
Data Trace-4	2.1	10.5	16.6	21.4	22.9	26.0	24.5	27.0	33.6	25.5	37.8	32.1	26.0
Data Trace-5	2.4	12.9	13.8	21.7	16.5	31.4	46.9	41.7	46.9	44.2	46.9	46.9	46.9
Data Trace-6	3.4	10.5	18.4	21.3	30.4	26.2	22.3	32.9	47.5	44.2	47.5	47.5	47.5
Data Trace-7	2.5	8.1	16.3	20.0	22.6	27.2	35.7	33.7	48.3	48.3	48.3	48.3	48.3
Data Trace-8	1.8	12.5	21.2	15.1	22.6	37.7	35.7	33.7	31.7	29.6	48.3	48.3	48.3
Data Trace-9	2.9	12.1	17.9	21.3	33.4	36.7	37.2	50.4	50.3	44.2	50.4	50.4	50.4
Data Trace-10	2.4	12.8	19.8	22.8	36.4	36.7	36.9	37.9	29.3	44.2	52.6	52.6	50.8
Data Trace-11	2.5	11.0	19.1	24.6	28.5	33.1	24.6	39.7	33.7	35.6	33.6	31.6	26.1
Data Trace-12	2.2	10.8	14.0	28.6	20.5	44.1	42.1	40.1	38.0	36.0	54.7	54.7	54.7
Data Trace-13	2.2	9.2	19.0	22.4	38.0	34.0	29.9	39.1	55.1	55.1	55.1	55.1	55.1
Data Trace-14	2.1	8.3	19.3	23.4	30.5	34.2	43.5	56.1	56.1	56.1	56.1	54.9	56.1
Data Trace-15	3.1	10.6	19.5	20.7	20.5	56.4	56.4	56.4	56.4	56.4	56.4	54.9	56.4
Data Trace-16	2.2	10.6	25.4	30.9	31.6	36.7	37.2	41.8	33.7	57.8	57.8	54.9	50.8
Data Trace-17	1.8	13.4	21.1	22.2	27.3	36.7	45.3	39.7	41.2	57.8	57.8	54.9	57.8
Data Trace-18	2.0	13.4	15.7	20.0	31.7	45.3	37.1	41.7	50.3	44.2	38.2	54.9	50.8
Data Trace-19	1.7	8.9	13.8	26.3	33.3	37.7	46.4	44.3	42.3	40.3	38.3	36.2	59.0
Data Trace-20	2.2	12.9	14.4	24.8	31.7	45.4	37.2	41.8	33.7	40.4	56.4	54.9	50.8
Data Trace-21	1.6	10.5	18.9	26.8	33.8	38.3	46.8	41.8	59.4	59.4	59.0	59.4	59.4
Data Trace-22	1.7	10.1	15.1	28.1	31.8	36.8	24.7	39.9	33.7	60.3	59.0	54.9	50.8
Data Trace-23	2.2	11.7	13.2	25.2	40.4	36.8	48.3	41.8	50.3	62.2	59.0	54.9	50.8
Data Trace-24	1.9	12.4	20.8	21.4	38.3	26.3	24.8	34.6	63.8	63.0	58.4	63.8	63.8
Data Trace-25	2.3	8.2	19.1	18.6	38.8	32.7	24.0	35.1	64.3	62.3	64.3	64.3	64.3
Data Trace-26	2.9	7.8	18.7	32.3	31.0	22.9	40.0	50.5	65.1	63.0	65.1	65.1	65.1
Data Trace-27	2.4	11.7	14.0	27.6	30.1	36.7	47.8	41.8	33.7	44.3	38.3	32.2	23.4
Data Trace-28	2.6	9.7	17.7	26.9	37.9	44.9	37.3	36.8	49.4	63.0	66.1	66.1	66.1
Data Trace-29	2.4	8.2	18.8	21.8	30.9	36.8	36.9	54.1	66.0	63.0	59.0	71.7	71.7
Data Trace-30	2.0	10.0	16.8	20.4	31.7	36.9	61.8	56.5	50.4	57.8	76.5	76.5	75.7
Mean	2.3	10.6	17.8	23.8	31.0	35.9	38.2	41.3	46.2	49.4	52.2	52.3	51.8
Standard Deviation	0.4	1.7	2.8	3.8	6.5	7.4	9.8	7.6	11.2	11.0	10.2	11.3	12.3
Confidence Interval	2.3 ±0.2	10.6 ±0.8	17.8 ±1.3	23.8 ±1.8	31.0 ±3.0	35.9 ±3.5	38.2 ±4.6	41.3 ±3.6	46.2 ±5.3	49.4 ±5.2	52.2 ±4.8	52.3 ±5.3	51.8 ±5.8

[13]Note: Data presented in the table is percentage of instructions computed between last sync point and disconnection.

represents the mobile user connectivity data traces for which the execution time is collected. The average execution time of PSS-based execution is less than the time of optimized VM-based cloudlet and COMET for all data traces.

5.6.2 Cloud Computation Wasted

Table 5.16 presents the collected data for comparing the computation wastage in optimized VM-based cloudlet, COMET, and PSS-based execution.

The presented data in the table represent the percentage of computation wasted out of the total computation performed. The column attribute of the table represents the specific solution whereas the rows represent the data traces for which the execution time is measured.

Table 5.15: Execution Time in Optimized VM-based Cloudlet, COMET and PSS-based Execution

	Optimized VM-based Cloudlet	PSS-based execution	COMET
Data Trace-1	2109.8	1475.7	1928.5
Data Trace-2	1974.9	1458.6	2153.2
Data Trace-3	1498.8	1439.3	1948.1
Data Trace-4	1965.3	1430.8	2169.3
Data Trace-5	1450.9	1429.7	2208.5
Data Trace-6	2276.7	1435.1	1650.2
Data Trace-7	1470.6	1404.0	2306.6
Data Trace-8	1485.6	1395.5	2381.6
Data Trace-9	1393.8	1388.0	1923.1
Data Trace-10	1974.9	1366.6	2153.2
Data Trace-11	1406.3	1360.2	1985.6
Data Trace-12	2095.9	1356.9	1951.7
Data Trace-13	1458.1	1355.9	2244.2
Data Trace-14	1446.3	1354.8	2185.3
Data Trace-15	1432.0	1360.2	2114.0
Data Trace-16	1436.0	1348.4	2133.6
Data Trace-17	1424.9	1344.1	2078.3
Data Trace-18	1919.3	1339.8	2246.0
Data Trace-19	1424.9	1330.2	2078.3
Data Trace-20	1933.2	1319.5	2222.8
Data Trace-21	1477.0	1310.9	2338.7
Data Trace-22	1353.5	1310.9	1721.6
Data Trace-23	1420.6	1314.2	2056.9
Data Trace-24	1404.2	1290.6	1974.9
Data Trace-25	1373.5	1290.6	1821.5
Data Trace-26	1448.1	1288.5	2194.3
Data Trace-27	1436.0	1278.8	2133.6
Data Trace-28	2096.9	1273.5	1949.9
Data Trace-29	1969.6	1271.4	2162.2
Data Trace-30	1411.7	1260.0	2012.3
Mean	1632.3	1352.8	2080.9
Standard Deviation	295.3	60.0	172.3
Confidence Interval	1632.3 $\pm$139.1	1352.8 $\pm$28.3	2080.9 $\pm$81.1

[14]Note: Data presented in the table is in seconds.

5.6.3 Energy Consumption

Table 5.17 shows the data collected for comparing energy consumption of the proposed solution with the state-of-the-art application execution frameworks that provide support for the intermediate process states exchange with the mobile device.

The energy consumption is measured in Joules. The column attribute of the table represents the type of the execution framework and row attribute represents the mobile user connectivity data traces for which the energy consumption is measured.

5.7 Conclusion

The performance of proposed solution is evaluated by implementing and testing in the emulated mobile cloud lab environment. The benchmarking is done by evaluating the prototype application on the emulated mobile device and the cloud server. Data are collected by sampling the evaluation parameters for 30 different mobile user connectivity data traces for intra-city metro transit. The point estimator for each experiment is mea-

Table 5.16: Comparison of Computation Wastage in Optimized VM-based Cloudlet, COMET and PSS-based Execution

	Optimized VM-based Cloudlet	PSS-based execution	COMET
Data Trace-1	50.84	2.84	0.61
Data Trace-2	38.05	2.13	0.00
Data Trace-3	0.00	1.83	1.01
Data Trace-4	37.14	2.13	0.61
Data Trace-5	0.00	2.23	1.42
Data Trace-6	66.66	3.45	1.22
Data Trace-7	0.00	2.54	0.41
Data Trace-8	0.00	1.83	0.71
Data Trace-9	0.00	2.44	0.91
Data Trace-10	38.05	2.03	0.00
Data Trace-11	0.00	2.54	0.41
Data Trace-12	49.52	2.23	0.81
Data Trace-13	0.00	1.72	0.91
Data Trace-14	0.00	2.13	1.22
Data Trace-15	0.00	2.84	0.71
Data Trace-16	0.00	2.23	1.12
Data Trace-17	0.00	1.83	1.22
Data Trace-18	32.77	1.72	0.81
Data Trace-19	0.00	1.32	1.22
Data Trace-20	34.09	2.23	0.61
Data Trace-21	0.00	1.62	0.10
Data Trace-22	0.00	1.72	0.20
Data Trace-23	0.00	2.23	0.91
Data Trace-24	0.00	1.52	1.01
Data Trace-25	0.00	1.62	0.61
Data Trace-26	0.00	2.03	0.71
Data Trace-27	0.00	1.93	1.12
Data Trace-28	49.62	2.23	0.91
Data Trace-29	37.54	2.44	1.01
Data Trace-30	0.00	2.03	0.41
Mean	14.48	2.12	0.76
Standard Deviation	21.64	0.44	0.38
Confidence Interval	14.48 $\pm$10.19	2.12 $\pm$0.21	0.76 $\pm$0.18

[15]Note: Data presented in the table is percentage of instructions wasted out of the total number of instructions.

sured by finding the sample mean of the sample space of 30 values for same experiment that is signified by computing the interval estimate with 99% confidence.

