A LIGHTWEIGHT FRAMEWORK FOR INTENSIVE MOBILE APPLICATION PROCESSING IN MOBILE CLOUD COMPUTING

MUHAMMAD SHIRAZ

FACULTY OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY
UNIVERSITI MALAYA
KUALA LUMPUR

2013

A LIGHTWEIGHT FRAMEWORK FOR INTENSIVE MOBILE APPLICATION PROCESSING IN MOBILE CLOUD COMPUTING

MUHAMMAD SHIRAZ

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

FACULTY OF COMPUTER SCIENCE AND
INFORMATION TECHNOLOGY
UNIVERSITY OF MALAYA
KUALA LUMPUR
2013

UNIVERSITI MALAYA

ORIGINAL LITERARY WORK DECLARATION

Name of Candidate: Muhammad Shiraz

Registration/Matric No: WHA100052

Name of Degree: Doctor of Philosophy

Title of Project Paper/Research Report/Dissertation/Thesis (“this Work”): Thesis

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 23/8/2013

Subscribed and solemnly declared before,

Witness’s Signature Date

Name:
Designation:

Abstract

Mobile Cloud Computing (MCC) enables computational intensive and ubiquitous mobile applications by leveraging the services of computational clouds. Human dependency on contemporary smartphones increases rapidly in various domains such as enterprise, e-learning and entertainment, gamming, management information systems, and healthcare. However, mobile applications on the latest generation of smartphones and tablets are still constrained by battery power, CPU potentials and memory capacity of the Smart Mobile Devices (SMDs). Therefore, MCC employs computational offloading as a significant software level solution for alleviating the resources limitations in SMDs. Recently, a number of computational offloading frameworks are proposed for the processing of computational intensive mobile applications in MCC. The traditional computational offloading frameworks implement intensive techniques for computational offloading in MCC which results in high energy consumption and longer turnaround time of the mobile applications. Therefore, lightweight techniques are imperative for the processing of computational intensive applications in MCC. Lightweight techniques enable computational intensive mobile application deployment and execution with minimal resources utilization on SMDs. As a result, mobile users can utilize distributed cloud services with lower computational load on mobile devices, shorter turnaround time of the application and longer lasting battery lifetime. This research investigates the resources intensive features of traditional computational offloading frameworks and proposes a lightweight framework for the processing intensive mobile applications in MCC. The additional cost of runtime computational offloading is investigated by implementing application offloading mechanism in the real mobile cloud computing environment. Distributed and Elastic Application Processing (DEAP) framework is proposed as a lightweight solution for the intensive application processing in MCC. DEAP framework reduces the cost of migration of application binary file and data file of the running instances of the mobile application. As a result, the size of data transmission over the wireless network medium, turnaround time of the intensive operations and energy consumption cost on mobile device is reduced considerably. DEAP framework is evaluated in the emulation environment on the Android virtual device instance. The performance of DEAP framework is validated by benchmarking prototype application in the real mobile cloud computing environment. Results of different experimental scenarios are compared to validate the lightweight nature of DEAP framework. It is found that by employing DEAP framework the cost of migration of application binary file and data file of the running instances of the application is reduced. As a result, the size of data transmission over the wireless network medium, turnaround time of the intensive operations and energy consumption cost on mobile device is reduced. DEAP framework reduces resources utilization and the cost of distributed processing of the prototype mobile application in MCC as follows: RAM allocation on mobile device by 71.5 percent, CPU utilization on mobile device by 55 percent, the size of data transmission over the wireless network medium by 84 percent, turnaround time of the application by 79.8 percent and energy consumption cost by 81 percent. Hence, DEAP framework provides a lightweight application layer solution for intensive mobile application processing in MCC.

Abstrak

Mobile Cloud Computing (MCC) membolehkan pengiraan intensif dan di mana-mana aplikasi mudah alih dengan memanfaatkan perkhidmatan awan pengiraan. Pergantungan manusia pada telefon pintar kontemporari meningkatkan pesat dalam pelbagai domain seperti perusahaan, e-pembelajaran dan hiburan, gamming, sistem maklumat pengurusan, dan penjagaan kesihatan. Walau bagaimanapun, aplikasi mudah alih pada generasi terbaru telefon pintar dan tablet masih dikekang oleh kuasa bateri, potensi CPU dan kapasiti memori Devices Pintar Bergerak (SMDS). Oleh itu, MCC menggunakan pengiraan pemunggahan sebagai perisian yang signifikan tahap penyelesaian untuk mengurangkan batasan sumber di SMDS. Baru-baru ini, beberapa rangka kerja pemunggahan pengiraan dicadangkan untuk memproses aplikasi intensif pengiraan mudah alih di MCC. Tradisional pengiraan pemunggahan rangka melaksanakan teknik intensif untuk Pemunggahan pengiraan di daerah yang menyebabkan penggunaan tenaga yang tinggi dan masa pemulihan yang lebih lama aplikasi mudah alih. Oleh itu, teknik ringan penting untuk pemprosesan aplikasi pengiraan intensif di daerah. Teknik Ringan membolehkan permohonan penempatan pengiraan intensif mudah alih dan pelaksanaan dengan sumber-sumber yang minimum ke atas penggunaan SMDS. Hasilnya, pengguna telefon bimbit boleh menggunakan perkhidmatan awan diedarkan dengan beban pengiraan pada peranti mudah alih yang lebih rendah, pemulihan yang lebih pendek masa permohonan dan bateri lebih tahan lama seumur hidup. Kajian ini menyiasat sumber ciri-ciri kerangka tradisional pemunggahan pengiraan intensif dan mencadangkan satu rangka kerja ringan untuk aplikasi intensif pemprosesan mudah alih di MCC. Kos tambahan Pemunggahan pengiraan runtime disiasat dengan melaksanakan mekanisme permohonan pemunggahan dalam persekitaran pengkomputeran awan sebenar bimbit. Diedarkan dan Pemprosesan Permohonan Anjal (DEAP) rangka kerja yang dicadangkan sebagai penyelesaian ringan untuk memproses permohonan intensif dalam daerah. DEAP rangka kerja mengurangkan kos penghijrahan permohonan fail fail dan data perduaan keadaan berjalan aplikasi mudah alih. Akibatnya, saiz penghantaran data melalui medium rangkaian wayarles, masa pemulihan operasi intensif dan kos penggunaan tenaga pada peranti mudah alih dikurangkan dengan ketara. DEAP rangka kerja dinilai dalam persekitaran emulasi pada contoh peranti Android maya. Prestasi DEAP rangka kerja disahkan oleh aplikasi prototaip penandaarasan dalam awan sebenar persekitaran pengkomputeran mudah alih. Keputusan senario eksperimen yang berbeza berbanding untuk mengesahkan sifat ringan rangka kerja DEAP. Ia mendapati bahawa dengan menggunakan rangka kerja DEAP kos penghijrahan permohonan fail fail dan data perduaan keadaan berjalan permohonan dikurangkan. Akibatnya, saiz penghantaran data melalui medium rangkaian wayarles, masa pemulihan operasi intensif dan kos penggunaan tenaga pada peranti mudah alih dikurangkan. DEAP rangka mengurangkan penggunaan sumber dan kos pemprosesan diedarkan permohonan prototaip mudah alih di daerah seperti berikut: RAM peruntukan pada peranti mudah alih 79,8 peratus, penggunaan CPU pada peranti mudah alih 77 peratus, saiz penghantaran data melalui rangkaian wayarles sederhana peratus 84, masa pemulihan peratus permohonan

80,6 dan penggunaan tenaga kos 69,9 peratus. Oleh itu, rangka kerja DEAP menyediakan lapisan permohonan penyelesaian ringan untuk memproses permohonan intensif mudah alih di MCC.

Acknowledgement

First of all, I am thankful to Almighty Allah who enabled me to complete my studies with distinction. I would like to offer special thanks to my supervisor, Associate Professor Dr. Abdullah Gani for his invaluable guidance, supervision, and encouragement to me throughout this research. Dr. Abdullah Gani, not only provided helpful suggestions, but also accepted responsibility to oversee this research, and guided me to the successful completion of this thesis. He provided me the opportunity to broaden my professional experience and prepare me for future challenges.

I would like to express my sincerest gratitude and appreciation to my family for their endless love and support during my life. Without their moral support, this dissertation would never have been completed. No words can express my real feelings, so I dedicate my first achievement in my life as a small gift to them. I would like to express my deep appreciation to my dear lab friends, who provided so much support and encouragement throughout this research and studies process. I wish them all the best in their future undertaking.

Table of Contents

List of Figures

List of Tables

List of Acronyms

ACID	Atomicity, Concurrency, Isolation and Durability
API	Application Program Interface
APT	Application Processing Time
ARM	Advanced RISC Machine
CEU	Client Execution Unit
CES	Cloud Elasticity Service
CORBA	Common Object Request Broker Architecture
CPU	Central Processing Unit
CFI	Cloud Fabric Interface
CM	Cloud Manager
DAPFs	Distributed Application Processing Frameworks
DEMAC	Distributed Environment for Mobility Aware Computing
DEM	Device Elasticity Manager
DISHES	Distributed Shell System
DVM	Dalvik Virtual Machine
ECC	Energy Consumption Cost
EC2	Elastic Cloud Compute
FTP	File Transfer Protocol
GB	Giga Byte
GPRS	General Packet Radio Service
HMAC	Hash-Based Message Authentication Code
HTTP	Hypertext Transfer Protocol
I/O	Input / Output
IP	Internet Protocol
JVM	Java Virtual Machine
MAUI	Mobile Assistance Using Infrastructure
MB	Mega Bytes
MIPS	Millions of Instruction Per Second
NDIS	Network Driver Interface Specification
LAN	Local Area Network
MAC	Medium Access Control
MB	Mega Byte
MCC	Mobile Cloud Computing
DEAP	Distributed Elastic Application Processing
OS	Operating System
P2P	Peer to Peer
PC	Personal Computer
PDA	Personal Digital Assistant
POP	Primary Operating Procedure
PIE	Pipe I/O Exec Subsystem
RISC	Reduced Instruction Set Computer
RAM	Random Access Memory
RCP	Rich Client Platform
RTT	Round Trip Time
SAL	Storage Abstraction Layer
SD	Service Directory

S3	Simple Storage Service
SOAP	Simple Object Access Protocol
SOP	Secondary Operating Procedure
SMD	Smart Mobile Device
SSL	Secure Socket Layer
TCP	Transmission Control Protocol
TSL	Transport Layer Security
TSP	Telecommunication Service Provider
TT	Turnaround Time
UDP	User Datagram Protocol
UMSC	Universal Mobile Service Cell
URL	Uniform Resource Locator
VFS	Virtual File Service
VM	Virtual Machine
WiFi	Wireless Fidelity
WSDL	Web Service Description Language
XML	Extensible Markup Language

CHAPTER 1

Introduction

This chapter presents theoretical framework and motivations for the proposed research. It discusses the problem statement, states the objectives and describes the methodology used for the proposed research. The chapter is divided into six sections. Section 1.2 highlights motivations for the proposed research by explaining the importance of the proposed work and significance of the proposed solution. Section 1.3 summarizes the problem statement by highlighting issues in the traditional computational offloading frameworks. Section 1.4 highlights the research objectives. Section 1.5 summarizes the methodology used in this research and section 1.6 sketches the layout of the thesis.

1.1 Background

Cloud computing facilitates to increase the computing capabilities of resource constraint client devices by offering leased infrastructure and software applications. Therefore, Mobile Cloud Computing (MCC) enables computational intensive and ubiquitous mobile applications by leveraging the services of computational clouds. The compact design, resources constraints, mobile nature and wireless access medium features of Smart Mobile Devices (SMDs) require lightweight frameworks for the processing of intensive mobile applications in MCC. Mobile devices are predicated as the dominant future computing devices with high user expectations for accessing computational intensive applications analogous to powerful stationary computing machines. In spite of all the advancements in recent years, SMDs are still low potential computing devices which are limited in memory capacity, CPU speed and battery power lifetime (Shiraz et al., 2012).

MCC extends the services and resources of computational clouds for alleviating the limitations of computing resources in SMDs. MCC utilizes the application processing services of computational clouds for the processing of computationally intensive mobile applications. In MCC, computational offloading is implemented as a software level solution for outsourcing the computationally intensive applications to powerful cloud server nodes. However, leveraging cloud resources and services for mobile devices with lightweight access techniques is highly challenging for the reasons of unique hardware architecture, heterogeneous operating system platforms and the intrinsic limitations associated with wireless network medium. This research focuses on the lightweight frameworks for the processing of intensive mobile applications in MCC.

1.2 Motivation

The report of Gartner Incorporation (Gartner, 2011) states that in the second quarter of 2011 worldwide sale of mobile devices increased 16.5 percent (428.7 million units) as compared to the second quarter of 2010 which is an evidence of the increasing use smart mobile devices. Similarly, the report of Juniper Research (Holman, 2010) states that the consumer and enterprise market for cloud based mobile applications is expected to raise $9.5 billion by 2014 which is an evidence of the increasing use of distributed mobile computing. Recently, a number of computing and communication devices are replaced by smartphones towards all-in-one ubiquitous computing devices such as PDAs, digital cameras, Internet browsing devices, and Global Positioning Systems (GPS) (Prosper Mobile Insights, 2011). Human dependency on the contemporary smartphones is increased rapidly in various domains such as enterprise, e-learning and entertainment, gamming, management information systems, and healthcare (Albanesius, 2011). SMDs are expected to perform intensive computing analogous to their powerful stationary counterparts.

Mobile applications on the latest generation of smartphones and tablets are still constrained by battery power, CPU potentials and memory capacity of the SMDs. Even though mobile hardware technology is developing increasingly, however powerful processing hardware is highly energy consuming. For instance, the processing cycles of CPU, memory refresh instances of primary memory and backlit pixels on the display screen are energy consuming features of mobile device. Therefore, software level solutions are endeavored for augmenting the computing capabilities of SMDs. MCC employs computational offloading as a significant software level solution for alleviating the resources limitations in SMDs.

Recently, a number of computational offloading frameworks are proposed for the processing of computationally intensive mobile applications in MCC(Cuervo et al., 2010; Zhang et al., 2011; Huang et al., 2012). The traditional computational offloading frameworks implement resources intensive techniques for the processing of computationally intensive applications in MCC, which results in high energy consumption and longer turnaround time of the intensive mobile applications. Therefore, lightweight techniques are required for the processing of computational intensive applications in MCC. Lightweight techniques enable computational intensive mobile application deployment and execution with minimal resources utilization on SMDs. As a result, mobile users can utilize distributed services with lower computational load on mobile devices, shorter turnaround time of the application and relatively long lasting battery lifetime. Achieving the aim of lightness in the processing of computational intensive applications for MCC is a challenging research perspective. This research investigates the resources intensive aspects of traditional computational offloading frameworks and proposes lightweight framework for the processing intensive mobile applications in MCC.

1.3 Statement of Problem

Traditional computational offloading frameworks for MCC (Giurgiu et al., 2009; Chun et al., 2011; Cuervo et al., 2010; Zhang et al., 2011; Huang et al., 2012) require the configuration of ad-hoc distributed platform and partitioning of the mobile application at runtime which is resources intensive and time consuming. SMDs are required to select remote server node for each instance of component offloading at runtime, which increases the energy consumption cost and turnaround time of the application. The partitioning mechanism utilizes additional computing resources in runtime application profiling and solving which increases the computational load on mobile device (Satyanarayanan et al., 2009; Dou et al., 2010; Giurgiu et al. 2009; Cuervo et al., 2010; Zhang et al., 2011). As a result, the computing resources (RAM, CPU) and battery of the mobile devices are utilized abundantly and for a longer period of time.

Traditional computational offloading frameworks implement outsourcing of running instances of mobile application (Cuervo et al., 2010; Huang et al., 2012). The technique of outsourcing running instances to cloud server nodes includes the additional cost of saving the data states of the running application on mobile device and reconfiguration of the application on the remote service, which utilizes additional computing resources on mobile device.

The management of runtime distributed platform requires continuous synchronization between local SMD and remote cloud server node. The implementation of uninterrupted synchronization mechanism in the wireless network medium requires keeping SMD in active state which is energy consuming mechanism. Further, traditional computational offloading involves runtime transmission of the binary code of the application and data files which increases the overhead of data transmission over the wireless network medium.

The VM migration based application offloading frameworks (Goyal, 2004; Satyanarayanan et al., 2009; Chun et al., 2009; Chun et al., 2011), involve the overhead of VM deployment and management on SMD which results in additional resources and battery power utilization on SMD. Further, the migration of running instances of the application (partially or entirely) which are encapsulated in VM includes the issue of network attacks vulnerability.