It is concluded that PSS-based execution successfully reduces the execution time and synchronization overhead of an application in dynamic mobile user connectivity conditions of MCC environment. The PSS-based execution synchronizes the intermediate states with the mobile device. On the network disconnection that is mainly due to user mobility, the mobile device is able to resumes the application from the point of last synchronization. When the mobile device detected the connection re-establishment, the mobile device resync back the mobile device executed process states with the cloud server. The cloud server resumes the application execution from the resync point and on the completion of execution sent back the results to the mobile device. Synchronizing the process states enables the process resumption on the mobile device where either the portion of execution or complete execution is performed, thereby minimizing the execution

Table 5.17: Comparison of Energy Consumption in Optimized VM-based Cloudlet, COMET and Process State Synchronization Algorithm

	Optimized VM-based Cloudlet	PSS-based Execution	COMET
Data Trace-1	26.67	35645.91	72020.80
Data Trace-2	296.00	44852.60	72031.77
Data Trace-3	296.00	36449.14	72019.64
Data Trace-4	296.00	45508.62	72029.80
Data Trace-5	296.00	47115.07	72025.62
Data Trace-6	46.67	24250.36	72032.18
Data Trace-7	296.00	51134.40	72015.20
Data Trace-8	296.00	54203.29	72024.01
Data Trace-9	296.00	35428.31	72018.96
Data Trace-10	296.00	44852.60	72039.54
Data Trace-11	296.00	37985.19	72029.13
Data Trace-12	41.67	36596.34	72031.89
Data Trace-13	296.00	48577.52	72019.71
Data Trace-14	296.00	46167.84	72022.66
Data Trace-15	296.00	43246.15	72023.64
Data Trace-16	296.00	44049.38	72030.49
Data Trace-17	296.00	41783.71	72023.61
Data Trace-18	31.67	48651.12	72028.51
Data Trace-19	296.00	41783.71	72014.74
Data Trace-20	41.67	47700.69	72023.23
Data Trace-21	296.00	52449.64	72020.88
Data Trace-22	296.00	27172.05	72017.11
Data Trace-23	296.00	40906.88	72027.54
Data Trace-24	296.00	37546.77	72026.41
Data Trace-25	296.00	31261.78	72025.17
Data Trace-26	296.00	46532.65	72018.33
Data Trace-27	296.00	44049.38	72030.64
Data Trace-28	21.67	36522.74	72026.75
Data Trace-29	36.67	45217.41	72023.50
Data Trace-30	296.00	39079.62	72025.11
Mean	235.16	41890.70	72024.89
Standard Deviation	112.25	7055.76	5.67
Confidence Interval	235.16 $\pm$52.87	41890.70 $\pm$3323.56	72024.89 $\pm$2.67

[16]Note: Data presented in the table is in Joules.

time. The implementation of the PSS-based execution on the emulated mobile device and on the cloud server indicates the viability of PSS-based execution for minimizing the execution time, computation wastage and energy consumption in the MCC environment. Benchmarking of the prototype application with the different values of proposed system variables validates the performance gains of the PSS-based execution for cloud-based interactive mobile applications.

This chapter evaluates the performance of PSS-based execution and gives the insights on performance of the algorithm by varying the parameters values. The chapter discusses the analysis of the collected results presented in Chapter 5 for signifying the usefulness of the proposed PSS algorithm. The chapter also focuses on verification of the proposed solution by comparing PSS-based execution results with the execution of COMET and optimized VM-based cloudlet.

The rest of the chapter is organized into five sections. Section 6.1 discusses the validation of the mathematical model by comparing the results of PSS-based execution obtained through mathematical model with the results obtained in emulated MCC environment. Section 6.2 presents the comparison of cloud-based mobile application execution in different connection profiles. Section 6.3 investigates the performance of PSS-based execution in MCC environment. Section 6.4 compares the performance of PSS-based execution with the performance of COMET and optimized VM-based cloudlet. Section 6.5 concludes the chapter by highlighting the significance of PSS-based execution. The usefulness of the proposed solution is signified by analyzing the experimental results collected in different scenarios.

6.1 Model Validation

The correctness of developed mathematical model is validated by comparing the results obtained from mathematical model with that of experiments. The execution time and mobile device useful and useless computation difference are the parameters which we have studied for model validation.

6.1.1 Execution Time

Figure 6.1 presents the comparison of execution time computed through mathematical model and experiments. The y-axis shows the execution time that is measured in

seconds whereas x-axis represents the execution type for three different data traces. The graph shows that the results of mathematical model for all three data traces are closer to the results obtained from the experiments. The difference in execution time for mathematical model and experimental case is 3.7%, 1.2%, and 3.3% for data trace 1, data trace 2, and data trace 3. This small amount of difference validates the model results with that collected from experiments.

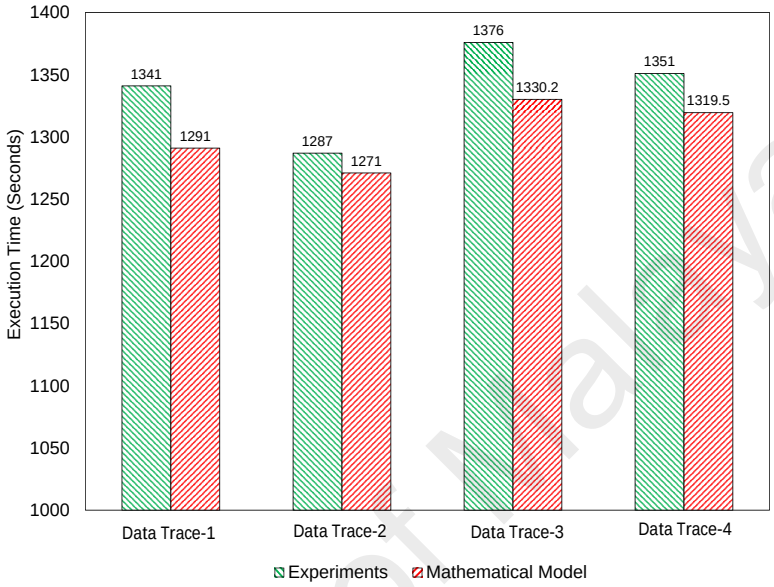


Figure 6.1: Comparison of Execution Time Empirical Results with Mathematical Model Execution Time

6.1.2 Mobile Device Useful and Useless Computation Difference

We verified the upper-bound on synchronization interval where time spent by the mobile device for useful and wasteful computations becomes equal. Ideally, the difference between useful and wasteful computations at this point should be zero. To verify the analytically computed upper bound on synchronization interval, we used the computed intervals in prototype implementation for the respective traces and computed the difference between useful and wasteful mobile computations. Five different randomly generated traces were used and the results were averaged. Empirical results are shown in Figure 6.2 where percentage against total mobile computations of absolute difference between useful and wasteful computations has been plotted against respective analytically computed upper-bound on synchronization intervals. The minimum and maximum

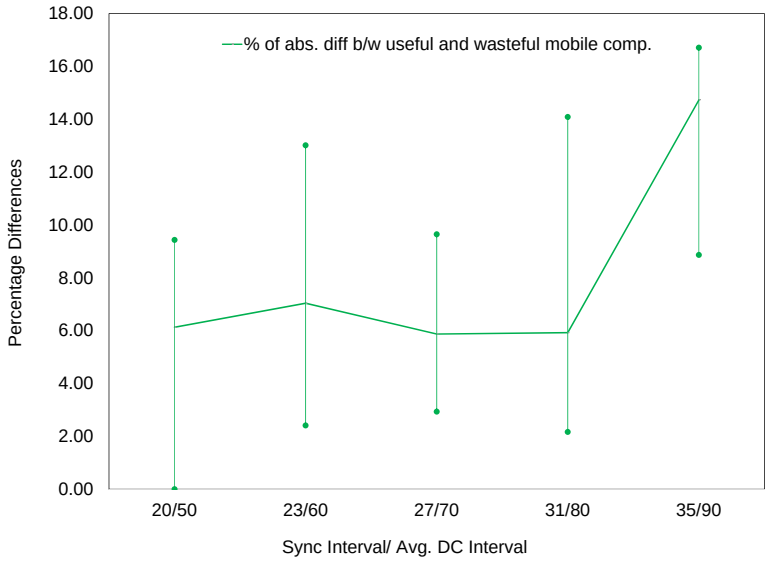


Figure 6.2: Useful and Useless Mobile Computation Absolute Difference for Different Data Traces

percentage difference has also been plotted as error bars. The graph shows that for all disconnection intervals except 90 seconds, the difference percentage is around 6%. This shows very good estimate of the upper-bound, keeping in view the fact that the traces have been generated randomly. In case of disconnection interval of 90 seconds, the value is 14%, which is acceptable but a little higher than 90% accuracy margin. The worst case difference between useful and wasteful computations on the mobile device in individual traces is 17% while the best case is 0%. The acceptable deviation from ideal values validates the analytically computed value of upper-bound on synchronization intervals.

6.2 Comparison of Mobile Application Execution in Different Connection Profiles

In this section, we compare the performance of a mobile application execution in different connection profiles. The mobile application execution performance is compared on the basis of execution time and computation wastage in different connection profiles. We have conducted a comparative study of five different scenarios: a) permanent disconnection, b) permanent disconnection with process state synchronization, c) intermittent disconnection, d) intermittent disconnection with process state synchronization, and e) always connected.

6.2.1 Execution Time

Figure 6.3 presents the comparison of execution time in different connection profiles. The execution time is measured in seconds. The x-axis represents the different connection profiles and y-axis represents the execution time. The execution time in case of permanent

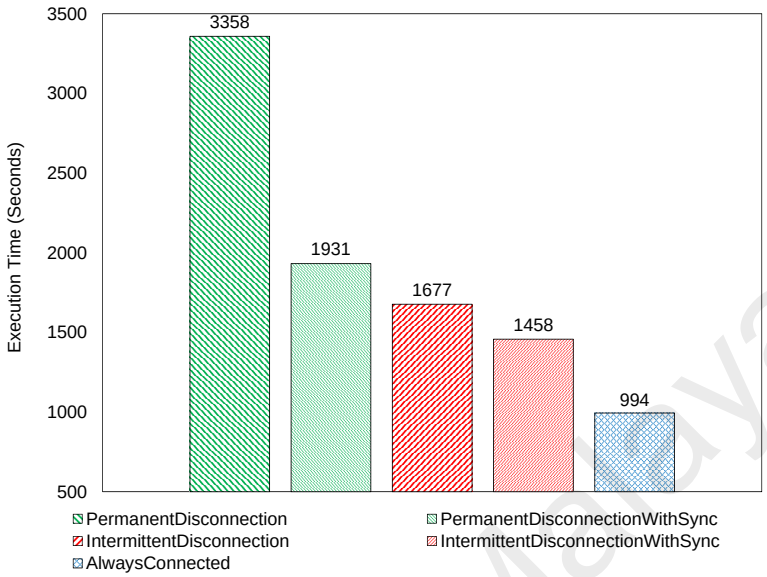


Figure 6.3: Execution Time in Different Connection Profiles

disconnection profile is the largest with a value of 3358 seconds. The execution time in case of permanent disconnection profile is constituent of the execution time in the cloud before the disconnection and the execution time of the whole task executed from the initial stage on the mobile device after the permanent network disconnection. However, when the process state synchronization support is integrated, the execution time is reduced 42% of the execution time of permanent disconnection without synchronization support. The execution time for intermittent disconnection is 50% lesser than the execution time in case of permanent disconnection. The decrease in the execution time is due to resumption of the execution after the disconnections and completion of whole execution on the cloud server. The execution time in case of intermittent disconnection profiles is further decreased 13% when the process state synchronization support is integrated with it. The graph shows the least execution time in case of always connected profile. The comparison shows that the process state synchronization significantly reduced the execution time in permanent and intermittent disconnection profiles.

6.2.2 Computation Wastage

Figure 6.4 presents the comparison of number of total computed instructions wasted for different connection profiles. The y-axis shows the computation wastage in terms of number of instructions and x-axis represents the different connection profiles.

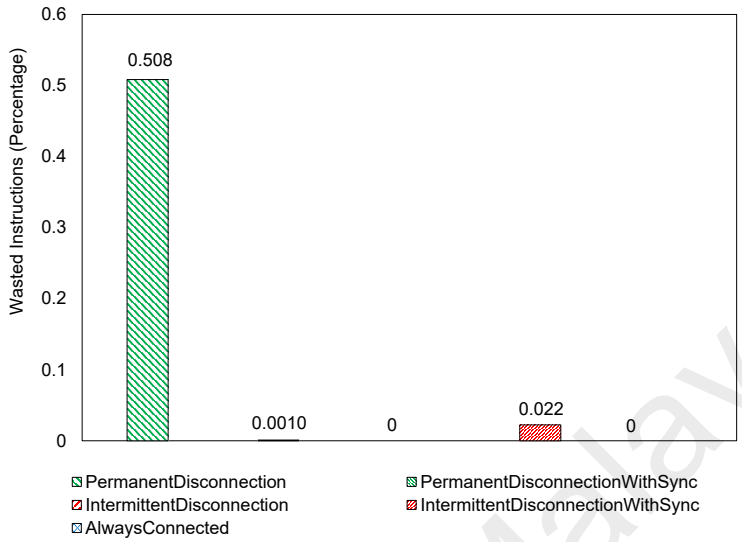


Figure 6.4: Comparison of Computation Wastage With Different Connection Profiles

The graph shows that the number of instructions wasted in case of permanent disconnection profile is the highest. The number of instructions wasted are reduced 99.8% of the permanent disconnection profile wastage when the process state synchronization is integrated. The computation wastage for intermittent disconnection and always connected profiles is zero. However, in case of intermittent disconnections the execution time will be higher for interactive applications when the process states are not synchronized with the mobile device. However, synchronizing the process states with mobile device in intermittent disconnection profiles results in 5% instructions wastage of the total instructions wastage in permanent disconnection without synchronization profile. However, unlike the intermittent disconnection without synchronization support, synchronizing the process states with the mobile device in intermittent disconnection profile reduces the application execution time by enabling the mobile device to resume and execute the application locally during the disconnection period.

6.3 Performance Analysis of Process State Synchronization Algorithm

In this section, we analyze the performance of PSS-based execution in MCC environment. The analysis studies the impact of synchronization interval on following parameters: a) application execution time, b) synchronization overhead, c) mobile device computation wastage, d) cloud computation wastage, e) valuable computation performed by mobile device, f) number of resynchronization, g) number of instructions executed between last sync point and disconnection and h) energy consumption. Apart from these parameters, we have also studied the valuable computation performed by mobile device in different data traces.

6.3.1 Impact of Synchronization Interval

In this subsection, we present the impact of synchronization interval on different performance measuring parameters.

6.3.1.1 Execution Time

Figure 6.5 presents the impact of synchronization interval on execution time for one data trace. The x-axis shows the synchronization interval whereas y-axis represents the execution time. Both parameters are measured in seconds. The execution time of the application is smaller when the synchronization interval is kept smaller. For the higher values of the synchronization interval, the execution time also increases significantly. In general, the increase in execution time with the increasing size of synchronization interval is exponential. However, the data observed in case of the synchronization interval of 105 and 165 show that the execution time is not only dependent on the synchronization interval but also on the disconnection period occurrence with respect to synchronization interval. If the synchronization interval occurs immediately before the disconnection point, the execution time wasted between the last sync point and the disconnection point will be smaller otherwise the execution time wastage will be higher. The higher execution time wastage will increase the overall execution time.

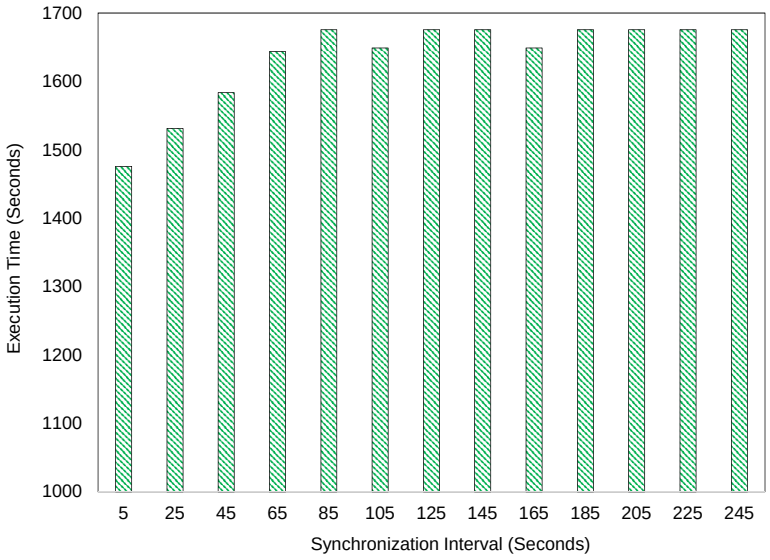


Figure 6.5: Impact of Synchronization Interval on Execution Time

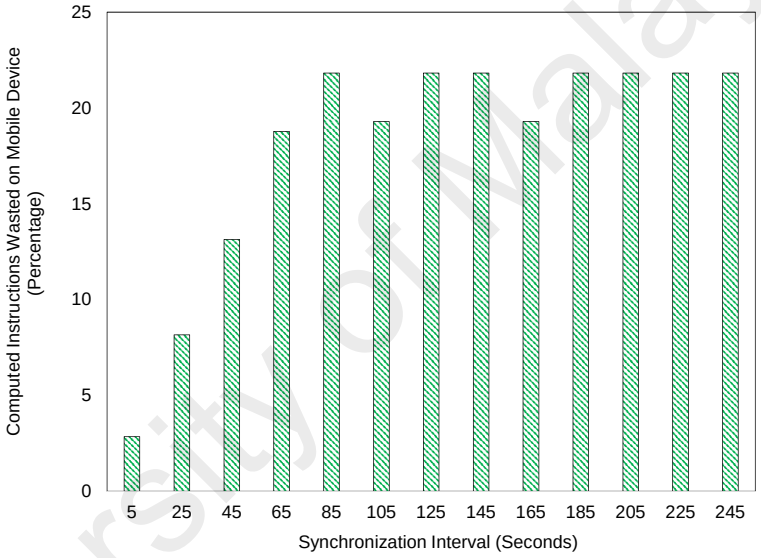


Figure 6.6: Impact of Synchronization Interval on Mobile Device Computation Wastage

6.3.1.2 Computation Wastage

In this subsection, we discuss the impact of synchronization interval on computation wastage. The computation wastage is measured as a percentage of computed instructions wasted on the computing device when the execution is migrated to the another computing device.

Mobile Device Computation Wastage: Figure 6.6 presents the impact of synchronization interval on the mobile device computation wastage in scenario of intermittent disconnection. In case of intermittent disconnection, the computation is not wasted on

the cloud server as the mobile device always connects back with the cloud server. The graph has synchronization interval and wasted instructions percentage of mobile device on x- and y-axis, respectively.

The mobile device computation wastage occurs because of repeatedly performing the computations from the last sync point to the disconnection time on the mobile device. The graph shows that for smaller values of synchronization interval, the computation wastage is relatively less on the mobile device. The reason is that the mobile device gets the larger time during the disconnection period than the time taken for the instructions computed on the cloud between the last synchronization point and disconnection point.

However, when the synchronization interval increases, the cloud has already computed large set of instructions between the last synchronization point and the disconnection point, therefore the mobile device does not get greater time of disconnection to perform that execution. As a result, mobile device does not resynchronize back the process states with the cloud. Hence, the locally computed instructions are wasted on the connection re-establishment. However, the values of synchronization interval greater than 85 seconds do not affect the computation wastage of the mobile device. The exceptional drop of computation wastage in case of 105 and 165 shows that computation wastage does not only depends on synchronization interval but also on the disconnection intervals that follow the exponentially distribution.

Cloud Computation Wastage: Figure 6.7 presents the impact of the synchronization interval on the computation wasted in cloud in case of permanent disconnection. In case of permanent disconnection, the computation is not wasted on the mobile device as the rest of the execution performed by the mobile device. The x-axis of the graph represents the synchronization interval in seconds whereas the y-axis of the graph shows the computed instructions wastage in percentage on the cloud. The computation wastage increases for initial values of synchronization interval, however, the computation wastage is reduced for larger values of synchronization interval as shown in Figure 6.7 in case of greater than 185. The computation wastage on the cloud depends not only on the syn-

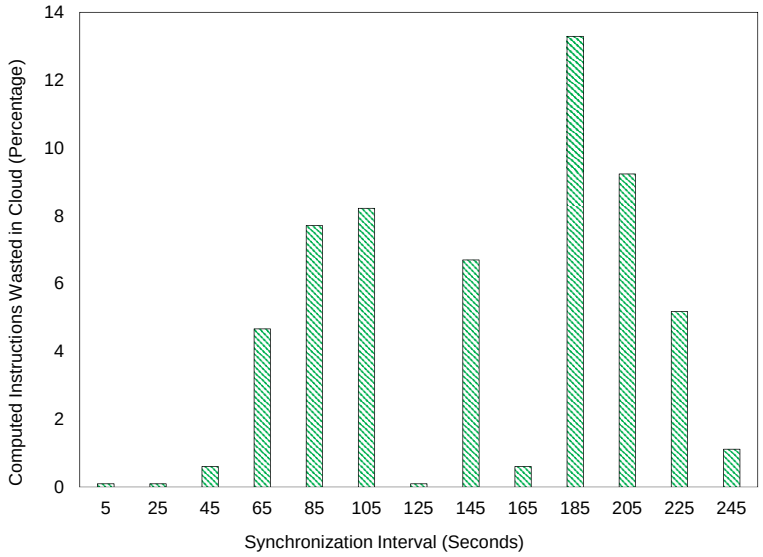


Figure 6.7: Impact of Synchronization Interval on Cloud Computation Wastage

chronization interval but also on the pattern of disconnection. This insight is gained from the lower percentage values of wasted instructions on synchronization interval of 125 seconds and 165 seconds. The lower computation wastage is because of the occurrence of disconnection just immediately after the synchronization time. Hence, less instructions are computed between this interval.

6.3.1.3 Valuable Computation Performed on Mobile Device

Figure 6.8 presents the impact of synchronization interval on the valuable computation performed on the mobile device. The x-axis and y-axis of the graph represent the synchronization interval and the percentage of valuable instructions computed on the mobile device respectively.

In case of the lower values of synchronization interval, the number of valuable instructions computed on the mobile device is higher. However, in case of higher values of synchronization interval such as greater than and equal to 65 seconds, the mobile device computes the small amount of valuable instructions as in most of the cases the mobile device decides not to resynchronize back with the cloud server as the cloud has already executed the instructions. Hence, a large set of computation performed on the mobile device got wasted when the synchronization interval is higher.