The traditional computational offloading frameworks lack in the consideration of additional resources utilization in runtime component offloading and emphasize on leveraging the Infrastructure as a Service (IaaS) provisioning model for computational offloading which is resources intensive and time consuming. Traditional computational offloading frameworks involve the cost of the migration of application binary file and data file of the running instances of the application. As a result, the size of data transmission over the wireless network medium, turnaround time of the intensive operations and energy consumption cost on mobile device is increased. Hence, the traditional computational offloading frameworks employ heavyweight procedures for the distributed processing of computational intensive mobile applications in MCC.

1.4 Statement of Objectives

We aim at proposing a lightweight framework for the processing of computationally intensive mobile applications in mobile cloud computing. The following are the objectives of the research.

- To review the state-of-the-art for computational offloading in mobile cloud computing.
- To investigate the additional cost (energy consumption, timing, data transmission) of computational offloading in the traditional computational offloading.

- To propose a lightweight framework for the processing of intensive mobile applications in mobile cloud computing.
- To evaluate the proposed framework by testing synthetic workload in the emulation environment and validate the performance by benchmarking prototype application in the real time environment and comparing results of different experimental scenarios.

1.5 Proposed Methodology

We studied the state-of-the-art to identify issues in the current computational offloading frameworks for MCC. The traditional computational offloading frameworks are categorized on the basis of thematic taxonomy (Shiraz et al., 2012). We identify the issues in the traditional offloading frameworks, which hinder the optimization goals of cloud based application processing for MCC.

The research problem is investigated by studying VM deployment for application processes in the simulation environment and implementing traditional runtime offloading technique in the real mobile cloud computing environment. Simulation is performed by using the CloudSim, which is a simulation toolkit for modeling the infrastructure as a service model of the computational clouds. CloudSim is employed for the evaluation of the impact of virtual machine deployment for application processing.

We propose a lightweight Distributed and Elastic Application Processing (DEAP) framework for the processing of intensive mobile applications in MCC. The proposed framework implements a distributed architecture for minimizing the instances of runtime component offloading and implements runtime component offloading for addressing the issue of dynamic processing load on SMD.

The proposed framework is evaluated in emulation and real time mobile cloud computing environment. Synthetic workload is tested on the Android virtual device instance, which is enabled to operate in the distributed mobile cloud computing environment. Prototype application is developed for the Android device, which is tested with varying computational intensities in the distributed mobile cloud computing environment. Experimental results are validated by benchmarking prototype application with different computational intensities in the real mobile cloud computing environment.

The execution behavior of the application is analyzed from the perspective of resources utilization on local mobile device and remote server node, size of data transmission on the wireless network medium, and execution time of the application in the traditional and proposed computational offloading techniques.

Empirical data are collected by testing each component of the prototype application with 30 different computational intensities. The value of sample mean for each experiment is signified with 99% confidence interval for the sample space of 30 values. The lightweight nature of DEAP framework is validated by comparing experimental results in three different execution scenarios of the prototype application.

1.6 Layout of Thesis

This thesis is composed of seven chapters, which are structured as follows.

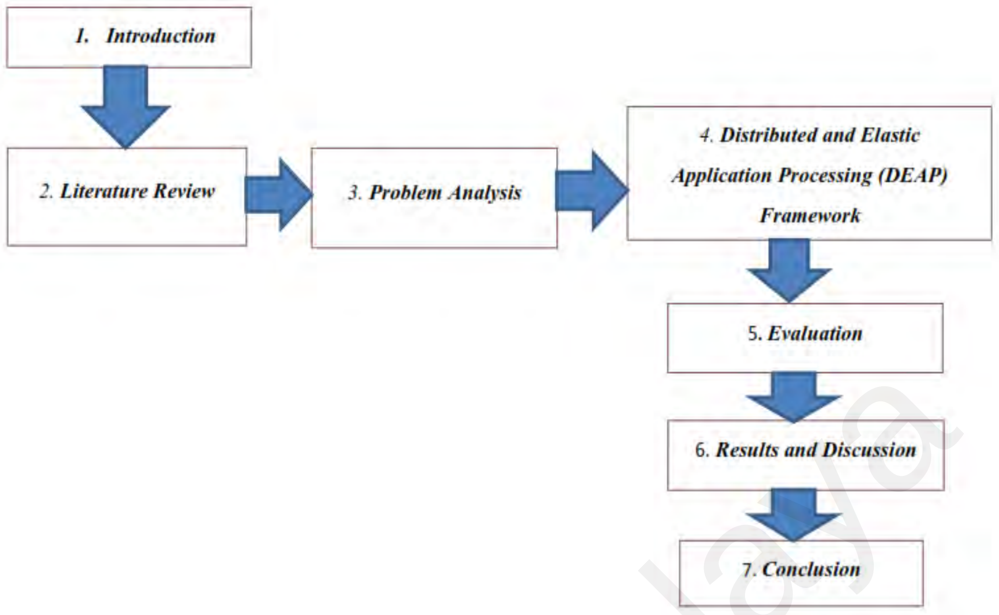
1. Introduction
2. Literature Review
3. Problem Analysis
4. Distributed and Elastic Application Processing (DEAP) Framework
5. Evaluation
6. Results and Discussion
7. Conclusion

Figure 1. 1: Thesis Organization

Chapter 2 presents the epistemology of mobile cloud computing and reviews the state-of-the-art in application offloading for mobile cloud computing. It classifies current offloading models on the basis of thematic taxonomy and compares current frameworks on the basis of significant parameters. The challenges to traditional offloading models and issues in cloud-based application processing for MCC are identified.

Chapter 3 analyzes additional resources utilization in traditional runtime computational offloading by testing the prototype application in the real mobile cloud computing environment. Traditional computational offloading is implemented by offloading the resource intensive service components (sort service and matrix multiplication service) with varying computational intensities to remote cloud server node. The measurement parameters for problem analysis include; energy consumption cost, turnaround time of the component offloaded at runtime and the size of data transmission over the wireless network medium. The cost of virtual machine deployment for application processing is analyzed in the simulation environment by using CloudSim.

Chapter 4 proposes a lightweight Distributed and Elastic Application Processing (DEAP) framework for intensive mobile applications. It explains the architecture of proposed framework, and distinct operating procedures of the proposed framework for the operating modes of the mobile application in accessing the services of cloud server node.

Chapter 5 reports on the data collection method for the evaluation of the proposed framework. It explains the tools used for testing the proposed framework, data collection technique and the statistical method used for the processing of data.

Chapter 6 presents the usefulness of the proposed framework by analyzing the experimental results presented in chapter 5. It discusses the significance of the proposed solution by analyzing the results of the experimentation and comparing the results of different experimental scenarios.

Chapter 7 concludes the thesis by reporting on the reexamination of the research objectives. It explains the findings of the research work, highlights the significance of the proposed solution, states the limitations of the research work and proposes future directions of the research.

CHAPTER 2

Literature Review

This chapter presents theoretical framework for Mobile Clod Computing (MCC), reviews the state-of-the-art and provides thematic taxonomy for current computational offloading frameworks in MCC. The chapter is organized into six sections. Section 2.1 explains the fundamental concepts of cloud computing, mobile cloud computing, computational offloading for MCC and the distributed models deployed for processing of intensive mobile applications. Section 2.2 presents thematic taxonomy of the traditional offloading models, reviews current computational offloading frameworks and investigates the implications and critical aspects of the current offloading frameworks. Section 2.3 compares current computational offloading frameworks by comparing the commonalities and deviations on the basis of significant parameters. Section 2.4 highlights the issues and challenges in computational offloading for MCC. Section 2.5 summarizes the chapter with conclusive remarks.

2.1 Background

This section elaborates the concept of cloud computing and mobile cloud computing. Further, it explains the mechanism of augmenting smartphone through computational clouds.

2.1.1 Cloud Computing

Cloud computing is the distributed computing model that implements the utility computing vision (Buyya et al., 2009), wherein computing services are provided on demand basis. Cloud service models enable with new IT business models such as on-demand, pay-as-you-go, and utility computing. The objective of the cloud computing model is to increase the capacity and capabilities of client devices by accessing leased infrastructure and software applications instead of owning them. Cloud computing has introduced new kind of information and services and new ways of communication and collaboration. Cloud has created online social networks in which scientists share data and analysis tools to build research communities (Kumar et al., 2010; Barga et al., 2011).

In cloud computing, applications are delivered as services over the Internet and user access computing resources from centralized cloud servers through service providers (Armbrust et al., 2009). Computational clouds implement different types of service models for implementing the on demand computing vision (Buyya et al., 2009). Service providers provide services in the form of various service models; Software as a Service (SaaS), Infrastructure as a Service (IaaS), and Platform as a Service (PaaS). Figure 2.1 shows an abstract level layered cloud computing architecture.

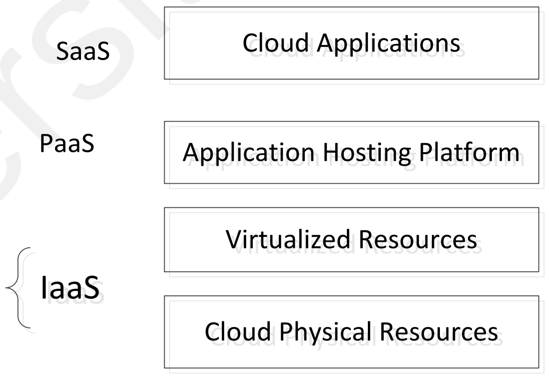

Figure 2. 1: Layered Cloud Computing Architecture

The hardware resources in the cloud datacenters are the physical resources of computational clouds. Access to the physical resources is provided in the form of virtual machines. A middleware (hypervisor) masks access to the physical resources and is responsible for the deployment and management of virtual machines. The application hosting platform is composed of cloud programming environments and tools and monitoring tools such as QoS negotiation, admission control, pricing and billing. The cloud applications run on the virtual machine instances in complete isolation.

2.1.2 Mobile Cloud Computing

Mobile cloud computing is the latest practical computing paradigm that extends utility computing vision of computational clouds to resources constrained SMDs. MCC is a distributed computing model for mobile applications wherein the storage and the data processing are outsourced from the mobile device to resources rich and powerful centralized computing datacenters in computational clouds (Shiraz et al., 2012). The centralized applications, services and resources are accessed over the wireless network technologies based on web browser on the SMDs. Successful practices of accessing computational clouds on demand for stationary computers motivate for leveraging cloud services and resources for SMDs. MCC has been attracting the attentions of business persons as a profitable business option that reduces the development and execution cost of mobile applications and mobile users are enabled to acquire new technology conveniently on demand basis. MCC enables to achieve rich experience of a variety of cloud services for SMD at low cost on the move (Hoang et al., 2011).

MCC prolongs diverse services models of computational clouds for mitigating computing resources (battery, CPU, memory) limitations in SMDs. The objective of MCC is to augment computing potentials of SMDs by employing resources and services of computational clouds (Fernando et al., 2012). MCC focuses on alleviating resources limitations in SMDs by employing different augmentation strategies; such as screen augmentation, energy augmentation, storage augmentation and application processing augmentation of SMD. Abolfazli et al. (2012) highlighted mobile augmentation techniques and proposed a taxonomy including three main approaches, namely high-end resource production, native resource conservation, and resource requirement reduction. A number of approaches have been analyzed and it is argued that MCC lessens need to high-end hardware, reduces ownership and maintenance cost, and alleviates data safety and user privacy.

The MCC model is composed of three major components; SMDs, internet wireless technology and computational cloud. SMDs use wireless network technology protocols such as 3G, LTE, or Wi-Fi to access the services of computational cloud in mobile environment. As SMD inherit its nature of mobility, it needs to execute location-aware services which consume resources and turned it to be a low-powered client. Figure 2. 2 shows a generic model of MCC in which the cloud that provides off-device storage, processing, queuing capabilities, and security mechanism is integrated with SMD via wireless network technologies.

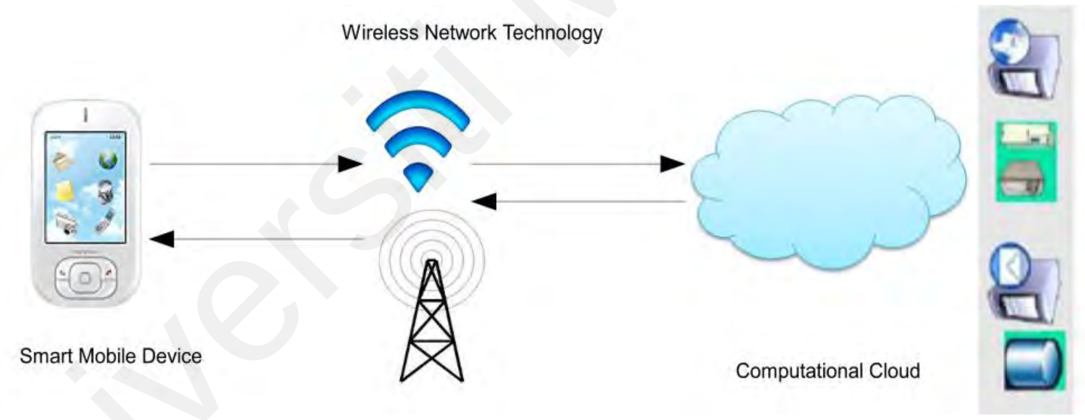

Figure 2. 2: Model of Mobile Cloud Computing (Shiraz et al. 2012)

MCC utilizes cloud storage services (Amazon S3, Google Docs, MobileMe and Dropbox) for providing online storage and cloud processing services for augmenting processing capabilities of SMDs (Zhang et al., 2011). Processing capabilities of SMDs are augmented by outsourcing computational intensive components of the mobile applications to cloud datacenters. The following section discusses the concept of augmenting smartphones through computational clouds.

2.1.3 Computational Offloading for Mobile Cloud Computing

MCC implements a number of augmentation procedures for leveraging resources and services of cloud datacenters. Examples of the augmentations strategies include; screen augmentation, energy augmentation, storage augmentation and application processing augmentation of SMD. In MCC, two categories of the cloud services are of special interest to research community; cloud contents and computing power. Cloud contents are provided in the form of centralized storage centers or sharing online contents such as live video streams from other mobile devices. A number of online file storage services are available on cloud server which augments the storage potentials by providing off-device storage services. Examples of the cloud storage services include Amzon S3 and DropBox. Mobile users outsource data storage by maintaining data storage on cloud server nodes. However, ensuring the consistency of data on the cloud server nodes and mobile devices is still a challenging research perspective.

SmartBox (Zheng et al., 2009) is online file storage and management model which provides a significant approach for online cloud based storage and access management system. Similarly, the application processing services of the cloud datacenters is leveraged by outsourcing computational load to cloud server nodes. The technique of outsourcing computational task to remote server is called computational offloading or cyber foraging. The term "cyber foraging" is introduced by Satyanarayanan (2001) to augment the computing potentials of wireless mobile devices by utilizing the available stationary computers in the local environment. The process of outsourcing computational load to remote surrogates in the close proximity is called cyber foraging (Goyal. et al., 2004). Researchers implement process offloading techniques for Pervasive Computing (Oh et al., 2006) , Grid Computing (Chunlin et al., 2010) and Cluster Computing (Begum et al., 2010). The contemporary approaches for computational offloading in MCC employ the analogous approach of traditional computational offloading for pervasive computing. Mobile applications, which are attributed with the features of runtime partitioning are called elastic mobile applications (Shiraz et al., 2012). Elastic applications are partitioned at runtime for the establishment of distributed processing platform.

Elastic mobile applications are attributed with the following features (Messer et al., 2002) .

- Ad-hoc platform creation is an important attribute of elastic mobile applications. Distributed application processing platform is established on ad-hoc basis at runtime in which elastic mobile application is partitioned dynamically and computational intensive components are migrated to remote server nodes. Mobile clients dynamically arbitrate with cloud servers or surrogates to determine appropriate server node for remote application processing.
- Elastic applications are designed in such a manner so that computational intensive components of the mobile application are separated dynamically at runtime. Applications are partitioned at different granularity level depending upon the design and partitioning policy of the offloading algorithm.
- Adaptive offloading of the intensive components of the applications is a significant attribute of elastic mobile applications. Partitions of the application are offloaded to remote machines for remote execution which augments the computing capabilities of SMDs. Application offloading occurs whereas keeping in view different objective functions; such as energy saving, processing power, memory storage, and fast execution.

- Transparency in the distributed execution platform is a significant attribute of elastic applications. Transparency assures that elastic mobile application executes transparently on remote surrogates/server nodes. A transparent distributed processing environment gives the notion as entire application is being executed locally on SMD. All the complexities of remote execution are concealed from mobile users. Researchers determine applications offloading as an appropriate software level solution for alleviating resources limitations in SMDs.