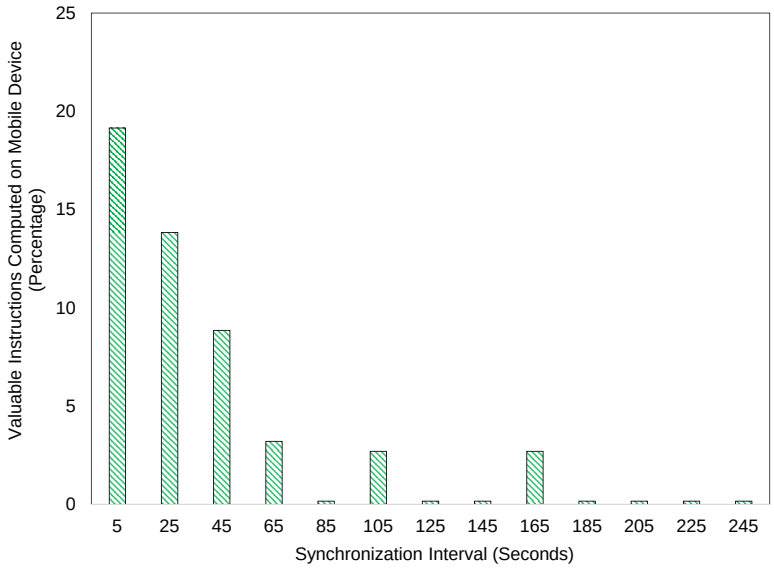


Figure 6.8: Impact of Synchronization Interval on Valuable Computation Performed on Mobile Device

6.3.1.4 Number of Resynchronizations

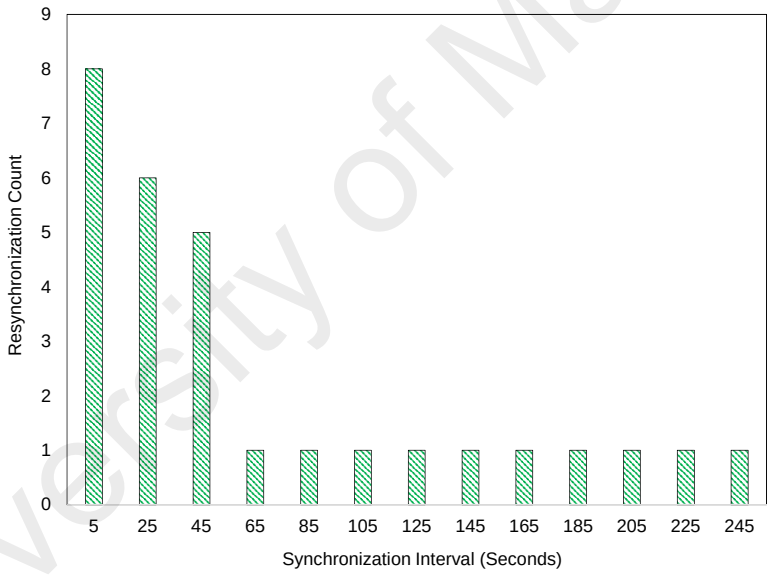


Figure 6.9: Impact of Synchronization Interval on Number of Resynchronizations

Figure 6.9 shows the impact of synchronization interval on resynchronization count. The x-axis and y-axis represent the synchronization interval and resynchronization count, respectively. For smaller sizes of synchronization interval, most of time the mobile device resynchronizes back the process states with the cloud server. Initially, the number of resynchronization decreases with the increasing size of synchronization interval. However, it becomes constant for the size of synchronization interval 65 seconds and onward. For higher synchronization interval, the mobile device may not get enough time, due to

relatively shorter length of disconnection, to compute the more instructions which have already been computed by the cloud between the last synchronization point and disconnection point. Hence, the mobile device does not resynchronize back its executed process states with the cloud server.

6.3.1.5 Instructions Executed Between Last Sync Point and Disconnection

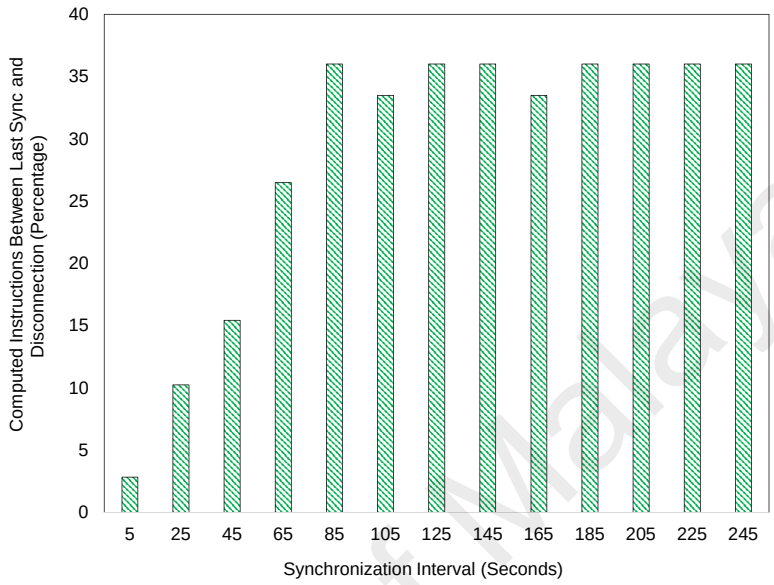


Figure 6.10: Impact of Synchronization Interval on Instructions Executed Between Last Sync Point and Disconnection

Figure 6.10 shows the impact of synchronization interval on the instructions executed between the last sync point and disconnection. The percentage of instructions executed between last sync point and disconnection is increasing linearly with the increasing size of synchronization interval. However, the percentage of instructions executed between last sync point and disconnection becomes constant for the size of synchronization interval 85 seconds and onward with exceptions of 105 second and 165 second. The exception in constant trend of computed instructions shows that the percentage of computed instructions between last sync point and disconnection is not only dependent on the synchronization interval but also on the time of the disconnection.

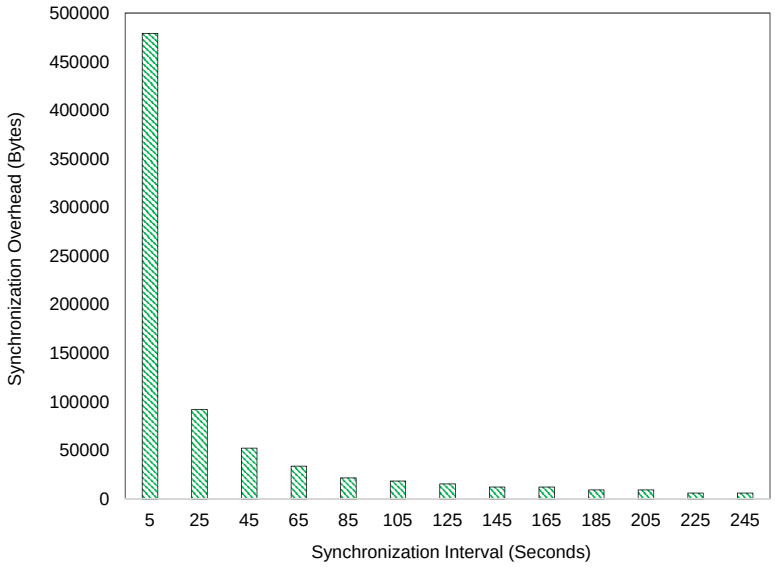


Figure 6.11: Impact of Synchronization Interval on Synchronization Overhead

6.3.1.6 Synchronization Overhead

Figure 6.11 presents the impact of synchronization interval on synchronization overhead for one data trace. The x-axis and y-axis of the graph represent the synchronization interval and synchronization overhead respectively. The graph shows that the synchronization overhead is higher for small size synchronization interval such as in case of synchronization interval of 5 seconds. The higher synchronization overhead for small size synchronization interval is because of more frequent exchange of process states with the cloud server. The higher synchronization interval value results in low frequency of process states exchange that decreases the synchronization overhead.

6.3.1.7 Energy Consumption

Figure 6.12 shows the impact of synchronization interval on the energy consumption of the mobile device.

The x-axis and y-axis represent the synchronization interval and energy consumption respectively. Similar to synchronization overhead, the energy consumption is also higher for small size synchronization interval such as synchronization interval of size 5 seconds. The graph also shows that the energy consumption also decreases with the increased size of the synchronization interval. The higher energy consumption for small size synchro-

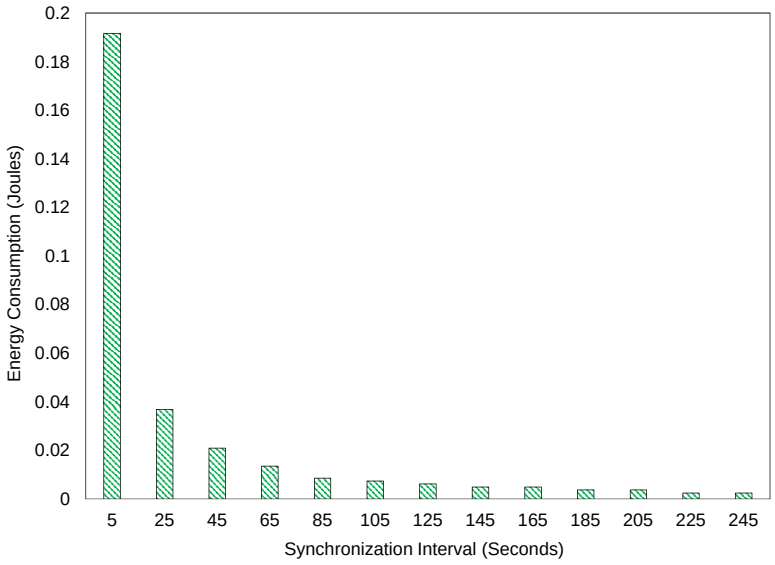


Figure 6.12: Impact of Synchronization on Energy Consumption

nization interval is because of more frequent exchange of process states from the cloud server with the mobile device.

6.3.2 Synchronization vs. Non synchronization-based Execution

Figure 6.13 presents the comparison of execution time for different traces in case of synchronization-based and non-synchronization based mobile application execution in the cloud. The x- and y-axis represents the data traces and execution time. It can be observed that synchronization-based cloud execution of mobile application has reduced the execution time in all cases. The minimum reduction is 2.8% while maximum reduction in execution time is 10.7%. The reduction in execution time for case of synchronization-based execution is due to the resumption of the application from the last sync point in place of initial stage. Another important reason is the resynchronization of the process states with the cloud server on the network connection reestablishment.

6.3.3 Valuable Computation on Mobile Device

Our proposed solution, process state synchronization algorithm for mobility support, has enabled the mobile device to resume the application locally during the disconnection. We have studied the amount of instructions a mobile device can execute during the disconnection period. The valuable instructions computed on mobile device are those

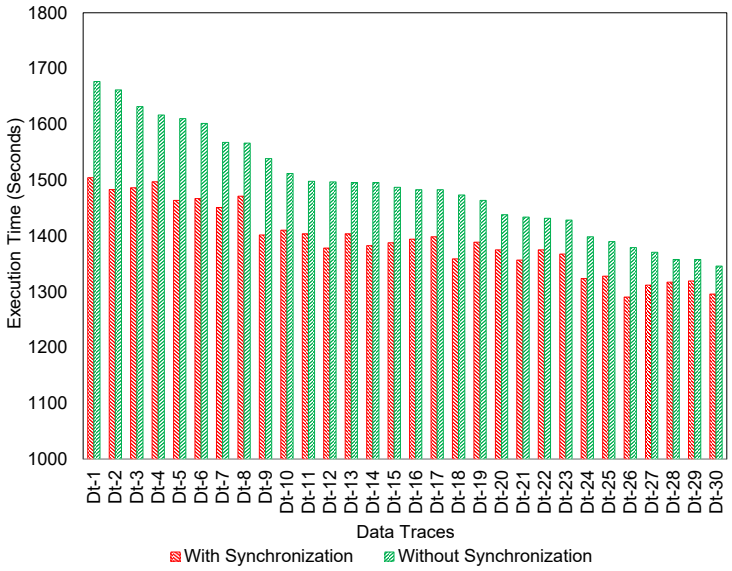


Figure 6.13: Execution Time in Synchronization and Non-Synchronization

instructions which can be resynchronized with the cloud server when the connection is re-established. The mobile device only synchronizes back the computed instructions if the disconnection period is greater than the time taken to compute the instructions from the last sync point to the disconnection point and time taken to suspend, transfer and resume on the cloud server. Figure 6.14 presents the valuable instructions computed on mobile

Figure 6.14: Valuable Instructions Percentage Computed on Mobile Device

device for our proposed solution during the disconnection period. The data traces have varied disconnection time and the data traces are sorted in ascending order with respect to disconnection time. The graph shows the insight that valuable instructions computed on

mobile device are higher for a data traces with larger disconnection time. However, the valuable instructions computed on mobile device is lower for the data traces with smaller disconnection time. The graph also shows that always significant amount of valuable computation is performed by the mobile device for each data trace case.

6.4 Comparison of PSS-based Execution with COMET and Optimized VM-based Execution

In this section, we compare the performance of PSS-based execution with COMET and optimized VM-based execution. The parameters used for the comparison of PSS-based execution are execution time, computation wasted and energy consumption.

6.4.1 Execution Time

Figure 6.15 presents the comparison of PSS-based execution, with the optimized VM-based cloudlet (Ha et al., 2013) and COMET (Gordon et al., 2012). Herein, we present the results for three different data traces. The execution time for all the 3 data traces is least in case of PSS-based execution. In case of first data trace, the PSS-based execution takes 1475.7 seconds, whereas execution time for optimized VM-based cloudlet and COMET is 42.96% and 30.68% greater than that of PSS-based execution. The increased execution time of optimized VM-based cloudlet is because the mobile device is unable to get back the VM-overlay from the previously visited cloudlet before the network disconnection. The mobile device has to re-offload the execution from initial stage on the new cloudlet.

In case of second data trace, the execution time for PSS-based execution is 1339.8 seconds. However, the execution time for optimized VM-based cloudlet and COMET is 43.25% and 67.64% greater than that of the PSS-based execution. The increase in COMET time as compared to the execution time of optimized VM-based cloudlet is because of the earlier occurrence of disconnection time than that of the first data trace. Hence, COMET has to execute more remaining part of the task on the mobile device and in case of optimized VM-based cloudlet less amount of the execution is wasted due to

failure of getting back the VM-overlay from the previously visited cloudlet.

In case of third data trace, the mobile device is able to get back the results from the first cloudlet, as the mobile device got enough time between the movement starting point and the complete disconnection. The time between the movement started and the complete disconnection is generated as exponential distribution with a mean value of 12 that was computed from the collected data traces. In this case, the execution time for PSS-based execution is 1260 seconds, whereas in case of optimized VM-based cloudlet and COMET, the increase of 12.03% and 59.71% has been observed in the execution time. The execution time for optimized VM-based cloudlet is smaller than COMET as the mobile device has got back the intermediate states in the form of VM-overlay from the previously visited cloudlet and the VM-overlay has transferred to another cloudlet for execution. The newly visited cloudlet derived the launch VM from the overlay VM and resumed the execution. For all data traces, PSS-based execution takes least time, the reason for the least execution time is synchronization of states with the mobile device, resumption of the execution on the mobile device during the disconnection period, and resynchronizing the process states with the newly discovered cloudlet. These three characteristics of PSS-based execution contributes in the least execution time.

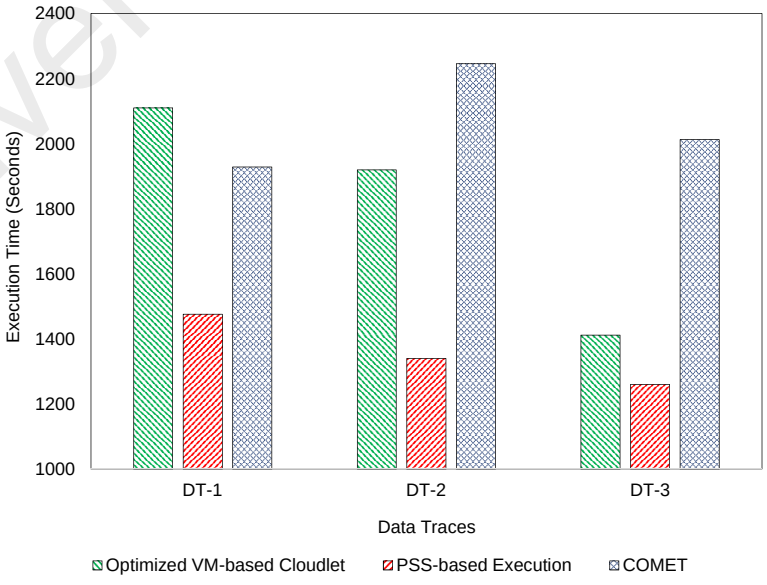


Figure 6.15: Execution Time Comparison With State-of-the-art Application Execution Frameworks

6.4.2 Computation Wasted

Figure 6.16 presents the comparison of PSS-based execution with the optimized VM-based cloudlet and COMET in terms of computation wastage because of the network disconnection. In case of first and second data trace, the computation wastage for optimized VM-based cloudlet approach is significantly higher than that in case of PSS-based execution and COMET. The increased computation wastage is because of the failure of the mobile device to get back the VM-overlay from the previously visited cloudlet. Hence,

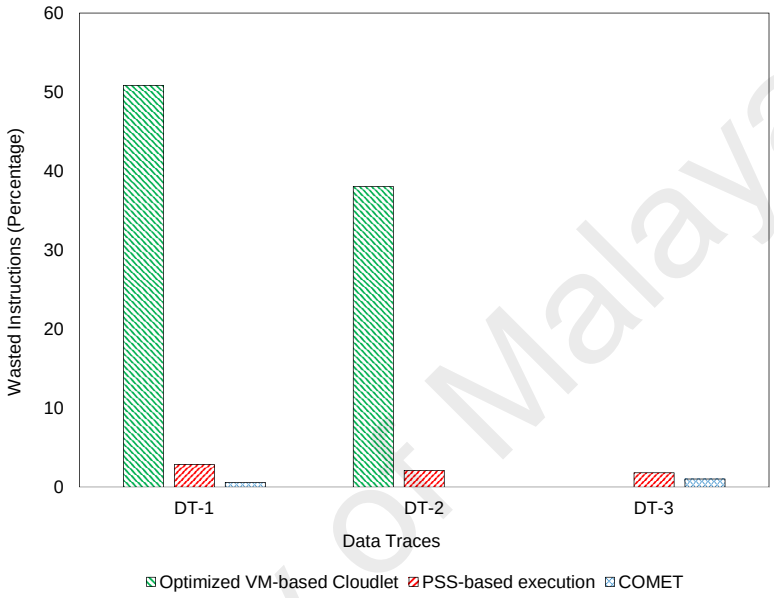


Figure 6.16: Computation Wastage Comparison With State-of-the-art Application Execution Frameworks

the computation performed on the previous cloudlet get wasted and the mobile device has to re-offload the execution from the initial stage to the newly discovered cloudlet. A small difference of computation wastage for PSS-based execution and COMET exists that is due to periodic nature of the PSS-based execution over the event-based approach of COMET. The PSS-based execution only shares the intermediate process states when the synchronization interval arrives. The synchronization interval for this scenario is of 15 seconds. However, COMET exchanges the intermediate states after the double of RTT that is relatively smaller than the synchronization interval considered for the PSS-based execution. In the third data trace, the computation wastage for optimized VM-based cloudlet is zero because the mobile device successfully gets back the VM-overlay from the first cloudlet.

However, we have observed through experiments that it is relatively less probable for mobile device to get back the VM-overlay before the complete disconnection.

6.4.3 Energy Consumption

Figure 6.17 shows the graphical representation of the comparison of PSS-based execution with the optimized VM-based cloudlet and COMET in terms of energy consumption.

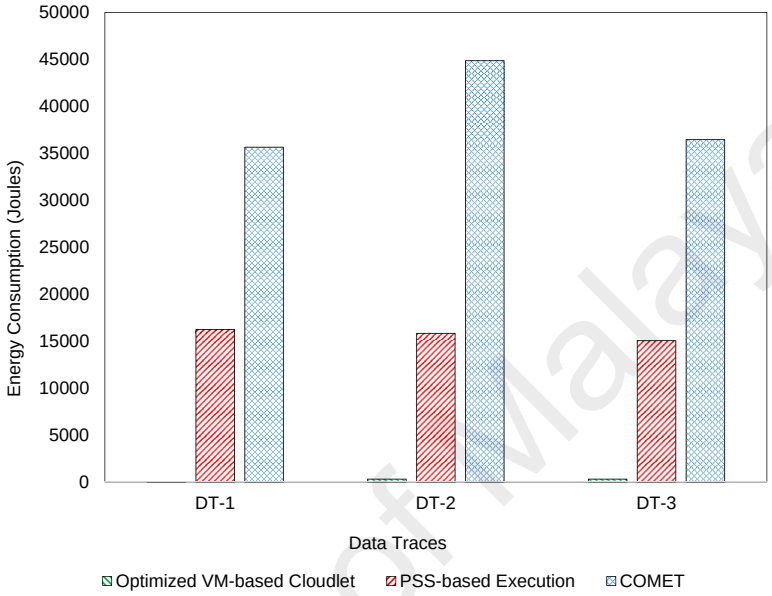


Figure 6.17: Energy Consumption Comparison With State-of-the-art Application Execution Frameworks

The energy consumption for optimized VM-based cloudlet is the least for all representative data traces. The energy consumption in case of PSS-based execution for three data traces is 16218.6 joules, 15825.6 joules, and 15040.2 joules, respectively. However, the optimized VM-based cloudlet consumes 99.83%, 98.12%, and 98.03% less energy than the consumption of PSS-based execution. The mobile device in case of optimized VM-based cloudlet execution does not perform the execution locally during the disconnection period. However, PSS-based execution supports the resumption of execution locally and performs the execution locally during the disconnection period. The energy consumption for COMET-based execution is 54.50%, 64.72%, and 58.74% higher than the consumption of PSS-based execution for three data traces respectively. The high consumption of energy for COMET-based execution is because of performing the rest

of the execution locally on the mobile device when the network disconnection occurs. However, PSS-based execution resynchronizes back the process states with the cloud and resumes the execution on the newly discovered cloudlet after the network disconnection that causes the relatively less consumption than that of COMET.