Currently application offloading is implemented in a number of ways. The application offloading frameworks outsource computational load of SMD at different granularity levels. The static application partitioning approach is used to separate the intensive components of mobile application only once. The dynamic partitioning approach is implemented to address the issue of dynamic application processing load on SMDs at runtime. Dynamic partitioning of the intensive mobile application at runtime is a robust technique for coping with the dynamic processing loads on SMD. In dynamic partitioning application is partitioned dynamically at runtime casually or periodically. In casual partitioning runtime profiling and solving mechanisms are activated in critical conditions to offload intensive components of mobile application. In periodic partitioning the runtime optimization mechanism evaluates computing resources utilization on SMD periodically.

Current dynamic partitioning approaches analyze the resources utilization on SMDs, computational requirements of the mobile application and search for runtime solving of the problem of resource limitations on SMD. The profiling mechanism evaluates computing resources requirements of mobile application and the availability of resources on SMD. In critical condition (the unavailability of sufficient resources on SMD) elastic mobile application is partitioned and the computational intensive components of the application are offloaded dynamically at runtime. SMDs negotiate with cloud servers for the selection of appropriate server node. At that moment, the partitions of the application are migrated to remote server node for remote processing. Upon successful execution of the remote components of the application, result is returned to main application running on SMD.

Empirical analysis ascertains the significance of distributing application processing load to remote server nodes. However, the deployment of distributed application processing platform is obstructed by a number of unresolved challenges for MCC. The traditional and contemporary computational offloading frameworks focus on the establishment of dynamic distributed application processing platform at runtime. For the selection of cloud server node, SMDs arbitrate with cloud server node dynamically at runtime. Therefore, the configuration of distributed processing platform at runtime is a resources starving and energy consuming mechanism. Dynamic runtime offloading involves the issues of dynamic application profiling and solving on SMD, runtime application partitioning, migration of intensive components and continuous synchronization for the entire duration of runtime execution platform. Therefore, the development and deployment of intensive mobile applications on the basis of current algorithms is still a challenging research issue.

2.1.4 Distributed Models for Computational Offloading

Current offloading algorithms employ diverse models for the outsourcing computational load. The following section discusses different mobile computing models, which are employed for application offloading.

2.1.4.1 Local Surrogate Based Distributed Model

In local surrogate based distributed model, SMD is enabled to select an appropriate surrogate for application offloading. A remote surrogate is either a stationary computer or mobile device in local environment. The model implements a centralized server for the establishment of distributed platform and provision of computing resources. Goyal et al. (2004) implement a lightweight cyber foraging framework for outsourcing computational load to surrogates in the localized environment.

2.1.4.2 Mobile Devices Based Ad-hoc Distributed Model

In mobile devices based ad-hoc distributed model virtual or ad-hoc cloud computing environment is established among SMDs in the close proximity. In such an environment, sharing of the computing resources and services is restricted to the computing capabilities mobile devices in the virtual cloud environment. The virtual cloud lacks in the centralized management for the sharing of resources and services. The peer SMDs share computing resources and provides remote service for offload processing. Canepa et al. (2010) implement virtual cloud model for sharing image processing load among peer SMDs in the close proximity.

2.1.4.3 Centralized Server Based Mobile Devices Distributed Model

In centralized server based mobile devices distributed model, remote computing services are provided by mobile worker nodes. However, a centralized server monitors the establishment and management of distributed application execution platform. In such a computing model, distributed resources and services provision are restricted to the computing potentials and services of worker nodes (SMDs). Dou and Kalogeraki (2010) implement MapReduce model for sharing the computational workload among mobile worker nodes.

2.1.4.4 Cloud Datacenters Based Distributed Model

The cloud datacenter based distributed model is composed of centralized monitoring mechanism for providing access to shared resources and services. The service providers provide access to the widespread services on demand basis. Zhang et al. (2011) implement elastic application model for outsourcing application partitions to cloud server nodes. Figure 2.3 highlights mobile computing models which are employed for application offloading.

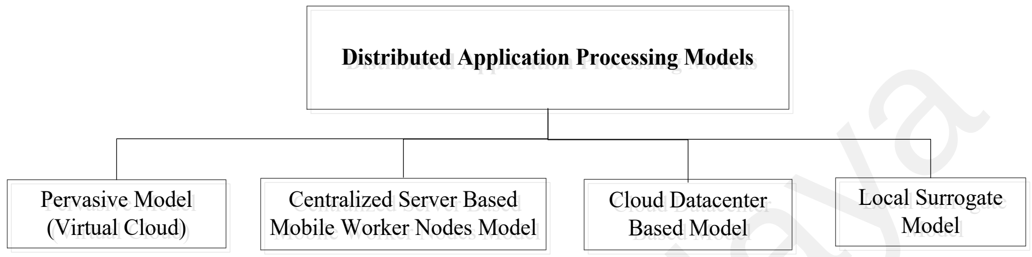
Distributed Application Processing Models
Pervasive Model (Virtual Cloud)
Centralized Server Based Mobile Worker Nodes Model
Cloud Datacenter Based Model
Local Surrogate Model

Figure 2. 3: Distributed Models for Computational Offloading

2.2 Review of Traditional Computational Offloading Frameworks

This section classifies the traditional computational offloading frameworks on the basis of thematic taxonomy and analyzes the implications and critical aspects of the traditional Distributed Application Processing Frameworks (DAPFs). Section 2.2.1 explains the taxonomy of the traditional offloading models and section 2.2.2 reviews the traditional offloading frameworks on the basis of thematic taxonomy.

2.2.1 Taxonomy of Distributed Application Processing Frameworks for MCC

Thematic taxonomy is derived on the basis of the following attributes: framework nature, migration pattern, migration support, partitioning approach and objective functions. This section presents an introduction to the attributes of the thematic taxonomy. Section 2.2.2 reviews current DAPFs on the basis of framework nature, whereas section 2.3 contains a detailed discussion on objective functions, migration pattern, migration support, migration granularity and partitioning approaches. Section 2.3 compares the traditional DAPFs on the basis of the parameters presented in the taxonomy.

2.2.1.1 Framework Nature

The nature of a framework represents the primary mechanism employed for the establishment of runtime distributed platform in offload processing. We categorize current DAPFs on the basis of virtual machine migration, entire application migration and application partitioning.

a) ***Virtual Machine Migration***: Virtual machine migration nature of DAPFs indicates that SMD offload mobile application (partially or entirely) by encapsulating the offloading component in VM instance on SMD. In VM migration based offloading mechanism the data states of the running application are saved (Chun and Maniatis, 2009) or the entire image of the running application is encapsulated in the VM instance (Satyanarayanan et al., 2009). The instantiated VM instance is migrated to the remote server nodes. A number of current DAPFs employ VM migration based approach for computational offloading (Hung et al., 2011; Zao et al., 2011).

b) ***Entire Application Migration*****:** Entire application migration nature of DAPFs indicates that SMD offload entire processing job to remote server nodes. Current DAPFs offload entire application in two different manners. Running application is offloaded to remote server node (Canepa and Lee, 2010) or entire job is offloaded to remote server for outsourced processing (Liu et al., 2009).

c) ***Application Partition*****:** Application partitioning nature of the framework indicates the elastic nature of offloading framework. Elastic mobile applications have the attributes of dynamic runtime partitioning. The intensive partitions of the application are offloaded to remote server nodes at runtime (Cuervo et al., 2010; Zhang et al., 2011).

2.2.1.2 Partitioning Approach

The partitioning approach of a framework indicates the mechanism of separating intensive components of the application. Current DAPFs implement runtime application partitioning in two different manners; static partitioning and dynamic partitioning. In static application partitioning the application partitioning logic is implemented only once either at compile time or runtime. The partitioning mechanism separates the locally annotated components for local execution on SMD, whereas the remotely annotated components are offloaded to remote server nodes.

The dynamic partitioning mechanism follows dynamic evaluation mechanism for the evaluation of the computational load on SMD. In dynamic partitioning, the application partitioning algorithm continuously monitors the statistics of the resources allocated to mobile application on SMD. In critical conditions (availability of low computational resources on SMD) the pre-identified intensive (remotely annotated) components of the mobile application are offloaded to cloud server nodes.

2.2.1.3 Migration Support

The attribute of migration support indicates the level of support required for migrating application or partitions of the application at runtime. Currently, migration support is provided either at system level or application level. In system level migration the support of additional services is required on local operation system. For example the VM migration based offloading requires the additional support for VM deployment on SMD. In application level migration, the offloading logic is implemented at the application layer and does not require additional support at the operating system level.

2.2.1.4 Migration granularity

The migration granularity attribute represents the granularity level at which application is migrated. Current DAPFs offload intensive components of the application at different granularity level. For example thread level granularity indicates that running thread is offloaded for remote processing. In the same way, method level granularity indicates that methods of the application are offloaded for remote processing.

2.2.1.5 Migration pattern

The migration pattern attribute represents the mechanism for transferring mobile application to remote server node. Current DAPFs employ a number of migration patterns such as VM migration, download using URL on remote host, mobile agent serving as courier for application transfer, binary code transfer of the application or copying entire proxy of the application on distributed computing nodes.

2.2.1.6 Objective functions

The objective function attribute indicates the primary objective of a framework for application offloading. Traditional application offloading frameworks implement a number of objective functions for making the decision of application offloading; such as saving energy on SMD, efficient bandwidth utilization, saving processing power on SMD, user preferences for fast application processing, or execution cost parameter. Figure 2.4 shows the thematic taxonomy of current DAPFs for MCC.

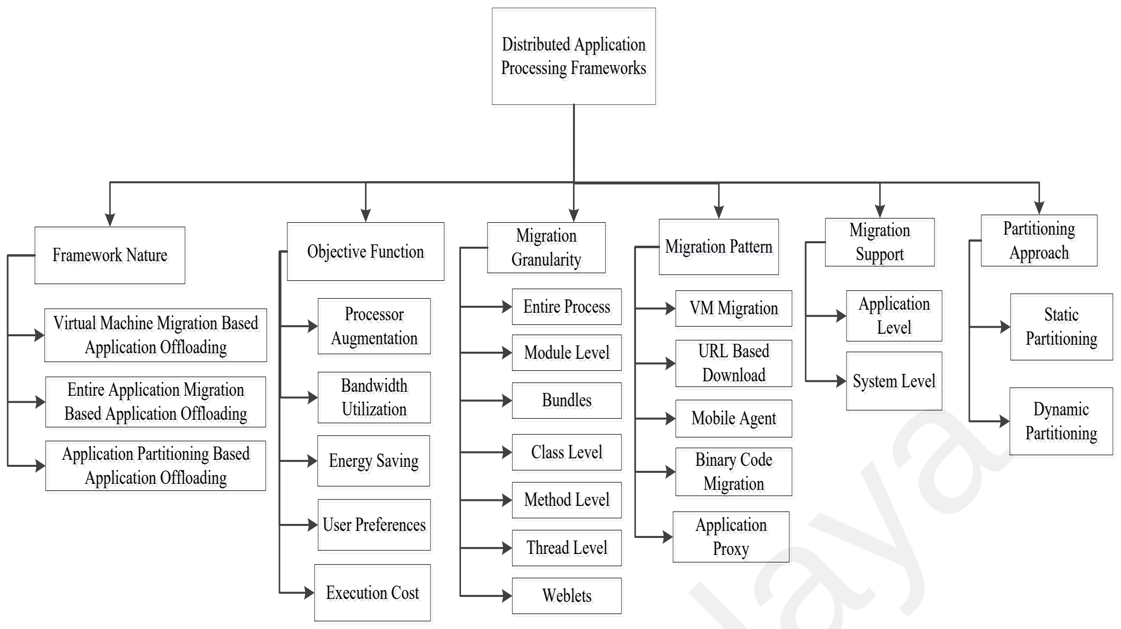

Figure 2. 4: Taxonomy of Mobile Application Offloading Frameworks for MCC (Shiraz et al., 2012)

2.2.2 Review on Distributed Application Processing Frameworks

This section reviews current DAPFs on the basis of framework nature presented in Figure 2.4. It also investigates the implications and critical aspects current DAPFs.

2.2.2.1 VM Migration Based Computational Offloading

Cyber foraging framework (Goyal et al., 2004) is employed to utilize computation resources of the computing devices (stationary or mobile) in the close proximity of the SMD. The framework implements client/server architecture. Mobile devices request for process offloading and surrogate server provides the services on demand. The framework supports configuration of multiple surrogate servers simultaneously and employs virtual machine technology for remote application processing. A single surrogate server is capable to run a number of independent virtual servers in a controlled manner and simple cleanup mechanism is employed for releasing the resources allocated to VM instance. Each offloaded application executes on isolated virtual server. The framework ensures secure communication by deploying cryptographic measures for communication between SMD and surrogate server.

The framework includes the benefits of low latency, local accessibility of remote surrogates and fewer concerns of security and privacy. The critical aspects of such approach is the deployment of template based virtualization approach which is a highly time consuming and resources starving mechanism for VM deployment (Wang et al., 2011). The framework requires the annotation of individual components of the application as local or remote which is an additional effort for the application developers. Further, surrogate based cyber foraging is restricted to the availability of services and resources on local servers.

VM based cloudlets framework (Satyanarayanan et al., 2009) differs from cyber foraging (Goyal and Carter, 2004) by migrating image of the running application to the explicitly designated remote server. A cloudlet is a trusted resource rich computer or cluster of computers which is accessible for SMDs. In the proposed model SMDs are employed as thin client which implement user interface components of the mobile application. The actual application processing is performed on the cloudlet in distributed environment. Virtual machine technology is deployed to rapidly copy customized service software on a nearby cloudlet and access the service in ubiquitous local area network environment.

The framework implements hardware supported VM technology for the customization of cloudlet infrastructure. VM instance in the cloudlet machine separates the delegated guest application processing environment from the cloudlet infrastructure's permanent host software environment. The framework employs different procedures for VM migration. The critical aspects are that the framework requires additional hardware level support for the implementation of VM technology and is based on cloning mobile device application processing environment to remote host. The mechanism of transferring the entire image of mobile application involves the issues of VM deployment and management on SMD, privacy and access control in migrating the entire execution environment and security threats in the transmission of VM instance.

Chun and Maniatis (2009) proposed a clone cloud based framework which is a significant approach for offloading different types of mobile applications. Clone cloud differs from other approaches (Goyal and Carter, 2004; Satyanarayanan et al., 2009) by employing three different offloading algorithms for different types of applications. However, the attribute of offloading image of the running states of the application to remote server resembles to the VM based Cloudlet approach (Satyanarayanan et al., 2009). The framework reduces the dynamic transmission overhead of application code by deploying a simple approach for synchronization.

Clone cloud employs the mechanism of primary functionality outsourcing by offloading computational intensive tasks to remote host whereas simple tasks such as user interfaces are executed on mobile devices. Primary functionality outsourcing strategy is useful for applications which involve two types of processing; user interfaces which are displayed on the mobile devices, and high power resource starving computation. Example of the application includes speech recognition, image processing and video indexing. Background augmentation mechanism offloads the entire application to remote host and returns result from the background process to the mobile device. The background augmentation strategy is useful for the applications which are composed of intensive processing loads and do not require frequent user interactions. Example of the applications includes scanning of files for viruses, indexing files for faster search. In these scenarios entire process is marked as remote by either; programmer, user or automatically inferred as background process at runtime. Mainline augmentation strategy is implemented which are resources intensive and requires frequent user interactions. Such type of applications need to interact with other parts of the applications or users and having some computational intensive load. Examples of the applications include; fault tolerance and debugging.

Clone cloud is a significant framework for offloaded processing which includes a simple approach for synchronization between SMD and remote server. The critical aspect of the Clone cloud is the migration of the execution environment of the application on SMD to remote server which involves the issues of security, privacy, access control, and the complications of VM deployment and management on SMD. The deployment of variant strategies for application migration on the basis of application nature results in enlarged overhead on mobile devices. Clone cloud deploys a single thread approach which increases jitter in the execution time of the application components.

The elastic CloneCloud (Chun et al., 2011) extends the concept of local Clone cloud (Chun and Maniatis, 2009) to remote cloud datacenters. The framework is based on partitioning of the application on thread basis and requires application level support for the establishment of distributed application processing platform at runtime. The framework is implemented in two phases; partitioning and migration. The partitioning phase determines the candidate intensive threads of the mobile application which are required to be migrated to the cloud server for remote execution. The partitioning phase involves static analysis, application dynamic profiling, and optimization solution. Static analysis is performed to identify migration and reintegration points in the code. The framework deploys some constraints for partition migration and ensures to follow the constraints for identifying migration and reintegration points in the application.

In CloneCloud the partitioning and integration of the application occurs at application level. The running data states of the outsourcing components of the mobile application are encapsulated in VM instance and VM migration is employed for partition migration to cloud server node. The framework employs a centralized monitoring mechanism for the establishment and management of distributed application execution platform. CloneCloud is a significant approach for extending the concept of VM based offloading from local distributed platform to centralized cloud servers. The framework considers the objective functions of application execution time and energy consumption at mobile device.

CloneCloud implements a complicated architecture on SMD for the establishment and management of distributed platform. The framework is based on VM instance migration to the cloud node which involves the concerns of secure communication of running application states encapsulated in VM and privacy and access on remote server node. A major limitation of the architecture is that a single thread is migrated to the cloud at a time which reduces concurrency of the execution of application components.

Hung et al. (2012) propose a virtualized execution environment for mobile applications. The framework utilizes application level process migration and employs Android platform for distributed application deployment. A running application is encapsulated in VM on SMD and VM is migrated to remote cloud computing environment. Cloud server creates fresh VM instance, and the delegated application VM is cloned into the newly created VM instance on the cloud server node. A synchronization mechanism is provided between SMD and cloud server. A middleware is placed between mobile device OS and hardware to support runtime workload migration and to better utilize the heterogeneous resources of mobile device and cloud servers. The framework deploys pause/resume scheme of the android platform for transferring . The framework employs application level process migration strategy for offload processing and employs hardware base trusted platform module. The framework provides mechanism for storing encryption keys and performs cryptographic operations on sensitive data. The critical aspects are that the framework requires heavy and traffic intensive synchronization mechanism for ensuring consistency between SMD and cloud server. The framework necessitates a separate program called agent to be installed on SMD and cloud server which results in additional overhead on SMDs.

Mirror server (Zao et al., 2011) is a distinct augmentation framework which employs Telecommunication Service Provider (TSP) based remote services. A TSP is a type of communications service provider which provides voice communication services such as land line telephone and cellular phone call services. The framework leverages cloud computing virtualization technique for the deployment of mirror server. A mirror server is a powerful server configured in TSP backbone which maintains VM template for different mobile devices platforms.

The VM template for each mobile device is kept with default settings. A new VM instance is created for offloaded component of the mobile application. The VM template for each mobile device is called its mirror and the server responsible for the deployment and management of the mirrors is called mirror server. The server creates fresh VM instance as per the platform of the requesting SMD. Mirror server is scalable and is capable to create hundreds of mirrors at a time. Mirror server augments smartphones by providing three different types of services; security (file scanning), storage (file caching) and computation offloading.

The significant aspect of mirroring smartphone is that it provides reliable services through 3G network and addresses the challenging aspect of heterogeneity in SMDs platforms. The framework provides a lightweight protocol for SMDs for accessing remote services on mirror server and employs an optimized mechanism for downloading and offloading. The critical aspects of mirroring based DAPF is the deployment of TSP based mirror servers which are not basically designed for data processing, for that reason limited services can be acquired as compared to cloud datacenters.

The following section describes the generic sequence of operations for VM migration based application offloading. (a) The first step for application offloading is to arbitrate for appropriate surrogate or remote server host. Subsequently, the running application is encapsulated in VM on SMD which involves the creation of VM instance, VM configuration for running application and encapsulating all the state information of running application in VM instance. (b) The VM instance is migrated to the remote server through wireless medium. A new VM instance is created on remote server and the migrated VM is cloned onto the newly created VM instance on remote server. Running states of the application are resumed and application is executed on remote server host. Finally, results are returned to the SMD. (c) Remote server ensures complete isolation of guest VM which means that the executing environment of guest VM is prevented from interference. Figure 2.5 shows abstract level flowchart of VM migration based application offloading.

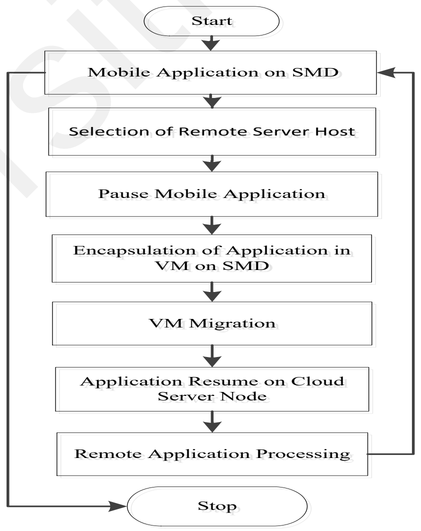
Start
Mobile Application on SMD
Selection of Remote Server Host
Pause Mobile Application
Encapsulation of Application in VM on SMD
VM Migration
Application Resume on Cloud Server Node
Remote Application Processing
Stop

Figure 2. 5: Generic Flowchart for the VM Migration Based Application Offloading (Shiraz et al., 2012)

Virtual machines lead to high CPU utilization. VMs share the same CPU/core which increases the CPU scheduling latency for each VM (Wang et al., 2011). VM migration based offloading requires additional computing resources and time for the deployment and management of VM on SMD. As a result, such approaches increase the execution cost and time of the application. Migration of the running application along with its data states is susceptible to security breaches and attacks. Further, a number of other research challenges such as privacy and access control are still addressable which obstruct the goals of optimal VM based migration algorithms for MCC (Shiraz et al., 2013).

2.2.2.2 Entire Application Migration Based DAPFs

Lightweight secure cyber foraging infrastructure (Goyal et al., 2004) employs Virtual Server Manager (VSM) which handles requests from SMDs for surrogate operations. SMD sends a request to VSM which is composed of URL to the program to be executed on surrogate. The entire program is downloaded on that URL and executed remotely. The background augmentation strategy of Clone cloud (Chun et al., 2009) employs entire application migration to remote host. The application is migrated to remote local servers using VM instance migration and results are returned from background process to the mobile device.

Canepa et al. (2010) propose the virtual cloud computing provider solution for mobile devices which is an ad-hoc cloud framework. The virtual cloud model focuses on the establishment of virtual cloud of SMDs. The virtual cloud computing environment is composed of SMDs in the proximity which remains in the same locality and stable mode. Mobile devices in the proximity set up an ad-hoc or virtual cloud environment and enables SMDs in the vicinity to share the computational load. The framework is composed of different components. The context manager component of the architecture maintains information regarding volunteer SMDs for resource sharing. The offloading manger component is responsible for sending and receiving entire applications, management of runtime distributed environment and detecting failure and failure management. Offload manager coordinates with p2p component for application offloading and returning results.

Universal Mobile Service Cell (UMSC) based framework (Liu et al., 2009) is a unique mobile agent based optimization solution which focuses on virtual cloud of mobile devices. The distinguishing features of the framework are the employment of mobile agent (UMSC) for application offloading and virtual cloud based service provision. The proposed architecture is composed of mobile hosts, UMSC, and mobile cloud units. Cloud unit support several services such as offloaded computing and remote storage. The mobile cloud is composed of two kinds of cloud units such as local cloud unit and remote cloud unit. The framework uses UMSC for the offloading of entire application to remote cloud unit. UMSC serves as a mobile agent and works as a proxy for transmission between mobile cloud and mobile host. UMSC is implemented as an intelligent software module which carries the requests of users. UMSC does not send request or responses to the network; instead UMSC itself migrates into the cloud to search response. The framework is composed of mobile agents in local mobile cloud computing environment for mobile devices and uses a genetic algorithm based scheduling policy for UMSC. The mobile environment is divided into a large number of cell regions. Each cell region is composed of several mobile cloud units. The cloud units in the cloud regions collectively form the virtual mobile cloud environment. Cloud units are the mobile support stations for providing services.

The framework addresses the intrinsic issues associated with mobile computing; mobility, heterogeneity, and low bandwidth. UMSC based approach is a hybrid solution that combines mobile agent technology with virtual cloud model which is composed of SMDs. UMSC employs mobile IP to compensate the problem of mobility and provides a mechanism to overcome the problem of mobile host disconnection. UMSC guarantees the quality and stability of wireless connections. The critical aspects are that distributed services are restricted to the availability of mobile nodes in virtual distributed wireless environment. The framework exploits localized approach for accessing distributed resources and involves a decentralized monitoring mechanism on SMDs which increases the demand for computing resources on SMDs. The framework implements management of mobile agents on mobile devices, which is a sophisticated and resource consuming mechanism.

Chung et al. (2010) propose Distributed Shell System (DISHES) which is the extension of UNIX kernel shell to support ubiquitous distributed computing platform for SMDs. The architecture is composed of a centralized server, which contains a Service Directory (SD). The ambient computers register with the server which employs SD for maintaining database of all the nodes which are willing for sharing resources. Mobile clients make use of SD services for tracking appropriate remote server. DISHES serves as an interface middleware between a mobile user and network computers. SMD makes request for the availability of remote host for application processing, SD responds with the IP address of appropriate volunteer remote host for application process. SMD offloads entire application to remote host for remote processing and results are returned to SMD on successful completion of the remote processing.

DISHES includes performance optimization mechanism to monitor network traffic and provides remote execution services in transparent manner. The critical aspects of DISHES are the decentralized approach, unavailability of centralized mechanism for the establishment and management of distributed platform and entire application migration for offloaded processing. DISHES imposes additional assistant process creation on SMDs which involves intensive monitoring overhead on SMDs.

Dou and Kalogeraki (2010) propose Misco which extends the concept of MapReduce to the distributed cloud environment which is composed of centralized server and mobile worker nodes. MapReduce is a flexible distributed data processing framework which automatically parallelizes the processing of long running applications in cluster environment (Dean and Ghemawat, 2004). In Misco, the master server is a centralized monitoring entity which is responsible for the implementation of MapReduce framework. A distinctive feature of the framework is that SMDs are the worker nodes which serve as serving components for remote application processing. The worker nodes coordinate with the master server for getting workload and returning result. The communication between worker and master server occurs through HTTP Server. The download and upload between master server and workers is performed in the form of XML files. Application developers identify the Map and Reduce functions during the application development process.

The framework provides a distributed platform for mobile applications. Misco provides a centralized monitoring mechanism for monitoring of the distributed execution platform. The critical aspects are that the framework consists of worker mobile nodes which are intrinsically resources poor and therefore the availability of computing services is restricted to the computing potentials of SMDs. Misco requires the developers to annotate the methods as map or reduce functions and does not perform any centralized processing of application which results in communication overhead repeatedly between worker nodes and master server. Communication overhead increases jitter in the process execution, bandwidth consumption and energy consumption.

Liu et al. (2010) implement privacy algorithm for offloading entire image object to grid power server nodes. It highlights the tradeoff between energy savings and privacy protection in offloading processing. Stenographic techniques (Nguyen, et al., 2006) are explored for disguising actual image from grid powered servers. The authors focus on the fact of privacy based offloading in which the contents of the offloaded components are hidden from the cloud node. The authors investigate the tradeoff between energy consumption and privacy of offloading and performed analysis of different execution patterns of the applications. Different parameters are involved in the energy consumption of mobile application processing; computation power of mobile, network power, idle power of mobile device, the speed of mobile system, speed of server, and bandwidth of network.

Iyer et al. (2011) propose Cogniserve which focuses on the evolving feature of Mobile Augmented Reality (MAR) for image processing and speech recognition applications. Cogniserve architecture is composed of three main components; application cores for processing over cloud server, application specific recognition accelerators for performance improvement and decreasing latency, architectural support for general purpose programming and efficient communication between small cores and accelerators. The recognition server is composed of small cores connected via interconnect to an integrated memory controller for attaching it to DRAM. The design is composed of simple chip multi-processor. Application specific accelerators are used to further enhance the recognition execution time. Three types of accelerators are deployed in the architecture; Gaussian mixture model for speech recognition, match accelerator and interest point detection for image recognition. The resulting architecture is heterogeneous by integrating small cores.

CogniServe deploys the concept of instruction set architecture, in which there is direct user access from the small core to accelerator. As a result, the user to kernel mode transitions is eliminated. The architecture also utilizes the concept of common memory management units which lead the accelerator to share virtual memory space with the core; as a result of this it eliminates data movement overhead with the kernel to user mode transition. The framework deploys heterogeneous server architecture for recognition applications of mobile devices. CogniServe provides direct access between server cores and accelerators instead of kernel to user space transition which eradicates address space transition and results in a low cost and energy efficient architecture. The critical aspects are that the framework requires special hardware level support for the implementation and is specially designed for recognition applications such as image processing and speech recognition applications.

The virtualized framework (Hung et al., 2012) employs application level process migration and uses Android platform for the deployment. A running application is encapsulated in VM on SMD and VM is migrated to remote cloud computing environment for remote processing. Mirror server based approach (Zao et al., 2011) involves the migration of the state of the entire running application on the smartphone device to mirror instance on server. Application is executed in the mirror VM instance and result is return to the smart mobile device. Figure 2.6 shows the generic flowchart of offloading entire application/job to remote server node.

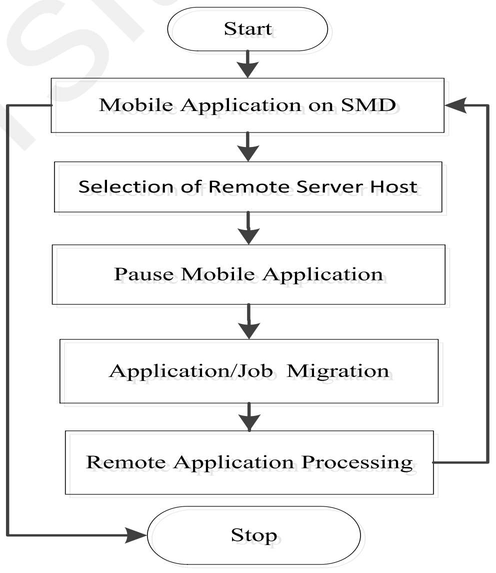
Start
Mobile Application on SMD
Selection of Remote Server Host
Pause Mobile Application
Application/Job Migration
Remote Application Processing
Stop

Figure 2. 6: Generic Flowchart for Entire Application Migration Based DAPFs (Shiraz et al., 2012)

SMD arbitrate with cloud server nodes for the selection of remote server node, at that moment entire application or job is migrated to remote server node. Upon the successful execution of the application on remote server ultimate results are returned to SMD.

2.2.2.3 Application Partitioning

The latest DAPFs employ application partitioning based offloading mechanism for outsourcing the intensive components of the mobile application to remote server node. Current partitioning algorithms employ runtime application profiling and solving mechanism for evaluating the computational load of SMDs. Elastic application frameworks employ different objective functions for the identification of intensive components of the mobile application and making the decision of application offloading. Application partitioning algorithms are classified in two broad categories; static partitioning and dynamic partition. The following section reviews existing elastic application framework for MCC on the basis of aforementioned application partitioning approaches.

2.2.2.4 Static Application Partitioning Based DAPFs

In static application partitioning the mobile application is partitioned in fixed number of partitions either at compile time or runtime. The computational intensive partitions of the applications are outsourced to remote servers. In the primary functionality offloading (Satyanarayanan et al., 2009) application is statically partitioned in two major partitions. Such applications involve two types of processing; user interface which are required on mobile device; and computational intensive parts of the application are offloaded to remote surrogates or cloud servers. In Misco (Dou et al., 2010) the application is statically partitioned into two types of functions; map and reduce. Map function is applied on the set of input data and produces intermediary <key, value> pairs; such pairs are grouped into a number of partitions. All pairs in the same partition are passed to a reduce function which produces the final results. Application developers are responsible for implementing the map and reduce functions and the system handles all the remaining mechanism. The worker nodes process map and reduce functions and results are returned to master server.

2.2.2.5 Dynamic Application Partitioning Based DAPFs

Dynamic partitioning of the intensive mobile application at runtime is a robust technique for coping with the dynamic processing loads on SMD. Current dynamic partitioning approaches analyze the resources consumption of SMDs, computational requirements of the mobile application. Such frameworks search for runtime solving the critical conditions of resources shortage on SMD. In dynamic partitioning application is partitioned dynamically at runtime casually or periodically. In casual partitioning runtime profiling and solving mechanisms are activated in critical conditions to offload intensive components of mobile application. In periodic partitioning the runtime optimization mechanism evaluates computing resources utilization on SMD periodically. Dynamic partitioning of the mobile application is implemented in different manners. In the follow section we review current dynamic application partitioning frameworks for MCC.

AIDE (Messer et al., 2002) establishes distributed platform which is composed of different computing devices such as laptops, PC's, PDA's, and smartphones. The framework is composed of surrogate server and mobile device client. SMD searches for suitable surrogate to share application processing load. The partitioning component of the AIDE partitions the application by following a partitioning policy. The framework exercises class level granularity for the partitioning of elastic mobile application. The application profiling component establishes the feasibility of offloading. Application profiler reflects on two parameters, the execution history of the application and prediction of the future resources required for the application. Profiler aims for offloading the components that could improve the performance of the system.