6.5 Conclusion

PSS-based execution is a two way synchronization of process states to address the issue of network disconnection while executing the application on the cloud server. PSS-based execution reduces the application execution time, synchronization overhead, and computation wastage for cloud-based mobile application execution when disconnection occurs during the execution.

We validated the set of equations presented in chapter 4 using real experiments. The execution time results of mathematical model are within the 3.7% of the empirical results. Similarly, the worst case difference between useful and wasteful computations on the mobile device is 17% while the best case is 0% as compared to 0% in case of analytical results. We studied the impact of synchronization interval on different performance parameters. It was observed that the execution time and computation wastage for shorter synchronization interval are less. However, the shorter synchronization interval increases the synchronization overhead. The results also gave an insight that the benefit of synchronization in terms of execution time and computation wastage diminishes as the synchronization interval increases.

In best case the execution time of PSS-based execution is 64% and 78% less than the execution time of optimized VM-based cloudlet and COMET, respectively. However, in the worst case the execution time of PSS-based execution is 0.41% and 14.99% less than the execution time of optimized VM-based cloudlet and COMET, respectively. Similarly, in best case, the computation wastage of PSS-based execution is 19.21 times less than the computation wastage of optimized VM-based cloudlet. However, in worst case the computation wastage of PSS-based execution is 3.4% of the computation wastage

of optimized VM-based cloudlet. The computation wastage of PSS-based execution is 2.2% and 0.10% higher than that of the COMET in worst and best case respectively. The optimized VM-based cloudlet consumes 99.83% and 98.03% less energy than the PSS-based execution in the worst and the best cases, respectively. However, PSS-based execution consumes 64.72% and 54% less energy than that of COMET in the best case and the worst case, respectively. Considering the results obtained by the performance analysis and comparison with the existing frameworks, it is concluded that PSS-based execution supports a lightweight two way synchronization mechanism for the cloud-based distributed mobile applications that efficiently manages the execution in case of network disconnection during the cloud execution.

We conclude the thesis by reflecting on the objectives set for this research in the first chapter. The purpose of this chapter is to summarize the research contributions and highlight the future work.

The chapter is organized into four sections. Section 7.1 discusses the reassessment of the objectives of this research work. Section 7.2 highlights contribution of the research work. Section 7.3 examines the scope and limitation of the research work. Section 7.4 gives future directions of the research work.

7.1 Reappraisal of the Research Objectives

The problem of application state loss because of network disconnections and its adverse impact on application execution time has been investigated and addressed in this thesis. Four objectives were set for the research in Section 1.4. We revisit the four objectives and highlight how the research study met the objectives.

The first objective was to review the application execution frameworks in MCC for acquiring the insight on the state-of-the-art with reference to network disconnection issue during the execution of cloud-based mobile application. A thematic taxonomy of conducted literature review has been devised to achieve the objective of 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 studied 150 papers in the broader domain of mobile computing and mobile cloud computing and reviewed the current literature on application execution frameworks by selecting 23 frameworks published during last five years. A number of features employed by frameworks to optimize the application performance have been identified. Qualitative analysis was done to investigate the critical aspects of state-of-the-art application execution frameworks performance and to identify the open issues for application execution frameworks in MCC.

The second objective was to investigate the impact of user mobility on cloud-based

mobile application execution in the cloud. To achieve this objective, we have collected the original mobility traces of the mobile users in a typical mobility scenario. We classified the mobility profiles into three types of permanent disconnection, intermittent disconnection with reconnection to same computing source and reconnection with different computing source. The mobility profiles have been replicated in the lab environment and impact on application execution has been studied. The empirical analysis showed the non-trivial impact of mobility and resulting disconnection on the application execution time. Formal analysis of the problem has been conducted. The analysis shows that under very realistic conditions, the existing frameworks were not able to adequately mitigate the impact of disconnections on cloud based mobile application execution. This established the problem of adverse impact of mobility on cloud based mobile application execution as non-trivial.

The third objective was to design and develop the solution to save the execution state for mobility support in order to minimize the execution time and synchronization overhead. A process state synchronization algorithm has been proposed to address the issue of network disconnection during the application execution on the cloud server. The process state synchronization algorithm exchanges the process states with the mobile device during the execution periodically, following a synchronization interval. On the network disconnection, the mobile device is able to resume the application execution locally from last synchronization state. The mobile device continues the execution until it gets back the connection to the cloud server.

On the connection re-establishment, the mobile device decides whether to continue the execution locally or resynchronize back the process states with the cloud server. If the mobile device resynchronizes back the process states with the cloud server, the cloud resumes the execution from the newly resynchronized process states. If the mobile device does not resynchronize back and just sends a resumption signal to the cloud server, the cloud server resumes from its own previous state. PSS algorithm minimizes the execution time in case of network disconnection during the execution of mobile application on the

cloud server. The proposed algorithm also reduces the computation wastage and energy consumption.

The final objective was to develop mathematical model for the proposed solution, validate the model using empirical analysis and compare the performance of proposed solution with the state-of-the-art cloud-based mobile application execution frameworks. An emulated MCC environment has been designed and developed by using the Desktop systems and Ethernet technology to emulate the mobile device, wireless technology, and cloud server. Proposed PSS has been implemented as a pair of linux kernel modules for empirical validation. A mathematical model has been developed to validate the performance of PSS implementation. The performance verification of PSS is done by comparing the performance of PSS with that of the optimized VM-based cloudlet and COMET.

The results for evaluation of PSS algorithm have been collected and compared for different disconnection profiles and state-of-the-art execution frameworks. The proposed algorithm reduces the execution time 42% and 17% in case of permanent disconnection and intermittent disconnections, respectively. For permanent disconnection, the computation wastage in case of PSS-based execution is reduced 99% as compared to without synchronization support. The comparison with existing execution frameworks shows that PSS reduces the execution time by up to 47% in case of intermittent network connectivity compared to COMET and by up to 35% in case of optimized VM-based cloudlet execution.

7.2 Contribution of the Research

This research produced a number of contributions to the body of knowledge which are summarized as follows:

- **Thematic Taxonomy:** The taxonomy is used to analyze the critical aspects of the current application execution frameworks and compare the frameworks on the basis of

significant parameters. The literature review contributed to identify open issues of application execution frameworks in MCC.

- **Process State Synchronization Algorithm:** We propose a PSS-based execution algorithm for addressing the issue of execution disruption that leads to additional execution time and computation wastage. Unlike the contemporary approaches for handling the mobility in MCC, PSS-based execution reduces the execution disruption with minimal synchronization overhead between the mobile device and cloud server. In case of network disconnection, the task can be resumed on the mobile device from last point of synchronization. The locally executed process states can be resynchronized back with the cloud server on the connection reestablishment. The existing frameworks such as COMET fails to resynchronize back with the cloud server and also suffers with the synchronization overhead because of the VM level information exchange. On the contrary, PSS is a lightweight distributed synchronization algorithm that reduces the operational overhead on the mobile device. Another key feature of PSS is bidirectional exchange of process states between the mobile device and the cloud server.

- **Mathematical Model:** We have mathematically modelled the proposed solution that is used to validate the performance of PSS algorithm. The model captures the key features of PSS to represent the proposed solution in mathematical form. The developed mathematical model was validated by comparing the results of the model with the emulated PSS-based execution performed in the lab environment.

We have successfully published our work in well reputed journals.

- **Accepted Articles on Research Topic:**

Ejaz Ahmed, Abdullah Gani, Muhammad Khurram Khan, Rajkumar Buyya, Samee U. Khan, Seamless Application Execution in Mobile Cloud Computing: Motivation,

Taxonomy, and Open Challenges, Journal of Network and Computer Applications, Vol. 52, Pages 154-172, June 2015 (Impact Factor 2.229)

Ejaz Ahmed, Abdullah Gani, Mehdi Sookhak, Siti Hafizah Ab Hamid, Feng Xia, Application Optimization in Mobile Cloud Computing: Motivation, Taxonomies, and Open Challenges, Journal of Network and Computer Applications, Vol. 52, Pages 52-68, June 2015 (Impact Factor 2.229)

Ejaz Ahmed, Adnan Akhunzada, Md. Wahiduzaman, Abdullah Gani, Siti Hafizah Ab Hamid, Rajkumar Buyya, Network-centric Performance Analysis of Runtime Application Migration in Mobile Cloud Computing, *Simulation Modelling Practice and Theory* Vol. 50, Pages 42-56 January, 2015 (Impact Factor 1.159).

- **Articles Under Review on Research Topic:**

Ejaz Ahmed, Anjum Naveed, Abdullah Gani, Siti Hafizah Ab Hamid, Formal Analysis of Seamless Application Execution in Mobile Cloud Computing, under review, 2015 (Q1)

Ejaz Ahmed, Anjum Naveed, Abdullah Gani, Siti Hafizah Ab Hamid, Cloud-based Execution Management for Mobility Support using Process State Synchronization, under review, 2015 (Q1)

- **Accepted Articles in Other Research Areas:**

Ejaz Ahmed, Abdullah Gani, Saeid Abolfazli, Liu Jie Yao, Samee U. Khan, Channel Assignment in Cognitive Radio Networks: Taxonomy, Open Issues, and Challenges, *IEEE Communication Surveys and Tutorials*, in press, October, 2014 (Impact Factor 6.9)

Ejaz Ahmed, Muhammad Shiraz, Abdullah Gani, Spectrum-aware Distributed Channel Assignment in Cognitive Radio Wireless Mesh Networks, *Malaysian Journal of Computer Science*, Vol. 26(3), Pages 232-250, September 2013 (Impact Factor 0.5)

Ejaz Ahmed, Junaid Qadir, Adeel Baig, High-Throughput Transmission-Quality-Aware Broadcast Routing in Cognitive Radio Networks, *Wireless Networks*, Vol. 21, No. 4, October, 2014 (Impact Factor 1.02)

This research has contributed to the following collaborative research articles:

- **Articles in Collaboration with Group Members:**

Adnan Akhanzada, Ejaz Ahmed, Abdullah Gani, Muhammad Khurram Khan, Muhammad Imran, Sghaier Guizani, Securing the Software Defined Networks: Taxonomy, Requirements, and Open Issues, *IEEE Communications Magazine*, Vol. 53, No. 4, December 2014 (Impact Factor 4.46)

Jie Yao Liu, Ejaz Ahmed, Muhammad Shiraz, Abdullah Gani, Rajkumar Buyya, Ahsan Qureshi, Application Partitioning Algorithms in Mobile Cloud Computing: Taxonomy, Review and Future Directions, *Journal of Network and Computer Applications*, Vol. 48, Pages 99-117, February 2015 (Impact Factor 1.77)

Mehdi Sookhak, Hamid Teleban, Ejaz Ahmed, Abdullah Gani, Muhammad Khurram Khan, A Review on Remote Data Auditing in Single Cloud Server: Taxonomy and Open Issues, *Journal of Network and Computer Applications*, Vol. 43, Pages 121-141 August 2014 (Impact Factor 1.77)

Saeid Abolfazli, Zohreh Sanaei, Ejaz Ahmed, Abdullah Gani, Raj Buyya, Cloud-based Augmentation for Mobile Devices: Motivation, Taxonomies, and Open Issues, *IEEE Communications Surveys and Tutorials*, Vol. 16 Issue 1, Pages 337-368, February 2014 (Impact Factor 6.9)

Muhammad Shiraz, Ejaz Ahmed, Abdullah Gani, Investigation on Runtime Partitioning of Elastic Mobile Applications for Mobile Cloud Computing, *Journal of Supercomputing*, Vol. 67, Issue 1, Pages 84-103, January 2004 (Impact Factor 0.9)

7.3 Research Scope and Limitations

The proposed process state synchronization algorithms are effective for all interactive cloud and cloudlet based mobile application executions. The algorithms will work for legacy applications as well as newly developed applications. In addition, the client server based applications can also take advantage of the algorithms, provided the additional data to be transfered with the application is smaller in size.