AIDE provides a transparent distributed application deployment framework for mobile applications. The sophistication of application migration and remote execution are masked from mobile users. AIDE employs a dynamic partitioning and migration approach for offloaded processing and employs computing services of the remote hosts in the local distributed environment. AIDE implements distributed execution platform in transparent manner and gives the notion of application being executed on local device. AIDE incorporates the option to use multiple surrogates for remote execution. The critical aspects of AIDE are that the runtime partitioning of the application requires additional computing resources exploitation for the establishment of distributed platform. AIDE is a decentralized distributed platform for dynamic partitioning and migration, therefore heavy monitoring overhead is implemented on SMD.

Giurgiu et al. (2009) introduced a middleware framework for sharing the application processing load on SMD dynamically between cloud server node and mobile devices. Objective of the framework is to deploy the application in optimal mode by automatically and dynamically determining the execution location for modules of an application. Application profiling component of the architecture partitions the application in modules on the basis of its behavior and represents modules in the form of data flow graph known as consumption graph. The framework exercises existent module management such as R-OSGi (Rellermeyer et al., 2008) and deployment tool such as AlfredO (OSGi Alliance, 2007). The framework implements both static partitioning and dynamic partitioning strategies. K-Step and ALL algorithms are used for application partitioning. K-Step is deployed for dynamic partition at runtime, whereas ALL is employed for static partitioning of the application. Preprocessing of the consumption graph is performed before running the algorithm to reduce search space. Preprocessing separates local and remote bundles of the application. The framework looks for outsourcing the components of mobile application which are feasible for offloading. It means that the intensive components of the mobile application with higher offloading cost than local execution are not offloaded.

The implications of the framework are that it employs both static and dynamic partitioning algorithms for the establishment of runtime distributed platform between SMDs and cloud datacenters. A significant aspect of the framework is that SMDs are assigned application processing load on the basis of the availability of memory and processing capacity. The framework derives optimal solution for optimization problem in order to optimize different objective functions such as interaction time, communication cost, and memory consumption. The critical aspect of the framework is the runtime partitioning strategy which puts additional computational load on SMDs in dynamic analysis, profiling, synthesizing, partitioning and migration. The framework requires SMDs to continuously synchronize with the cloud server node which requires maintaining SMD in active state for the entire duration of distributed platform which is an energy starving mechanism (Kelenyi et al. 2009; Pedersen, 2009).

Mobile Assistance Using Infrastructure (MAUI) (Cuervo et al., 2010) is a dynamic partitioning framework which focuses on energy saving for the SMD. MAUI partitions the application dynamically at runtime in which the computational intensive components of the application are offloaded to the cloud server nodes. Programmers annotate the individual methods of the application as local or remote. MAUI profiler determines the remote methods of application to be offloaded to cloud server. Whenever a method is called, the profiler component evaluates it for energy saving which consumes additional computing resources (CPU, energy) on SMD.

MAUI solver decides the destination location for the execution of the method annotated as remote. The decision of MAUI solver is based upon the input of MAUI profiler. Proxies of the application are created for execution on both cloud server node and mobile device for communication between local methods and remotely executable methods. MAUI generates a wrapper for each method marked as remote at compile time. The type signature of the wrapper methods differs from two perspectives; one additional input argument, and one additional return type. Input argument is required for the state transfer of smartphone to MAUI server through client application proxy. The additional return value is used to transfer the application state back to smart mobile device using server proxy. State of the method is transferred in serialized form.

MAUI is a cloud server based dynamic partitioning framework which considers energy saving on SMD as the main objective function for offloaded processing. MAUI masks the complexity of remote application execution from mobile user and gives the notion as the entire application is being executed on SMD. The framework is based upon method state migration as a substitute of method code migration. MAUI copes with the mobility of the mobile user and provides optimized solution periodically to adapt to the changes in network and user location.

The critical aspect of MAUI is the dynamic partitioning of the application at runtime which activates the profiler and solver component dynamically to determine execution point for application partitions. Development of the applications on the basis of MAUI requires additional developmental efforts for annotating the execution pattern of each individual method the application. MAUI deploys full proxies of the application on both SMD and cloud datacenter. MAUI obliges the overhead of dynamic application profiling, solving, partitioning, migration, and reintegration on SMD.

CloneCloud (Chun et al., 2011) employs dynamic partitioning of the application at runtime. Partitioning phase of the framework involves; static analysis, application dynamic profiling, and optimization solution. Mobile device uses preprocess migratory thread to assist a process with suspending, packaging, resuming and merging thread states. Chun et al. (2010) address the issue of application partitioning between mobile devices and clouds. The optimization problem is modeled through a mathematical expression, which includes execution cost of each module on mobile device, execution cost of each module on cloud, and the cost of communication between the two modules. Variant objective functions are considered for partitioning, minimize execution time and minimize battery power consumption or cost of execution of the application.

Zhang et al. (2011) propose elastic application model for augmenting the computing capabilities of mobile devices. Application is partitioned into weblets and migrated dynamically between mobile device and remote cloud server. Variant elastic patterns are used for the replication of weblets on the remote cloud. The execution destination for the weblet is determined dynamically at runtime. The framework employs different parameters for offloaded processing of the weblets such as status of the mobile device, cloud, application performance measures and user preferences which comprise power saving mode, high speed mode, low cost mode and offload mode. The framework implements an optimal cost model for the execution configuration of the weblets. The cost model considers different costing factors such as power consumption, monetary cost, performance attributes and security and privacy.

The framework proposes a security mechanism for ensuring the integrity of communication between SMD and cloud server (Zhang et al., 2009). Whenever a weblet is downloaded on SMD, the integrity of each weblets is ensured by the installer of the device by re-computing hash value for each weblet and comparing it with the hash value stored in the weblet. The installer registers the application with Device Elasticity Manager (DEM). The DEM maintains a table of installed applications on the device which need elasticity manager support. The table maintains detailed information about weblets such as signed hashed values and migration settings. Several parts of the elastic application are installed on Cloud Elasticity Service (CES). CES maintains installed applications for users. For this purpose users register with CES and authenticate with CES during installation. The cloud based application manager is able to download the application code from an application store instead of uploading from mobile device. The node manager executes the weblet binary provided by application manger. The local weblet can query DEM to obtain the list of all active weblets in the same session. The local weblet can broadcast the URLs returned by DEM to any other weblet that needs to communicate.

The implications of elastic application model are that it accomplishes application level partitioning and migration of applications. The framework employs a comprehensive cost model to dynamically adjust execution configurations and optimizes application performance in terms of a set of objectives and user preferences. The framework provides a security mechanism for the authentication and authorization of weblets migration and reintegration and provides support for synchronization between application on mobile device and weblets running on cloud node. The critical aspect is the establishment of runtime distributed platform for SMD which requires additional computing resources for the establishment and management of distribute platform. The framework deploys replication of the application both on the mobile device and application manager of the cloud server. The framework implements a sophisticated mechanism for the migration of weblets between SMD and remote cloud nodes. It imposes extensive overhead of application profiling, dynamic runtime partitioning, migration, reintegration, and rigorous synchronization on mobile devices for offload processing.

Figure 2.7 shows a generic flowchart for application partitioning based offloading frameworks. The profiling mechanism evaluates computing resources requirements of mobile application and the availability of resources on SMD. Profiling mechanism works differently in different frameworks. The critical situation indicates the unavailability of sufficient computing resources on SMD. The computational intensive components of the application are separated at runtime. SMD negotiate with cloud servers for the selection of appropriate server node. The partitions of the application are migrated to remote server node for remote processing. Upon successful execution of the remote components of the application, result is returned to main application running on SMD.

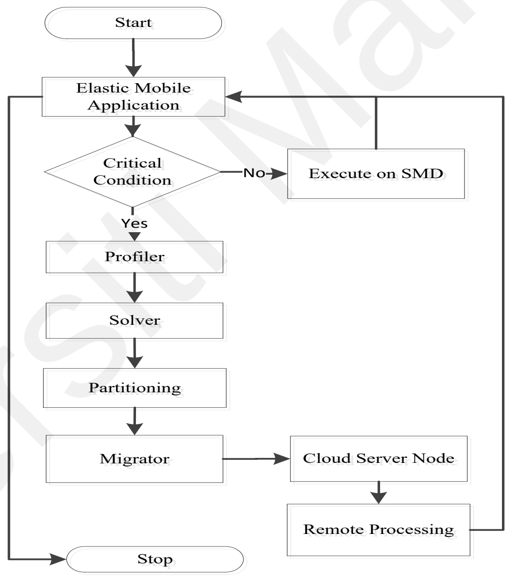

Figure 2. 7: Generic Flowchart for Flowchart Partitioning Migration Based DAPFs (Shiraz et al., 2012)

2.3 Comparison of Distributed Application Processing Frameworks

This section categorized current DAPFs on the basis of local resources utilization model and server resources utilization model. It investigates commonalities and deviations in such frameworks on the basis significant parameters such as offloading scope, partitioning approach, migration support, migration granularity, application developer support, migration pattern and execution monitoring. The following section discusses such parameters in detail.

2.3.1 Offloading Scope (OS)

The offloading scope attribute of DAPFs represent the scope of distributed platform established at runtime. Current DAPFs deploy the following offloading scopes.

a) ***Local Resources Utilization:*** The local resources utilization models utilize computing resources and services of local computing nodes. Current DAPFs implement the following three different types of decentralized computing resources utilization models.

i) Decentralized distributed platform which is composed of stationary remote hosts or mobile nodes. In this model the distributed platform is established by utilizing computing resources of remote servers in close proximity.

ii) Virtual or Ad-Hoc distributed platform which is composed of mobile nodes. In this model SMDs establish distributed platform in pervasive fashion in local environment.

iii) Centralized distributed platform which is composed of centralized server and mobile worker nodes. In such a model management and monitoring is performed my centralized servers, however actual processing of the application is performed on the decentralized mobile worker nodes.

b) ***Server Based Resources Utilization*:** The server based resources utilization models utilize computing resources and services of the centralized servers. Current DAPFs implement the following three different types of centralized computing resources utilization models.

a) Grid server based distributed platform in which remote services are provided by grid servers.

b) Telecommunication Service Provider (TSP) based distributed platform wherein remote services are configured at TSP servers.

c) Cloud servers based distributed platform in which remote services are configured at cloud datacenters.

2.3.2 Partitioning Approach (PA)

The partitioning approach attribute of offloading frameworks represent the partitioning strategy of the framework. Current DAPFs implement application partitioning in two different ways. Static partitioning in which the application is partitioned in fixed number of partitions either at compile time or runtime. Dynamic partitioning approach is used for partitioning of elastic mobile application at runtime. Static partitioning frameworks are represented as '**static**', whereas the dynamic frameworks are represented with '**dynamic**' The notation **'n/a'** is used for non-partitioning DAPFs.

2.3.3 Migration Granularity (MG)

The migration granularity attribute of the traditional DAPFs represent the granularity of migrating component of the application. The possible granularity levels for currents DAPFs are as follows. a) Module level migration represents that entire module or bundle of the application is migrated to remote environment. b) Method level migration represents that partitioning occurs at application method level and intensive methods of the application are migrated to remote server. c) Object level migration represents that entire object is migrated to remote environment for outsourced processing. d) Thread level migration represents thread level partitioning and migration of the application to remote environment. e) Entire application migration in which case entire application is offloaded to remote server.

2.3.4 Migration Support (MS)

The migration support attribute of the offloading model represents the level of support required for the migration of application. Current DAPFs require two different levels of migration support. a) System level support requires additional operating system support for the migrating components of the application such as VM deployment and management. b) Application level support means offloading is performed without additional support from operating system.

2.3.5 Migration Pattern (MP)

The migration pattern attribute represents the pattern of migration of application to remote node. The following MPs are implemented by current DAPFs.

a) Application transfer is a migration pattern in which case the code of the application is outsourced to remote server.
b) URL download represents a migration pattern in which case a URL is provided to remote host and application is downloaded from that URL as a substitute of transferring the application directly from SMD.
c) VM Instance represents a migration pattern in which the application is encapsulated in VM instance (partially or entirely) and VM instance is migrated to remote server. A fresh VM instance is created on the remote server and guest VM instance is copied to the freshly created VM on remote server.
d) UMSC is a migration pattern in which mobile agent is employed for the migration of outsourcing application. UMSC serve as a courier for the migration of the application between SMD and remote server.

e) File download represents a migration pattern in which the mobile application is offloaded by downloading the application file. The communication between SMD and remote server occurs in the form of XML files.

f) Module/Bundle transfer represents a migration pattern in which case modules of the application are migrated to remote servers either by VM migration or code transfer.

g) Application proxy is a migration pattern in which entire proxies of the application are replicated on remote server.

h) Object transfer is a migration pattern in which case entire object is outsourced to remote server at application level.

2.3.6 Developer Support (DS)

A number of current DAPFs required developers support for defining execution scope of the components of application at different granularity level. DS shows the requirement of additional support required for the development of the application.

2.3.7 Execution Management (EM)

The execution Management attribute shows the management policy for the deployment and management of runtime distributed application platform. a) Decentralized management represents the unavailability of the centralized mechanism for the deployment and management of distributed platform. Therefore, SMDs are responsible for monitoring distributed platform and distributed application execution. b) Centralized management represents that a centralized management and monitoring mechanism is provided for the establishment of distributed platform and monitoring of application execution.

2.3.8 Security Support (SS)

SS represents the security provision attribute of the DAPFs. The availability of the security support mechanism in the framework is represented with the value 'yes', whereas unavailability of the security support in the offloading model is represented with the value 'no'. Table 2.1 shows the commonalities and deviations in the local resources sharing based DAPFs on the basis of the aforementioned parameters

Table 2. 1: Comparison of Local Resource Sharing Based Application Offloading Frameworks (Shiraz et al., 2012)

Framework	OS	PA	MG	MS	MP	SS	DS	EM
Distributed Platform for Resources Constrained Devices (Messer et al. 2002)	Local	Dyna mic	Class Level	Application Level	Application Transfer	No	Not Required	Decentralized
Secure Cyber Foraging (Goyal et al., 2004)	Local	n/a	Entire Applicatio n	System Level for VM	URL download	Yes	Required	Decentralized
Clone cloud (Chun and Maniatis, 2009)	Local	Static	Entire Applicatio n/ Partitionin g	System level	VM Instance	No	Required	Decentralized
Optimized Solution for Mobile Devices (Liu et al., 2009)	LocalAd -Hoc Cloud	n/a	Entire Applicatio n	n/a	UMSC	No	n/a	Decentralized
VM-Based Cloudlets (Satyanarayana n et al., 2009)	Local	n/a	Entire Applicatio n	System Level	VM Instance	No	n/a	Decentralized
DISHES (Chung et al., 2010)	Local	n/a	Entire Applicatio n Offloadin g	System Level	Code Download/ Transfer.	No	Not required	Centralized
Virtual Cloud Computing (Canepa et al., 2010)	Local Ad-hoc Cloud	n/a	Entire Applicatio n	n/a	Application Transfer	Yes	No	Decentralized
Misco (Dou. et al., 2010)	Ad-hoc mobile cloud	Static	Method level	Application Level	File download	No	Yes	Centralized

Local application offloading frameworks employ decentralized monitoring approach for process offloading which results in the extensive involvement of SMDs for the management of distributed processing. Further, local offloading frameworks are deficient in the centralized management and the availability of resources for the provision of remote services. In the scenario of unavailability of local remote service provider, remote services become inaccessible which hinders the objectives of availability and scalability of services in distributed computing paradigm. To cope with the issues of decentralized DAPFs, centralized server based solutions are implemented. Table 2.2 compares server based offloading frameworks in which centralized resources are available for the management and provision of remote services.

Table 2. 2: Comparison of Server Based Application Offloading Frameworks (Shiraz et al., 2012)

Framework	**OS**	**AP**	**MG**	**MS**	**MP**	**SS**	**DS**	**EM**
Calling the Cloud (Giurgiu et al., 2009)	Cloud Server	Dynamic	Modules (Bundle)	Application Level	Bundles Transfer	No	Not Required	Centralized
MAUI (et al., 2010)	Cloud Server	Dynamic	Method Level	Application Level	Applicati on Proxy	No	Yes	Centralized
Dynamically Partitioning Applications (Chun et al., 2010)	Cloud Server	Dynamic	Module level	Application level	Applicati on Transfer	No	n/a	Centralized
Energy Savings and Privacy Protection (Liu. et al., 2010)	Grid Server	n/a	Image Object	Application Level	Object Migration	No	Not Required	Centralized
CloneCloud (Chun et al., 2011)	Cloud Server	Dynamic	Thread	System Level	VM Instance	No	Not Required	Centralized
COGNISERVE (Iyer et al., 2011)	Cloud Server	n/a	Entire Job	Application Level	Object Migration	No	Na	Centralized
Elastic Application Model (Zhang et al., 2011)	Cloud Server	Dynamic	Bundles (Weblets)	Application level	URL download	Yes	No	Centralized
Mirroring Smartphones (Zao et al., 2011)	TSP Based Server	n/a	Entire Application	Application Level	VM Instance	Yes	Not Required	Centralized
Virtualized Execution Environment (Hung et al., 2011)	Cloud Server	n/a	Entire Application	Application Level	VM Instance	No	Not Required	Centralized

Server based application offloading frameworks accomplish outsourced application processing in a number of ways. Several approaches exploit VM cloning; others focus on part(s) of the application to be offloaded. A number of approaches implement dynamic partitioning whereas other focus on entire job migration. Traditional server based offloading model implement diverse objective functions; such as saving processing power, efficient bandwidth utilization, saving energy consumption, user preferences, and execution cost. Server based DAPFs provide centralized management and ensure availability of remote services. However, a number of obstacles obstruct optimization goals of server based remote application processing. In the following section, we highlight key challenges to current DAPFs and identify general issues for the distributed processing of mobile applications for MCC.

2.4 Issues and Challenges for Distributed Application Deployment in MCC

The following section discusses issues in current offloading frameworks and identifies challenges to the cloud based application processing of resources intensive mobile applications.

2.4.1 Scalability and Availability of Services and Resources

Scalability of services is a challenging aspect of distributed application processing in mobile cloud computing. The traditional local DAPFs for remote application processing are deficient in centralized management of the distributed platform. A challenging issue in local DAPFs is the unavailability of centralized resources. For example; in the scenario of unavailability of remote service provider, remote services become inaccessible which hinders the objectives of availability of services in distributed computing paradigm.

Similarly, local resources are accessible to limited number of mobile devices in the local environment. Therefore, the possibility of inaccessibility of the remote services always remains associated in local distributed models. Whereas, scalable systems ensure the provision of services irrespective of the number of clients accessing the services. Therefore, unavailability of centralized resources and services and scalability of services is a challenging research issue for ad-hoc and virtual distributed models of MCC. It is challenging to implement peaceful degradation policy on SMDs in the critical conditions of unavailability of remote services. Scalable systems sustain the provision of services and resources for large number of clients whereas availability of services ensures the provision of remote services. It is imperative to ensure the scalability of services in cloud datacenters so that SMDs are enabled to access centralized services for distributed application deployment with high aim of scalable remote services.

The centralized datacenter based computational cloud are resources rich and computational resources and services are provided on demand basis. Cloud resources and services are accessible to both stationary computer clients and SMDs. However, the unique architecture, compact design, operating platforms, low computing potentials, and portable mobile nature of smart mobile devices require special services for ensuring the availability of cloud services. The mobile nature and the intrinsic limitations associated with the wireless access medium of SMD necessitate availability of cloud services and resources homogeneously worldwide. It is challenging in cloud based processing of mobile applications to ensure the availability of services and identical access to cloud services over different types of wireless network technologies (Wi-Fi, 3G and LTE). Therefore, sustaining uninterrupted provision of cloud services and resources to SMDs is a challenging research perspective of mobile cloud computing.

2.4.2 Lightweight Distributed Application Deployment

In current DAPFs, resources intensive distributed platform is established at runtime. Mobile applications offloading frameworks are developed on the basis of standalone application architecture, whereas the processing of application is performed in the distributed fashion. Therefore, current DAPFs establish a resources intensive and complex computing environment at runtime. Application offloading techniques are primarily based on either entire application/job migration or application partition migration to remote servers. The implementation of distributed architecture for virtual mobile cloud is hindered by the following obstructs.

1. Local distributed processing models lack in the availability of centralized management; for that reason it is difficult to configure explicitly defined client and server components for the mobile applications.
2. Virtual clouds necessitate special requirements for the establishment of distributed platform which is challenging to maintain for mobile devices which are participating in ad-hoc cloud. The special requirements include; SMDs remain in the close proximity, follow the same movement patterns, voluntariness for service and provision, implementation of specific service architecture (Canepa et al., 2010).

The additional computing resources of SMDs are utilized for the configuration of distributed platform and management of distributed services provision to the requesting client devices. Further, shorter battery life time of SMDs is major challenge in virtual/ad-hoc distributed application processing models. Therefore, the ad-hoc and virtualized nature of local distributed platform is another obstacle in explicitly defining client and server components of the mobile application. However, the availability of centralized resources and services and centralized management mechanism in cloud datacenters are the motivating factors for incorporating distributed architecture for the intensive mobile applications. It is challenging for distributed mobile application to incorporate the principles of distributed applications in such a manner so that mobile applications can operate in the situations of inaccessibility of cloud server nodes.

2.4.3 Seamless Connectivity and Consistent Distributed Platform

Mobility is an important attribute of SMDs. Mobile users enjoy the freedom of computing and communication on move. However, a number of obstacles hinder the goals of seamless connectivity and consistency in the distributed platform of mobile applications; for example handoffs, traveling with high speed, diverse geographical locations and different environmental conditions. As a result, providing seamless connectivity and uninterrupted access to the centralized cloud datacenters in distributed application processing is a serious research issue for MCC.

It is important that distributed application model provide versatile access to cloud resources and services on move with ubiquitous attributes and high degree of transparency. However, it is challenging to ensure the transparency of distributed environment. In particular to SMD, the issues and limitations in wireless medium hinder the transparency goals of distributed processing of mobile application. The seamless and transparent deployment of distributed platform for computational intensive applications is a challenging aspect for mobile cloud computing. It is mandatory for distributed model to mask the complexities of distributed environment from mobile user and give the notion as the entire application is being processed locally on SMD. Similarly, it is important to ensure successful execution of remote processing and returning results to SMD. Sustaining consistency of the offloaded components of the application with lightweight implementation procedures is a challenging aspect of DAPFs. Consistency is an issue for the components offloaded at runtime (Zhang et al., 2011), the replicated applications using proxies (Cuervo et al., 2010), and transactions involving related updates to different objects.

It is important that the distribution and replication of intensive mobile applications and data should be transparent to the mobile users and application running client device. Cloud based distributed processing of mobile application are required to fulfill Atomic, Concurrency, Isolation and Durability (ACID) properties of the distributed systems. It is challenging to provide location transparency, replica transparency, concurrency transparency, and failure transparency in cloud based application processing of mobile applications.

2.4.4 Homogenous and Optimal Distributed Platform

Homogenous and optimal cloud based application processing is an important research perspective in mobile cloud computing. Heterogeneity of SMD architecture and operating platform is challenging for distributed application processing in MCC. Mobile device vendors employ different hardware architecture and operating system platforms for the specific mobile product. Traditional application offloading frameworks focus on the implementation of platform dependent procedures for outsourcing computational intensive loads. For example, Weblets (Zhang et al., 2011) and MAUI (Cuervo et al., 2010) are application offloading frameworks which are applicable for .Net framework, whereas virtualized execution framework (Hung et al., 2011) and mirror server (Zao et al., 2011) are suitable frameworks for android platform. Hence, homogenous access to cloud services are highly expected wherein SMD are enabled to access widespread computing services of computational clouds irrespective of the concerns about operating hardware architecture and operating system platform. A homogenous distributed application deployment solution for the heterogeneous available SMDs platforms is a challenging issue for MCC.

Sanae et al., (2012) proposed a tripod of requirements with three legs of trust, energy efficiency, and ubiquity. It describes important metrics such as heterogeneity, under this tripod which are crucial for the success of cloud-mobile applications. Similarly, the deployment of distributed application processing platform at runtime is a resources intensive mechanism. It uses computing resources on SMDs for the evaluation of computing resources utilization on SMDs and partitioning of intensive mobile applications at runtime. Current, DAPFs necessitate continuous assessment of application execution requirements on SMD which is a resource intensive operation.

DAPFs employ runtime profiling and solving mechanism on SMDs periodically or casually to evaluate application processing requirements and the availability of computing resources on SMD. The centralized distributed application deployment models require arbitration of SMD with centralized server for the selection of appropriate server node. As a result, computing resources (CPU, battery power) of SMD are exploited abundantly for the entire process of application profiling and solving. The deployment of distributed platform, management and operation of remote application processing in the optimal possible fashion is an important perspective of cloud based application processing. It is challenging to provide homogenous solution for heterogeneous devices, operating platforms and network technologies with minimum possible resources utilization on the SMDs.

2.4.5 Security and Privacy in Cloud Based Application Processing

Privacy in the distributed platform and security of data transmission between mobile device and cloud server node are important concerns in cloud based application processing. Privacy measures are required to ensure the execution of mobile application in isolated and trustworthy environment, whereas security procedures are required to protect against network threats. Security and privacy are very important aspects for the establishing and maintaining the trust of mobile users in cloud based application processing (Subashini et al., 2010).

Security in MCC is important from three different perspectives: security for mobile devices, security for data transmission over the wireless medium and security in the cloud datacenter nodes. SMDs are subjected to a number of security threats such as viruses and worms. SMDs are the attractive targets for attacker. According to a report (Protecting Portable Devices, n.d.) the number of new susceptibilities in mobile operating systems increased 42 percent from 2009 to 2010. The number and complications of attacks on mobile phones is increasing speedily as compared to the countermeasures. Data transmission over the wireless networks is highly vulnerable to network security threats. For example, using radio frequencies, the risk of interruption is higher than with wired networks therefore attacker can easily compromise confidentiality (Choi et al., 2008). Similarly, in cloud datacenters the security threats are associated with the transmission between physical elements on the network, and traffic between the virtual elements in the network, such as between virtual machines within a single physical server. Therefore, in order to leverage the application processing services of computational clouds, a highly secure environment is expected at all the three entities of MCC model.

In current DAPFs, transmission of the running states of mobile application which is encapsulated in VM (Chun et al., 2009; Satyanarayanan et al., 2009; Giurgiu et al., 2009;) or binary transfer of the application code at runtime (Giurgiu et al., 2009 ;Chung et al., 2010;) is continuously subjected to security threats at mobile device, wireless medium and cloud datacenters. Therefore, secure transmission of the entire components of the application is a challenging issue for MCC.

It is important to implement reliable security measures for the data transmission, and synchronization between SMD and cloud datacenters in distributed processing platform. Similarly, access control, fidelity and privacy of distributed application components in the remote cloud datacenters is an important consideration for the distributed application processing in MCC. Cloud datacenters provide augmentation services which are unapproachable to mobile users. Therefore, it is highly demanding to ensure the privacy of data and computing operations in remote server nodes. A trustworthy distributed application model is highly expected to cope with such important issues and ensure the trustworthiness of remote computing environment. A reliable distributed environment is expected to provide authentic access to authorized mobile user for legitimate operations on cloud server nodes.

Considering the aforementioned research issues and challenges for distributed application deployment in MCC, lightweight and optimal distributed application deployment solution is extremely important. Such a solution should incorporate optimal procedures for the development, deployment and management of runtime distributed platform for MCC.

2.5 Conclusion

This chapter discusses the concept of cloud computing, mobile cloud computing and explains different techniques to augment the computing capabilities of SMDs based on resources available within the cloud. It analyzes current DAPFs by using thematic taxonomy and highlights the commonalties and deviations in such frameworks on the basis of significant parameters. It discusses issues in current DAPFs and highlights challenges to optimal and lightweight distributed application framework for MCC.

Current DAPFs accomplish process offloading in diverse modes. Several approaches employ entire application migration; others focus on part(s) of the application to be offloaded. A number of approaches employ static partitioning, others implement dynamic partitioning. Variant migration patterns are used; downloading application by providing URL to remote host, VM cloning, Mobile agent such as USMC, application binary transfer and use of proxies. Diverse objective functions are considered; saving processing power, efficient bandwidth utilization, saving energy consumption, user preferences, and execution cost. Objective of all approaches is to alleviate computing resources limitations of mobile devices in the processing of intensive mobile applications.

Current DAPFs for MCC are the analogous extensions of traditional cyber foraging frameworks for pervasive computing or local distributed platforms. Hence, current DAPFs are deficient of the deployment of distributed system standard architectures. As a result, additional complications arise in the development, deployment and management of distributed platform. Current frameworks focus on the establishment of runtime distributed platform which results in the resources intensive distributed management overhead on SMDs for the entire duration of distributed platform. The additional computing resources of SMDs are utilized in arbitration with cloud servers for the selection of remote node, dynamic the availability of resources on SMDs and resources requirement of mobile application, dynamic application profiling, synthesizing and solving for application outsourcing, application migration and reintegration and continuous synchronization with cloud servers for the duration of distributed platform. As a result, additional computing resources of the SMDs are utilized for the runtime configuration of distributed platform.

Hence, current computational offloading frameworks employ heavyweight procedures for distributed application deployment and management. The mobile nature, compact design, limited computing potential and wireless medium attributes of SMDs necessitate for lightweight procedures for the distributed deployment and processing of computational intensive applications in MCC.

CHAPTER 3

Problem Analysis

This chapter investigates additional computing resources utilization in the traditional computational offloading frameworks for MCC. The chapter is organized into four sections. Section 3.2 investigates the additional cost of energy consumption, timing cost and size of data transmission in traditional computational offloading for mobile cloud computing. Section 3.3 analyzes the impact of Virtual Machine (VM) deployment for application processing in MCC. Section 3.4 summarizes the chapter with conclusive remarks.

3.1 Introduction

The problem of additional resources utilization in traditional runtime computational offloading is analyzed by benchmarking the prototype application in the real mobile cloud computing environment. Traditional computational offloading is implemented by offloading the resource intensive service components of the prototype mobile application. Computational offloading to remote cloud server node is evaluated for varying computational intensities of the application. The measurement parameters for problem analysis include energy consumption cost, time taken in runtime component offloading (timing cost), and size of data transmission over the wireless network medium. The impact of virtual machine deployment for application processing is analyzed by using CloudSim simulation toolkit.

3.2 Analysis of Traditional Computational Offloading for MCC

The traditional Distributed Application Processing Frameworks (DAPFs) for MCC establish distributed platform at runtime, wherein additional computing resources are utilized on SMD. Traditional computational offloading frameworks employ runtime migration of the intensive components of the mobile application, wherein the intensive components of the mobile application are offloaded dynamically at runtime (Hung et al., 2012). A number of application offloading frameworks implement dynamic application profiling and partitioning technique for application offloading (Messer et al., 2002; Giurgiu et al., 2009; Cuervo et al., 2010; Chun et al., 2011; Zhang et al. 2011). The traditional computational offloading frameworks focus on what components of the application to offload, how to offload and where to offload the application partitions. However, such frameworks lack of considering the additional cost of runtime distributed application deployment for MCC.

This section analyzes the traditional computational offloading by outsourcing the resource intensive components of the mobile application with varying computational intensities to remote cloud server node. A prototype application is developed for Android devices, which is composed of two computational intensive components; sorting service and matrix multiplication service. The sorting service component implements the logic of bubble sorting for sorting liner list of integer type values. The runtime computational offloading for sorting service of the application is evaluated with 30 different computational intensities (11000-40000). The matrix multiplication service of the application implements the logic of computing the product of 2-D array of integer type values. Runtime computational offloading for matrix multiplication service component of the application is evaluated with 30 different computational intensities by varying the size of the 2-D array between 160*160 and 450*450. The total Energy consumption Cost (*Ec*), Timing Cost (*Tc*) and size of data transmission are evaluated in different experiments by offloading the service components of the mobile application at runtime.

3.2.1 Analysis of the Energy Consumption Cost

The additional energy consumed in runtime computation offloading is evaluated by Energy consumption cost (*Ec*) parameter in the units of Joules (J). *Ec* includes energy consumed in runtime component migration, energy consumed in saving the data states of running instance of the mobile application, energy consumed in uploading the data file to remote server node and energy consumed in returning the resultant data files to local mobile device. Hence, the total energy consumption cost for each component offloaded at runtime is given by the following equation.

$$Ec = E_m + E_s + E_u + E_d \qquad (3.1)$$

a) Energy consumed in component Migration (E_m) represents energy consumed in transferring the binary code of the component of mobile application which is being offloaded.

b) Energy consumed in Saving preferences (E_s) represents energy consumed in saving the running instances of the mobile application.

c) Energy consumed in Uploading preferences (E_u) represents energy consumed in uploading the data file (which is known as preferences file) to remote server node at runtime.

d) Energy consumed in Downloading preferences (E_d) represents energy consumed in downloading the resultant data file (preferences file) to the local mobile device.