The algorithms are not suitable for state transfer of virtualized application executions because of the heavy load of transferring the virtual machine state, most of which is trivial, with reference to a particular application execution. Similarly, the proposed algorithms are less suitable for use in heterogeneous environments for two reasons. Firstly, the state transfer and resume poses significantly higher overhead in such environments. Secondly, a generic method for state capture in heterogeneous environment is not yet available and requires extensive kernel support. Finally, the algorithms have been tested with linux kernel over x86 architecture and are not guaranteed to work on other platforms.

7.4 Future Work

A huge amount of research effort goes into a PhD based research study. However, a single PhD study is never enough to cover all the aspects of any research topic. In the following, we present an insight into some of the possible future directions where further research can be conducted by extending the proposed research work. This research is focused only on the synchronization of process states between the homogeneous computing platforms. The homogeneity of the platform in MCC environment is achieved by running the emulator on the cloud server. The focus of our research is to provide the two way synchronization of process states, which does not have an open file descriptors associated with it. Hence, the future research work includes extending the scope of this research to address the issue of synchronizing the process states including the all associated entities such as file descriptor and network sockets, synchronizing the process states between the heterogeneous computing platforms, and predicting the mobility to adapt the

synchronization interval accordingly.

The following section discusses the future directions of this research:

- The issue of synchronizing the process states along with all associated entities, such as file descriptors and network sockets, is aimed to be addressed in our future research. The application processes usually do not open the file descriptors and network sockets. The distributed processes, which have opened the network sockets, can only be migrated and executed from one cloudlet to another cloudlet if the associated network sockets are migrated along with the process states.

- We aim to address the issue of synchronizing the process states between the heterogeneous computing devices in our future research. The application execution that is performed on one computing platform cannot be directly resumed on the different computing platform. A mapping of memory segments and registers is required for the successful resumption of the process on the different computing platform.

- We also aim to address the issue of predicting the mobility patterns to adapt the synchronization interval according to the dynamic conditions of the mobile computing environment. The mobility pattern estimation will enable us to adapt the synchronization interval for reducing the computation wastage and synchronization overhead in highly dynamic mobile computing environment.

There can be multiple other research directions, that we might be unaware of at this particular stage. No matter how complete the research study be, the intriguing minds can always find new ways of looking at existing things and have always been able to come up with new ways of improving the state-of-the-art and the body of knowledge.

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.

Ahmed, E., Akhunzada, A., Whaiduzzaman, M., Gani, A., Ab Hamid, S. H., & Buyya, R. (2015). Network-centric performance analysis of runtime application migration in mobile cloud computing. *Simulation Modelling Practice and Theory*, *50*, 42–56.

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.

Ahmed, E., Gani, A., Sookhak, M., Ab Hamid, S. H., & Xia, F. (2015). Application optimization in mobile cloud computing: Motivation, taxonomies, and open challenges. *Journal of Network and Computer Applications*, *52*, 52-68.

Ahn, J. (2009). Lightweight fault-tolerance mechanism for distributed mobile agent-based monitoring. In *Proc. of 6th IEEE Consumer Communications and Networking Conference, (CCNC'09), Las Vegas, Nevada, USA* (p. 1-5).

Al-Muhtadi, J., Mickunas, D., & Campbell, R. (2002). A lightweight reconfigurable security mechanism for 3G/4G mobile devices. *IEEE Wireless Communications*, *9*(2), 60-65.

Amazon. (2014a, 2 December). *Amazon elastic compute cloud (amazon ec2).* Available from `http://aws.amazon.com/ec2/`

Amazon. (2014b, 2 December). *Amazon simple storage service (amazon s3).* Available from `http://aws.amazon.com/s3/`

Androcec, D., Vrcek, N., & Kungas, P. (2015). Service-Level Interoperability Issues of Platform as a Service. In *IEEE World Congress on Services (SERVICES)* (p. 349-356).

Ardagna, C., Damiani, E., Frati, F., Rebeccani, D., Ughetti, M., et al. (2012). Scalability patterns for platform-as-a-service. In *IEEE 5th International Conference on Cloud Computing (CLOUD'12)* (p. 718-725).

Benharref, A., & Serhani, M. (2014, Jan). Novel Cloud and SOA-Based Framework for E-Health Monitoring Using Wireless Biosensors. *IEEE Journal of Biomedical and Health Informatics*, *18*(1), 46-55.

Bertolli, C., Buono, D., Mencagli, G., Torquati, M., Vanneschi, M., Mordacchini, M., et al. (2010). Resource discovery support for time-critical adaptive applications. In *Proc. of the 6th International Wireless Communications and Mobile Computing Conference (IWCMC '10), Caen, France* (pp. 504–508).

Boniface, M., Nasser, B., Papay, J., Phillips, S. C., Servin, A., Yang, X., et al. (2010). Platform-as-a-service architecture for real-time quality of service management in clouds. In *Fifth International Conference on Internet and Web Applications and Services (ICIW'10)* (p. 155-160).

Bourouis, A., Zerdazi, A., Feham, M., & Bouchachia, A. (2013). M-health: Skin disease analysis system using smartphone's camera. *Procedia Computer Science*, *19*, 1116–1120.

Chen, S., Wang, Y., & Pedram, M. (2013). A semi-markovian decision process based control method for offloading tasks from mobile devices to the cloud. In *Globecom* (p. 2885-2890).

Chen, X. (2015). Decentralized computation offloading game for mobile cloud computing. *IEEE Transactions on Parallel and Distributed Systems*, *26*(4), 974-983.

Chitika. (2015, Accessed on: 23 April). *Hour-by-hour examination: Smartphone, tablet, and desktop usage rates.* Available from `https://chitika.com/browsing-activity-by-hour`

Chun, B.-G., Ihm, S., Maniatis, P., Naik, M., & Patti, A. (2011). Clonecloud: elastic execution between mobile device and cloud. In *Proceedings of the 6th EuroSys conference on Computer systems (EUROSYS'11), Salzburg, Austria* (pp. 301–314).

Chun, B.-G., & Maniatis, P. (2009). Augmented smartphone applications through clone cloud execution. In *Proceedings of the 8th Workshop on Hot Topics in Operating Systems (HotOS'09), Monte Verita, Switzerland* (pp. 8–14).

Chun, B.-G., & Maniatis, P. (2010). Dynamically partitioning applications between weak devices and clouds. In *Proc. of the 1st ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond, (MCS'10), San Francisco, USA* (pp. 1–5).

Church, A. (1940). A formulation of the simple theory of types. *The journal of symbolic logic*, *5*(02), 56–68.

Cisco. (2015, 3 february). *Cisco visual networking index: Global mobile data traffic forecast update, 2014-2019.* Available from `https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white_paper_c11-520862.pdf`

Cuervo, E., Balasubramanian, A., Cho, D.-k., Wolman, A., Saroiu, S., Chandra, R., et al. (2010). MAUI: making smartphones last longer with code offload. In *Proc. of the 8th international conference on Mobile systems, applications, and services (MobiSys'10), San Francisco, CA, USA*. (pp. 49–62).

Dean, J., & Ghemawat, S. (2008). MapReduce: simplified data processing on large clusters. *Communications of the ACM*, *51*(1), 107–113.

Developers, V. (2015, Accessed on 5 May). *Lackey: an example tool.* Available from `http://valgrind.org/docs/manual/lk-manual.html`

Di Modica, G., Tomarchio, O., & Vita, L. (2011). Resource and service discovery in SOAs: A P2P oriented semantic approach. *International Journal of Applied Mathematics and Computer Science*, *21*(2), 285–294.

Dropbox. (2015, Accessed on 2 June). Available from https://www.dropbox.com/

Facebook. (2015, Accessed on 2 June). Available from https://www.facebook.com/?_rdr

Fesehaye, D., Gao, Y., Nahrstedt, K., & Wang, G. (2012). Impact of cloudlets on interactive mobile cloud applications. In *Proc. of 16th International Enterprise Distributed Object Computing Conference (EDOC'12), Beijing, China* (pp. 123–132).

Forbes. (2015, Accessed on: 23 April). *Cities with the most extreme commutes.* Available from http://www.forbes.com/sites/jennagoudreau/2013/03/05/citieswith-the-most-extreme-commutes/

Ghosh, R., Longo, F., Xia, R., Naik, V. K., & Trivedi, K. S. (2014). Stochastic model driven capacity planning for an infrastructure-as-a-service cloud. *IEEE Transactions on Services Computing*, *7*(4), 667-680.

Giurgiu, I., Riva, O., Juric, D., Krivulev, I., & Alonso, G. (2009). Calling the cloud: Enabling mobile phones as interfaces to cloud applications. In *Proc. of the 10th ACM/IFIP/USENIX International Conference on Middleware (Middleware'09), Champaign, IL, USA* (p. 83-102). Springer.

Google. (2012, Accessed on: 2 December). *Google app engine.* Available from cloud.google.com/appengine

Gordon, M. S., Jamshidi, D. A., Mahlke, S., Mao, Z. M., & Chen, X. (2012). COMET: code offload by migrating execution transparently. In *Proc. of the 10th USENIX conference on Operating Systems Design and Implementation, (OSDI'12), Hollywood, CA.* (Vol. 12, pp. 93–106).

Goscinski, A., & Brock, M. (2010). Toward dynamic and attribute based publication, discovery and selection for cloud computing. *Future Generation Computer Systems*, *26*(7), 947–970.

Goyal, S., & Carter, J. (2004). A lightweight secure cyber foraging infrastructure for resource-constrained devices. In *Proc. of 6th IEEE Workshop on Mobile Computing Systems and Applications, (WMCSA'04), English Lake District, UK* (pp. 186–195).

Ha, K., Pillai, P., Richter, W., Abe, Y., & Satyanarayanan, M. (2013). Just-in-time provisioning for cyber foraging. In *Proc. of the 11th annual international conference on Mobile systems, applications, and services* (pp. 153–166).

Huerta-Canepa, G., & Lee, D. (2010). A virtual cloud computing provider for mobile devices. In *Proc. of the 1st ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond, (MCS'10), San Francisco, CA, USA* (pp. 1–5).

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.

Khan, A. N., Mat Kiah, M., Khan, S. U., & Madani, S. A. (2012). Towards secure mobile cloud computing: A survey. *Future Generation Computer Systems*, *29*(5), 1278–1299.

Kosta, S., Aucinas, A., Hui, P., Mortier, R., & Zhang, X. (2012). Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading. In *Proc. of 31st IEEE International Conference on Computer Communications (INFOCOM'12), Orlando, Florida, USA* (pp. 945–953).

Koukoumidis, E., Lymberopoulos, D., Strauss, K., Liu, J., & Burger, D. (2012). Pocket cloudlets. *ACM SIGPLAN Notices*, *47*(4), 171–184.

Kovachev, D., Cao, Y., & Klamma, R. (2012). Augmenting pervasive environments with an XMPP-based mobile cloud middleware. In *Proc. of Fourth International Conference on Mobile Computing, Applications and Services (MobiCASE'12), Seattle, Washington, United States* (pp. 361–372). Springer.

Kovachev, D., Yu, T., & Klamma, R. (2012). Adaptive computation offloading from mobile devices into the cloud. In *Proc. of 10th International Symposium on Parallel and Distributed Processing with Applications (ISPA'12), Madrid, Spain* (pp. 784–791).

Lee, B.-D. (2012). A framework for seamless execution of mobile applications in the cloud. *Recent Advances in Computer Science and Information Engineering*, *126*, 145–153.

Lingjun, P., Jingdong, X., Bowen, Y., & Jianzhong, Z. (2014). Smart cafe: A mobile local computing system based on indoor virtual cloud. *Communications, China*, *11*(4), 38-49.

Litzkow, M., Tannenbaum, T., Basney, J., & Livny, M. (1997). *Checkpoint and migration of unix processes in the condor distributed processing system*. Computer Sciences Department, University of Wisconsin.

Liu, J., Kumar, K., & Lu, Y.-H. (2010). Tradeoff between energy savings and privacy protection in computation offloading. In *Proc. of ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED'10), Austin, TX, USA* (p. 213-218).

Liu, J. y., Ahmed, E., Shiraz, M., Gani, A., Buyya, R., & Qureshi, A. (2015). Application partitioning algorithms in mobile cloud computing: Taxonomy, review and future directions. *Journal of Network and Computer Applications*, *48*, 99-117.

Ma, D., & Kauffman, R. J. (2014). Competition between software-as-a-service vendors. *IEEE Transactions on Engineering Management*, *61*(4), 717-729.

Mao, H., Xiao, N., Shi, W., & Lu, Y. (2012). Wukong: A cloud-oriented file service for mobile internet devices. *Journal of Parallel and Distributed Computing*, *72*(2),

171–184.

Marinelli, E. E. (2009). *Hyrax: Cloud computing on mobile devices using mapreduce* (Tech. Rep.). DTIC Document.

Martins, R., Narasimhan, P., Lopes, L., & Silva, F. (2010). Lightweight fault-tolerance for Peer-to-Peer middleware. In *Proc. of 29th IEEE Symposium on Reliable Distributed Systems (SRDS'10), Delhi, India* (pp. 313–317).

Mehendale, H., Paranjpe, A., & Vempala, S. (2011). Lifenet: a flexible ad hoc networking solution for transient environments. In *Proc. of the ACM SIGCOMM conference (SIGCOMM'11), Toronto, On, Canada* (p. 446-447). ACM.

Mitra, S. (2010). Seamless mobility management and QoS support for multihomed mobile node in heterogeneous wireless networks. In *Proc. of International Conference on Industrial and Information Systems (ICIIS'10), Dalian, China* (p. 145-150).

Modares, H., Moravejosharieh, A., Lloret, J., & Salleh, R. B. (2014). A survey on proxy mobile IPv6 handover. *IEEE Systems Journal, in press*.

Nguyen, K.-K., Cheriet, M., & Lemay, M. (2013). Enabling infrastructure as a service (IaaS) on IP networks: from distributed to virtualized control plane. *IEEE Communications Magazine*, *51*(1), 136-144.

Ojala, A. (2013, May). Software-as-a-Service Revenue Models. *IT Professional*, *15*(3), 54-59.

Osman, S., Subhraveti, D., Su, G., & Nieh, J. (2002). The design and implementation of Zap: A system for migrating computing environments. *ACM SIGOPS Operating Systems Review*, *36*(SI), 361–376.

Ou, S., Yang, K., & Zhang, J. (2007). An effective offloading middleware for pervasive services on mobile devices. *Pervasive and Mobile Computing*, *3*(4), 362–385.

Pirozmand, P., Wu, G., Jedari, B., & Xia, F. (2014). Human mobility in opportunistic networks: Characteristics, models and prediction methods. *Journal of Network and Computer Applications*, *42*, 45–58.

Raad, P., Colombo, G., Chi, D. P., Secci, S., Cianfrani, A., Gallard, P., et al. (2013). Achieving sub-second downtimes in internet-wide virtual machine live migrations in LISP networks. In *Proc. of IFIP/IEEE International Symposium on Integrated Network Management (IM'13), Ghent, Belgium* (p. 286-293).

Satyanarayanan, M., Bahl, P., Caceres, R., & Davies, N. (2009). The case for VM-based cloudlets in mobile computing. *IEEE Pervasive Computing*, *8*(4), 14–23.

Shaukat, U., Ahmed, E., Anwar, Z., & Xia, F. (2016). Cloudlet deployment in local wireless networks: Motivation, architectures, applications, and open challenges. *Journal of Network and Computer Applications*, *62*, 18-40.

Shi, C., Habak, K., Pandurangan, P., Ammar, M., Naik, M., & Zegura, E. (2014). COSMOS: computation offloading as a service for mobile devices. In *Proc. of the 15th ACM international symposium on Mobile ad hoc networking and computing* (pp. 287–296).

Shuja, J., Bilal, K., Madani, S., Othman, M., Ranjan, R., Balaji, P., et al. (2014). Survey of techniques and architectures for designing energy-efficient data centers. *Systems Journal, IEEE*, *PP*(99), 1-13.

Shuja, J., Madani, S. A., Bilal, K., Hayat, K., Khan, S. U., & Sarwar, S. (2012). Energy-efficient data centers. *Computing*, *94*(12), 973-994.

Spata, M. O., & Rinaudo, S. (2011). Virtual machine migration through an intelligent mobile agents system for a cloud grid. *Journal of Convergence Information Technology*, *6*(6).

Takahashi, K., Sasada, K., & Hirofuchi, T. (2012). A fast virtual machine storage migration technique using data deduplication. In *Proc. of The Third International Conference on Cloud Computing, GRIDs, and Virtualization (CLOUD COMPUTING'12), Nice, France* (pp. 57–64).

Verbelen, T., Simoens, P., De Turck, F., & Dhoedt, B. (2012a). AIOLOS: Middleware for improving mobile application performance through cyber foraging. *Journal of Systems and Software*, *85*(11), 2629–2639.

Verbelen, T., Simoens, P., De Turck, F., & Dhoedt, B. (2012b). Cloudlets: Bringing the cloud to the mobile user. In *Proc. of the third ACM workshop on Mobile cloud computing and services, (MCS'12), New York, USA* (pp. 29–36).

Verbelen, T., Stevens, T., Simoens, P., De Turck, F., & Dhoedt, B. (2011). Dynamic deployment and quality adaptation for mobile augmented reality applications. *Journal of Systems and Software*, *84*(11), 1871–1882.

Vu, L., Nguyen, P., Nahrstedt, K., & Richerzhagen, B. (2015). Characterizing and modeling people movement from mobile phone sensing traces. *Pervasive and Mobile Computing*, *17*, 220-235.

Wang, X., Chen, M., Kwon, T., Yang, L., & Leung, V. (2013, June). AMES-Cloud: A Framework of Adaptive Mobile Video Streaming and Efficient Social Video Sharing in the Clouds. *IEEE Transactions on Multimedia*, *15*(4), 811-820.

Wang, X., Gui, Q., Liu, B., Jin, Z., & Chen, Y. (2014, May). Enabling Smart Personalized Healthcare: A Hybrid Mobile-Cloud Approach for ECG Telemonitoring. *IEEE Journal of Biomedical and Health Informatics*, *18*(3), 739-745.

Wang, Y., Wu, J., & Yang, W.-S. (2013). Cloud-based multicasting with feedback in mobile social networks. *IEEE Transactions on Wireless Communications*, *12*(12), 6043-6053.

Whaiduzzaman, M., Haque, M. N., Rejaul Karim Chowdhury, M., & Gani, A. (2014). A study on strategic provisioning of cloud computing services. *The Scientific World*

Journal, *2014*.

White, T. (2012). *Hadoop: the definitive guide*. O'Reilly.

Wong, L.-H., Chai, C. S., Zhang, X., & King, R. (2015, Jan). Employing the TPACK Framework for Researcher-Teacher Co-Design of a Mobile-Assisted Seamless Language Learning Environment. *IEEE Transactions on Learning Technologies*, *8*(1), 31-42.

Wu, L., Garg, S. K., Versteeg, S., & Buyya, R. (2014). SLA-Based Resource Provisioning for Hosted Software-as-a-Service Applications in Cloud Computing Environments. *IEEE Transactions on Services Computing*, *7*(3), 465-485.

Xia, Y., Zhou, M., Luo, X., Zhu, Q., Li, J., & Huang, Y. (2015). Stochastic Modeling and Quality Evaluation of Infrastructure-as-a-Service Clouds. *IEEE Transactions on Automation Science and Engineering*, *12*(1), 162-170.

Xiang, L., Ye, S., Feng, Y., Li, B., & Li, B. (2014, April). Ready, Set, Go: Coalesced offloading from mobile devices to the cloud. In *Proc. of IEEE INFOCOM, Toronto, Canada* (p. 2373-2381).

Xiao, Z., & Xiao, Y. (2013). Security and privacy in cloud computing. *IEEE Communications Surveys & Tutorials*, *15*(2), 843-859.

Yang, L., Cao, J., Yuan, Y., Li, T., Han, A., & Chan, A. (2012). A framework for partitioning and execution of data stream applications in mobile cloud computing. In *Proc. of IEEE 5th International Conference on Cloud Computing, (CLOUD'12), Honolulu, Hawaii, USA* (pp. 794–802).

Zekri, M., Jouaber, B., & Zeghlache, D. (2012). A review on mobility management and vertical handover solutions over heterogeneous wireless networks. *Computer Communications*, *35*(17), 2055–2068.

Zhang, X., Jeong, S., Kunjithapatham, A., & Gibbs, S. (2010). Towards an elastic application model for augmenting computing capabilities of mobile platforms. *Mobile wireless middleware, operating systems, and applications*, *48*, 161–174.

Zhang, Y., Niyato, D., Wang, P., & Tham, C.-K. (2014). Dynamic offloading algorithm in intermittently connected mobile cloudlet systems. In *IEEE International Conference on Communications (ICC'14)* (pp. 4190–4195).

Zhao, B., Xu, Z., Chi, C., Zhu, S., & Cao, G. (2012). Mirroring smartphones for good: A feasibility study. *Mobile and Ubiquitous Systems: Computing, Networking, and Services*, *73*, 26–38.