Let E is the finite set of the energy consumption cost of the components of mobile application which are offloaded at runtime. Hence, set E is given as:

E= {the finite set of the energy consumption cost of the components of mobile application which are offloaded at runtime}

Let Ec_a represents the energy consumption cost of offloading a single component of the mobile application at runtime. Where $a=1, 2,\ldots, n$

$$\therefore\ E=\{Ec_1, Ec_2,\ldots, Ec_n\}$$

Ec_a represents the energy consumption in offloading a single component of the mobile application which is a positive Real number. Therefore, by using set builder notation the Ec_a is represented as:

$$E=\{\, Ec_a : Ec_a \in \mathbb{R} \wedge Ec_a > 0 \,\} \qquad \text{Whereas, } a=1, 2,\ldots,n$$

The energy consumption cost of offloading a single component of the mobile application belongs to the set of Real numbers and is greater than 0. The total energy consumption cost in runtime component offloading is the sum of energy consumption cost of all the instances a=1, 2,…, n of runtime component offloading. Let the total energy consumption of the runtime application offloading is represented by α_e, which is the sum of energy consumed in all instances $Ec_{a=1,2,\ldots,n}$ of the runtime component offloading. Therefore, α_c is represented as follows.

$$\therefore\ \alpha_e=(Ec_1+Ec_2+\ldots+Ec_n) \qquad \Rightarrow \forall\ Ec_a \in \mathrm{E} \wedge |\mathrm{E}| \geq 1 \quad \text{whereas } a=1,\ldots, n$$

By using summation notation the total energy consumption cost (α_e) of the runtime computational offloading of the mobile application is represented as follows:

$$\alpha_e=\sum_{a=1}^{n} Ec_a \qquad \Rightarrow \forall\ Ec_a \in \mathrm{E} \wedge |\mathrm{E}| \geq 1 \qquad (3.2)$$

For all Ec_a, which denotes the energy consumption cost of the single instance of runtime component offloading of the mobile application belongs to the set E and the cardinality of set E is greater than or equal to 1. E is the set of the energy consumption cost of the components of the mobile application which are offloaded at runtime. The precondition validates that E is none empty set.

The energy consumption cost (Ec_1) for offloading the sorting service component of the application at runtime is evaluated for 30 different computational intensities of sorting operation (11000-40000). The energy consumption cost of transferring application binary code (E_m), is evaluated in 30 experiments by offloading sorting service with 30 different computational intensities. It is examined that in all instances of offloading the binary code of the application, the size of binary application file (.apk) remains constant (44.4 KB). Hence, E_m remains constant in offloading sorting service of the application with different intensities. It is examined that the sample mean of E_m is 6.1(+/) 0.6 J with 99% confidence for the sample space of 30 values which shows that the possible range for E_m is between 5.5 and 6.7 J.

The energy consumption cost of saving the data states (preferences file) on the mobile device is examined for 30 different computational intensities of the sorting service component of the application. It is examined that the sample mean of E_s is 8.5(+/-)1 J with 99% confidence for the sample space of 30 values which shows that the possible range for E_s is between 7.5 J and 9.5 J. The energy consumption cost of uploading preferences file (E_u) to the cloud server node is examined for uploading the preferences file of 30 different computational intensities of the sorting service. It is examined that the sample mean of E_u is 36(+/-) 0.92 mJ with 99% confidence for the sample space of 30 values which shows that the possible range for E_u is between 35 mJ and 36 mJ. The energy consumption cost of downloading the resultant preferences file (E_d) from the remote server node to the local mobile device is evaluated in 30 different experiments. It is examined that the confidence interval for the average E_d is 10.9 (+/-) 0.28 J in downloading the resultant preferences file of sorting service for the sorting list length 11000-40000 values.

The total Energy Consumption Cost (Ec_1) in runtime computational offloading of sorting service is computed by using equation (3.1). Table 3.1 shows the total Ec_1 in offloading sorting service by using traditional computational offloading technique. The attribute of sorting length shows the length of sorting operation, the Energy consumption cost attribute indicates the point estimator for the sample space of 30 values in each experiment and the standard deviation (SD) shows the variation in the values of the sample space. The confidence interval attribute shows the possible range of the sample mean with 99% confidence for the sample space of 30 values in each experiment.

Table 3. 1: Total Energy Consumption Cost (Ec_1) in Traditional Offloading of Sorting Service

Length of Sorting List	**Energy Consumption Cost (J)**	**SD**	**Confidence Interval**
11000	29.6749	5.7245	29.6749(+/-)2.6965
12000	30.8829	4.4753	30.8829(+/-)2.1080
13000	31.0835	4.2346	31.0835(+/-)1.9946
14000	31.4911	3.7763	31.4911(+/-)1.7787
15000	31.8917	3.8377	31.8917(+/-)1.8077
16000	31.8919	4.1711	31.8919(+/-)1.9647
17000	32.4915	3.9094	32.4915(+/-)1.8414
18000	32.8923	3.835	32.8923(+/-)1.8064
19000	33.092	4.1244	33.092(+/-)1.9428
20000	33.2971	4.0274	33.2971(+/-)1.89707
21000	33.2989	4.6845	33.2989(+/-)2.2066
22000	33.6987	4.4209	33.6987(+/-)2.08242
23000	34.3002	4.3997	34.3002(+/-)2.0724
24000	34.7008	4.4183	34.7008(+/-)2.0812
25000	34.9014	4.214	34.9014(+/-)1.9849
26000	35.502	4.0472	35.502(+/-)1.90639
27000	35.7036	4.7645	35.7036(+/-)2.2442
28000	36.1034	4.8614	36.1034(+/-)2.2899
29000	36.705	4.5308	36.705(+/-)2.1342

Length of Sorting List	Energy Consumption Cost (J)	SD	Confidence Interval
30000	37.7055	4.1596	37.7055(+/-)1.9594
31000	38.507	4.616	38.507(+/-)2.17432
32000	39.7078	4.9652	39.7078(+/-)2.3388
33000	41.5086	4.7245	41.5086(+/-)2.2254
34000	42.5095	4.3692	42.5095(+/-)2.05807
35000	42.7109	4.7827	42.7109(+/-)2.2528
36000	43.1131	5.0774	43.1131(+/-)2.3916
37000	44.1151	4.9271	44.1151(+/-)2.3208
38000	44.3154	5.0068	44.3154(+/-)2.3584
39000	44.5168	5.3538	44.5168(+/-)2.5218
40000	45.3191	5.5839	45.3191(+/-)2.6303

Figure 3.1 shows the increase in the Ec_1 for offloading sorting service at runtime. It is examined that E_m, and E_s remains constant in offloading sorting service with varying sort list size. However, the size of preferences files increases by increasing the length of sort list. Therefore, the cost of E_u and E_d increases accordingly. It is examined that the average cost of E_u is 9.8 mj for uploading preferences file of sorting list length 11000 values, whereas the average cost of E_u is 54 mJ for uploading preferences file of sorting list length 40000 values. Hence, the E_u increases 81.9 percent for uploading preferences file of sort list length 40000 values as compared to uploading preferences file of sort list length 11000 values. Similarly, it is examined that the cost E_d increases according the size of preferences file. For instance, 7.5 J energy is consumed in downloading preferences file for sorting list size 11000 values; whereas 15.3 J energy is consumed in downloading preferences file for sorting list size 4000. It shows that the cost of E_d increases 51 percent for in downloading the preferences file for the sorting list of 40000 as compared to the preferences file for the sorting list size 11000 values. It shows that increase in the Ec_1 is the result of increase in uploading and downloading larger preferences files. The average total

energy consumption cost of offloading sorting service (Ec_1) at runtime is determined as 36.58 J.

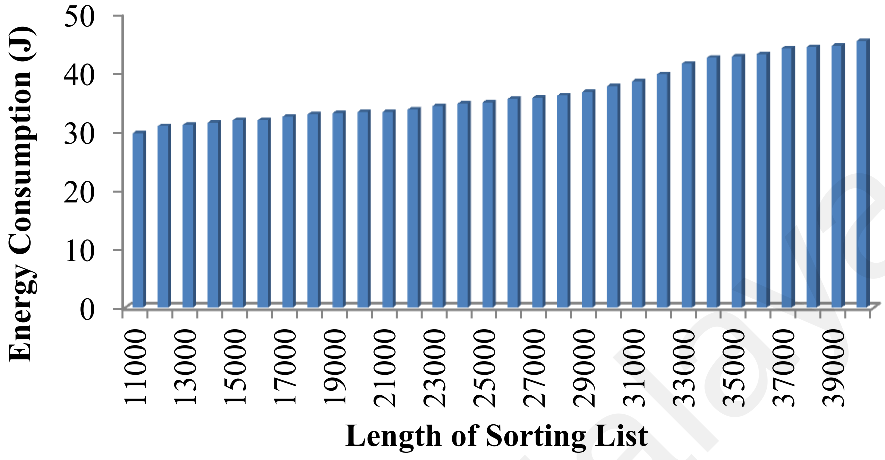

Figure 3. 1: Total Energy Consumption Cost (Ec_1) in Traditional Offloading of Sorting Service

The total Energy Consumption Cost (ECC) in offloaded execution of the sorting service varies for different intensities of the sorting operation. It is examined that total ECC in the remote execution of sorting operation increases with the increase in the length of sorting list. For instance, the total ECC of sorting list 11000 values is 49.8 J, whereas the total ECC of sorting list 40000 values is 201.4 J. It shows that the ECC increases 75.3 percent for sorting the list of 40000 values as compared to sorting the list of 11000 values. The average total ECC cost in offloaded execution of sorting service for sorting the list of 11000-40000 values is 111.2 J. with 42.2 percent RSD. The total energy consumption cost (Ec_2) for offloading the matrix multiplication service component of the application at runtime is evaluated for 30 different computational intensities of matrix multiplication operation (160*160-450*450). The energy consumption cost of transferring application binary code (E_m) for matrix multiplication service, is evaluated in 30 experiments by offloading matrix multiplication service with 30 different computational intensities. It is examined that in all instances of offloading the binary code of the application the size of binary application file (.apk) remains constant (46 KB). Hence, E_m remains constant in offloading matrix multiplication service of the application with different intensities. It is examined that the sample mean of E_m is 15.2(+/-)2.1 J with 99% confidence for the sample space of 30 values which shows that the possible range for *Em* is between 13.6 J and 17.3 J.

The energy consumption cost of saving the data states (preferences file) on the local mobile device is examined for 30 different computational intensities of the matrix multiplication component of the application. It is examined that the sample mean of E_s is 4.6(+/-)0.9 with 99% confidence for the sample space of 30 values which shows that the possible range for E_s is between 3.66 J and 5.46 J. The energy consumption cost of uploading preferences file (E_u) to the cloud server node is examined for uploading the preferences file of 30 different computational intensities of the matrix multiplication service. It is examined that the sample mean of E_u is 273.9(+/-)1.4 mJ with 99% confidence for the sample space of 30 values which shows the possible range for E_u is between 259.6 mJ and 288.1 mJ. The energy consumption cost of downloading the resultant preferences file (E_d) from the remote server node to the local mobile device is evaluated in 30 different experiments. It is examined that the confidence interval for the average E_d is 9.3(+/-) 1.4 J in downloading the resultant preferences file of matrix multiplication service for the matrix length 160*160-450*450 values.

The energy consumption cost (Ec_2) in runtime computational offloading of matrix multiplication service is computed by using equation (3.1). Table 3.2 shows the total Ec_2 in offloading matrix multiplication service by using traditional computational offloading technique. The attribute of length of matrix shows the size of 2-D arrays for matrix multiplication operation, the energy consumption cost attribute shows the point estimator for the sample space of 30 values in each experiment and the standard deviation (SD) attribute shows the variation in the values of the sample space. The confidence interval attribute shows the possible range of the sample mean with 99% confidence for the sample space of 30 values in each experiment.

Table 3. 2: Energy Consumption Cost (*Ec_2*) for Offloading Matrix Multiplication Service in Traditional Computational Offloading

Matrix Length	Energy Consumption Cost (J)	SD	Confidence Interval
160*160	28.7898	9.3609	28.7898(+/-)4.4093
170*170	28.9927	10.3147	28.9927(+/-)4.8586
180*180	31.198	8.7939	31.198(+/-)4.1422
190*190	31.5995	8.9789	31.5995(+/-)4.2294
200*200	30.8027	9.6921	30.8027(+/-)4.5654
210*210	31.8297	10.1287	31.8297(+/-)4.771
220*220	31.8349	9.2277	31.8349(+/-)4.3466
230*230	33.4504	9.3581	33.4504(+/-)4.4018
240*240	33.4534	9.1811	33.4534(+/-)4.3247
250*250	36.2603	7.3977	36.2603(+/-)3.4846
260*260	36.8631	7.5401	36.8631(+/-)3.5517
270*270	38.9291	8.2521	38.9291(+/-)3.8871
280*280	38.5867	7.8341	38.5867(+/-)3.6902
290*290	38.9974	7.7375	38.9974(+/-)3.6447
300*300	38.0067	7.6129	38.0067(+/-)3.5860
310*310	39.0187	8.1817	39.0187(+/-)3.8539
320*320	40.0983	7.5075	40.0983(+/-)3.5363
330*330	39.5353	8.5737	39.5353(+/-)4.0386
340*340	40.8619	8.7749	40.8619(+/-)4.1333
350*350	42.3617	7.8729	42.3617(+/-)3.7085
360*360	41.8239	7.7127	41.8239(+/-)3.633
370*370	42.0589	8.0985	42.0589(+/-)3.8147
380*380	42.3943	7.5279	42.3943(+/-)3.546
390*390	42.6383	7.5029	42.6383(+/-)3.5342
400*400	42.2728	8.0943	42.2728(+/-)3.8128
410*410	42.6444	7.7541	42.6444(+/-)3.6525
420*420	44.3066	8.0657	44.3066(+/-)3.7993

Matrix Length	Energy Consumption Cost (J)	SD	Confidence Interval
430*430	45.6333	7.8499	45.6333(+/-)3.6976
440*440	49.7541	9.2795	49.7541(+/-)4.371
450*450	52.6952	7.9959	52.6952(+/-)3.7664

Figure 3.2 shows the increase in the total Ec_2 for offloading matrix multiplication service at runtime. It is examined that the cost of E_m, and E_s remains constant in offloading matrix multiplication service with varying matrix length values. However, the size of preferences file increases by increasing the length of matrices. Therefore, the cost of E_u and E_d increases accordingly. It is examined that the average cost of E_u is 3.7 mJ for uploading preferences file of matrices length 160*160, whereas the average cost of E_u is 136.6 mJ, for uploading preferences file of matrices 450*450 length. Hence, the E_u increases 72.4 percent for uploading preferences file of matrices length 450*450 as compared to uploading preferences file of matrices length 160*160 values.

Similarly, it is examined that the cost E_d increases according to the size of preferences file. For instance, 4.5 J energy is consumed in downloading preferences file for matrices of length 160*160 values; whereas 16.2 J energy is consumed in downloading preferences file for matrices of length 160*160 values. It shows that the cost of E_d increases 72.2 percent in downloading the preferences file for matrices length 450*450 values as compared to the preferences file for the matrices length 160*160 values. It shows that increase in the Ec_2 is the result of increase in uploading and downloading larger preferences files. The average energy consumption cost of offloading matrix multiplication service (Ec_2) is determined 38.58 J.

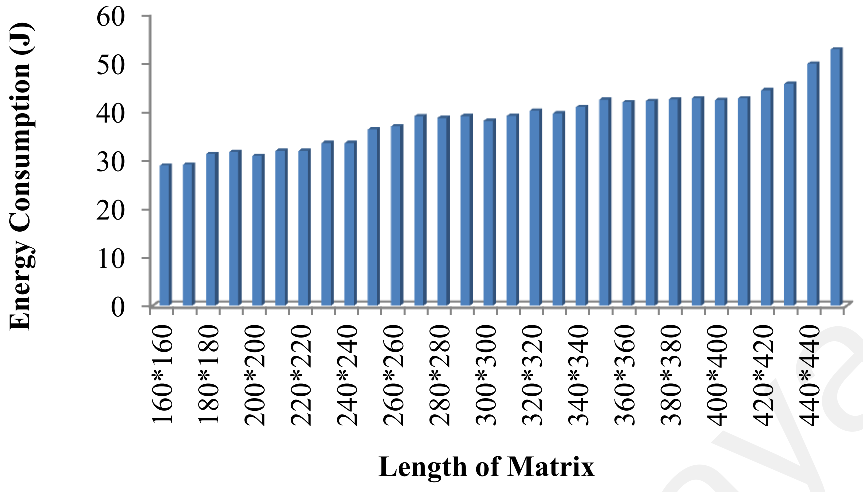

Figure 3. 2: Energy Consumption Cost (*Ec_2*) for Offloading Matrix Multiplication Service in Traditional Computational Offloading

The total Energy Consumption Cost (ECC) in offloaded execution of the matrix multiplication service varies for different intensities of the matrix multiplication operation. It is examined that the total ECC in the remote execution of matrix multiplication operation increases with the increase in the length of matrices. For instance, the total ECC of multiplying matrices 160*160 length is 40 J, whereas the total ECC of multiplying matrices 450*450 length is 131.7 J. It shows that the ECC increases 69.6 percent in multiplying matrices 160*160 length as compared to multiplying matrices 450*450 length. The average total ECC cost in offloaded execution of matrix multiplication service for multiplying matrices (160*160-450*450 length) is 79 J. with 37.4 percent RSD.

Analysis of the results indicates that runtime computational offloading increases the ECC of distributed application execution considerably. For instance, the additional *Ec_1* in traditional offloading of sorting service is 59.6 percent for sort list length 11000, 44 percent for sort list length 20000, 30.1 percent for sort list length 30000 and 22.5 percent for sort list length 40000. The average increase in the *Ec_1* of runtime computational offloading for sorting service is 37.1 percent for sort list length 11000-40000. The additional *Ec_2* in traditional offloading of matrix multiplication service is 72 percent for multiplying matrices 160*160 length, 62.3 percent for multiplying matrices 260*260 length, 47.6 percent for multiplying matrices 340*340 length and 40 percent for multiplying matrices 450*450 length. The average increase in the Ec_2 of runtime computational offloading for matrix multiplication service constitutes 53.2 percent for multiplying matrices 160*160-450*450 length.

We know that Ec_1 is 36.58 J (average energy consumption cost of offloading sorting service) and Ec_2 is 38.58 J (average energy consumption cost of offloading matrix multiplication service), hence by using equation (3.2) the total energy consumed cost (α_c) of runtime computational offloading for the mobile application is calculated as 75.15 J, which means that 39.4 percent additional energy is consumed in offloading the components of the mobile application at runtime.

3.2.2 Analysis of the Timing Cost

The additional time taken in runtime computation offloading is evaluated by using timing cost (Tc) parameter in the units of milliseconds (ms). *Tc* involves preferences saving time, binary code offloading time of the application, time taken in uploading the data states of the mobile application to remote server node, application download time to remote virtual device instance on the cloud server node, application reconfiguration and resuming time on the remote server node and time taken in returning the resultant data file to local mobile device. Therefore, the total offloading time of a single component of the mobile application which is being offloaded at runtime is given by the following equation.

$$T_{c=} T_{cm} + T_{ps} + T_{pu} + T_{dv} + T_{rr} + \mathrm{T}_{pr} \qquad (3.3)$$

a. Code Migration time (T_{cm}) represents time taken in transferring the binary code of the component of the mobile application to the remote server node.

b. Preferences Saving Time *(T_{ps})* represents time taken in saving the data states (preferences file) of the running instance of the component of the mobile application which is being offloaded.

c. Preferences Upload Time *(T_{pu})* represents time required for uploading the data state (preferences file) of the mobile application to remote server node.

d. Download Time to remote Virtual Device (T_{dv}) represents time taken in downloading the offloaded application to remote virtual device instance.

e. Reconfiguration and Resume Time *(T_{rr})* represents time required for the reconfiguration of the offloaded component of the mobile application and resuming the running state of the mobile application on the remote server node.

f. Preferences return Time (T_{pr}) represents time taken in returning the resultant preferences file from remote server node to the local mobile device.

Let T is the finite set of the offloading time of the components of mobile application which are offloaded at runtime. Hence, set T is given as:

T= {the finite set of the offloading time of the of the components of mobile application which are offloaded at runtime}

Let Tc_a represents the timing cost in offloading a single component of the mobile application at runtime. Whereas, $a=1, 2, \ldots, n$

$$\therefore \; T = \{ Tc_1, Tc_2, \ldots, Tc_n \}$$

Tc_a represents the total time taken in offloading a single component of the mobile application. Therefore, by using set builder notation the Tc_a is represented as:

$$T = \{ Tc_a : Tc_a \in \mathbb{N} \wedge Tc_a > 0 \} \qquad \text{Whereas, } a=1, 2, \ldots, n$$

The timing cost of offloading a single component of the mobile application belongs to the set of Natural numbers and is greater than 0. The total additional time taken in runtime computational offloading is the sum of the timing cost of all the components of the application which are offloaded at runtime. Let the total additional time taken in runtime application offloading is represented by α_t, which is the sum of the timing cost of all the instances Tc_a =1,2,…,n of the runtime component offloading. Therefore, α_t is represented as follows.

$$\therefore \quad \alpha_t = (Tc_1 + Tc_2 + \ldots + Tc_n) \qquad \Rightarrow \forall \ Tc_a \in \mathrm{T} \wedge |\mathrm{T}| \geq 1 \quad \text{where } a=1,\ldots,n$$

By using summation notation the total additional time taken in runtime computational offloading of the mobile application is represented as follows:

$$\alpha_t = \sum_{a=1}^{n} Tc_a \qquad \Rightarrow \forall \ Tc_a \in \mathrm{T} \wedge |\mathrm{T}| \geq 1 \qquad (3.4)$$

For all Tc_a, which denotes the timing cost of offloading a single component of the mobile application belongs to the set T and the cardinality of set T is greater than or equal to 1, whereas T is the finite set of the offloading time of the components of mobile application which are offloaded at runtime. The precondition validates that T is none empty set.

The timing cost (Tc_1) for offloading the sorting service component of the application at runtime is evaluated for 30 different computational intensities of sorting operation (11000-40000). The time taken in transferring application binary code (T_{cm}), is evaluated in 30 experiments by offloading sorting service. It is examined that in all instances of offloading the binary code of the application the size of binary application file (.apk) remains constant (44.4 KB). Hence, T_{cm} remains constant in offloading sorting service of the application with different computational intensities. It is examined that the sample mean of T_{cm} is 77(+/-)16 ms with 99% confidence for the sample space of 30 values which shows that the possible range for T_{cm} is between 61 ms and 93 ms.

The timing cost of saving the data states (preferences file) on the local mobile device is examined for 30 different computational intensities of the sorting service component of the application. It is examined that the sample mean of T_{ps} is 5076(+/-) 568 ms with 99% confidence for the sample space of 30 values which shows that the possible range for T_{ps} is between 45081 ms and 5645 ms. The timing cost of uploading preferences file (T_{pu}) to the cloud server node is examined for uploading the preferences file of 30 different computational intensities of the sorting service. It is examined that the sample mean of T_{pu} is 608(+/-) 94 ms with 99% confidence for the sample space of 30 values, which shows that the possible range for T_{pu} is between 514 ms and 704 ms.

The timing cost of downloading the application file to remote virtual device instance (T_{dv}) is evaluated in 30 different experiments. It is examined that the confidence interval for the average T_{dv} is 241(+/-)113 ms for the sample space of 30 values in each experiment, which shows the possible range of value for T_{dv} is between 128 ms and 354 ms. The timing cost of application reconfiguration on remote server node and resuming time (T_{rr}) is evaluated in 30 different experiments. It is examined that the confidence interval for the average T_{rr} is 6662(+/-)884 ms for the sample space of 30 values in each experiment, which shows the possible range of value for T_{rr} is between 5778 ms and 7547 ms.

The timing cost of downloading the resultant preferences file (T_{pd}) from the remote server node to the local mobile device is evaluated in 30 different experiments. It is examined that the confidence interval for the average T_{pd} is 11113(+/-)1813 ms in downloading the resultant preferences file of sorting service for the sorting list length 11000-40000 values. The total timing cost (Tc_1) in runtime computational offloading of sorting service is computed by using equation (3.3). Table 3.3 shows the Tc_1 in offloading sort service by using traditional computational offloading technique.

Table 3. 3: Timing Cost (Tc_1) in Traditional Computational Offloading for Sorting Service

Length of Sorting List	Timing Cost (ms)	SD in Timing Cost	Confidence Interval
11000	10931	855	10931(+/-)403
12000	12050	1320	12050(+/-)622
13000	13186	1025	13186(+/-)483
14000	14197	1025	14197(+/-)483
15000	14989	1082	14989(+/-)510
16000	15146	1018	15146(+/-)480
17000	16135	1044	16135(+/-)492
18000	17971	1732	17971(+/-)816
19000	18704	1464	18704(+/-)690
20000	19688	1432	19688(+/-)675
21000	20004	1637	20004(+/-)771
22000	22077	1317	22077(+/-)620
23000	23375	947	23375(+/-)446
24000	23777	1336	23777(+/-)629
25000	24807	1749	24807(+/-)824
26000	25189	1408	25189(+/-)663
27000	25671	1380	25671(+/-)650
28000	26520	2015	26520(+/-)949
29000	27000	1683	27000(+/-)793
30000	27955	1402	27955(+/-)660
31000	28612	2461	28612(+/-)1159
32000	29284	2183	29284(+/-)1028
33000	30045	1507	30045(+/-)710
34000	30402	1111	30402(+/-)523
35000	30793	1450	30793(+/-)683
36000	31636	1637	31636(+/-)771
37000	32770	2071	32770(+/-)976
38000	33103	1506	33103(+/-)709
39000	33555	2536	33555(+/-)1195
40000	33796	1797	33796(+/-)846

Figure 3.3 shows the increase in the timing cost (Tc_1) for offloading sort service at runtime. It is examined that the T_{cm}, and T_{dv} remains constant in offloading sorting service with varying sort list size. However, the size of preferences files increases by increasing the length of sort list. Therefore, the cost of T_{ps} , T_{pu} , T_{pd} increases accordingly. It is

examined that the average T_{ps} cost is 2438 ms for saving preferences file of sorting list length 11000 values, whereas the average T_{ps} cost is 6739 ms for saving preferences file of sorting list length 40000 values. Hence, the T_{ps} cost increases 63.8 percent for saving preferences file of sort list length 40000 values as compared to saving the preferences file of sort list length 11000 values.

Similarly, it is examined that the T_{pu} cost increases according the size of preferences file. For instance, 253 ms time is taken in uploading preferences file for sorting list size 11000 values; whereas 873 ms time is taken uploading preferences file for sorting list size 4000. It shows that T_{pu} cost increases 71 percent in uploading the preferences file for the sorting list of 40000 as compared to the preferences file for the sorting list size 11000 values. It is examined that the T_{pd} cost increases according the size of preferences file downloaded to the local . For instance, 4620 ms time is taken in downloading the resultant preferences file for sorting list size 11000 values; whereas 16294 ms time is taken in downloading preferences file for sorting list size 4000. It shows that T_{pd} cost increases 71.6 percent in downloading the preferences file for the sorting list of 40000 as compared to the preferences file for the sorting list size 11000 values. By using equation (3.3), the total timing cost of offloading sorting service (Tc_1) is determined as 23779 ms. Analysis of the results indicates that saving preferences on the local, preferences uploading, reconfiguration on the remote server node and downloading resultant file to the local increase the timing cost in traditional computational offloading.

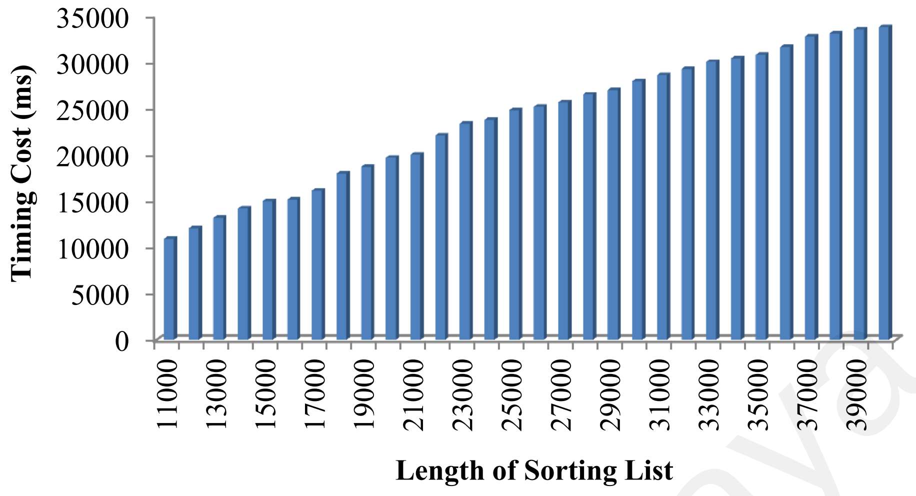

Figure 3. 3: Timing Cost (Tc_1) in Traditional Computational Offloading for Sort Service

The Tc_2 for offloading the matrix multiplication service component of the application at runtime is evaluated for 30 different computational intensities. The time taken in transferring application binary code (T_{cm}), is evaluated in 30 experiments by offloading matrix multiplication service application with 30 different computational intensities. It is examined that in all instances of offloading the binary code of the application, the size of binary application file (.apk) remains constant (46 KB). Hence, T_{cm} remains constant in offloading matrix multiplication service of the application with different computational intensities. It is examined that the sample mean of T_{cm} is 52(+/-)5 ms with 99% confidence for the sample space of 30 values, which shows that the possible range for T_{cm} is between 47 ms and 57 ms.

The timing cost of saving the data states (preferences file) on the local mobile device is examined for 30 different computational intensities of the matrix multiplication service component of the application. It is examined that the sample mean of T_{ps} is 28152(+/-) 11141 ms with 99% confidence for the sample space of 30 values which shows that the possible range for T_{ps} is between 17010 ms and 39293 ms. The timing cost of uploading preferences file (T_{pu}) to the cloud server node is examined for uploading the preferences file of 30 different computational intensities of the matrix multiplication service. It is examined that the sample mean of T_{pu} is 7177(+/-)3048 ms with 99% confidence for the sample space of 30 values, which shows that the possible range for T_{pu} is between 4128 ms and 10225 ms.

The timing cost of downloading the application file to remote virtual device instance (T_{dv}) is evaluated in 30 different experiments. It is examined that the confidence interval for the average T_{dv} is 205(+/-)15 ms for the sample space of 30 values in each experiment, which shows the possible range of value for T_{dv} is between 190 ms and 220 ms. The timing cost of application reconfiguration on remote server node and resuming time (T_{rr}) is evaluated in 30 different experiments. It is examined that the confidence interval for the average T_{rr} is 10349(+/-)2307 ms for the sample space of 30 values in each experiment, which shows the possible range of value for T_{rr} is between 8041 ms and 12656 ms.

The timing cost of downloading the resultant preferences file (T_{pd}) from the remote server node the local mobile device is evaluated in 30 different experiments. The sample mean of each experiment is determined with 99% confidence for the sample space of 30 values in each experiment. It is examined that the confidence interval for the average T_{pd} is 11238(+/-)2753 ms in downloading the resultant preferences file of matrix multiplication service for the matrices length 160*160-450*450 values.

The total timing cost (Tc_2) in runtime computational offloading of matrix multiplication is computed by using equation (3.3). Table 3.4 shows the total Tc_2 in offloading matrix multiplication service by using traditional computational offloading technique.

Table 3. 4: Timing Cost (Tc2) in Traditional Computational Offloading for Matrix Multiplication Service

Length of Matrix	Timing Cost (ms)	SD in Timing Cost	Confidence Interval
160*160	12001	596	12001(+/-)281
170*170	13150	1142	13150(+/-)538
180*180	15370	1984	15370(+/-)935
190*190	16136	1881	16136(+/-)886
200*200	18561	1454	18561(+/-)685
210*210	19663	1169	19663(+/-)551
220*220	20875	1185	20875(+/-)558
230*230	23677	1767	23677(+/-)832
240*240	26042	1451	26042(+/-)683
250*250	29881	1218	29881(+/-)574
260*260	31604	2017	31604(+/-)950
270*270	33983	2008	33983(+/-)946
280*280	36521	2612	36521(+/-)1230
290*290	41451	1535	41451(+/-)723
300*300	43681	2381	43681(+/-)1122
310*310	44719	4649	44719(+/-)2190
320*320	51008	6530	51008(+/-)3076
330*330	53588	4616	53588(+/-)2174
340*340	67642	5079	67642(+/-)2392
350*350	66899	6069	66899(+/-)2859
360*360	70484	5214	70484(+/-)2456
370*370	76762	7473	76762(+/-)3520
380*380	82148	6160	82148(+/-)2902
390*390	89802	12813	89802(+/-)6035
400*400	92389	7284	92389(+/-)3431
410*410	102987	11737	102987(+/-)5529
420*420	113381	12979	113381(+/-)6114
430*430	131396	20749	131396(+/-)9774
440*440	134848	20211	134848(+/-)9520
450*450	154495	26891	154495(+/-)12667

Figure 3.4 shows the increase in the timing cost (Tc_2) for offloading matrix multiplication service at runtime. It is examined that the T_{cm}, and T_{dv} remains constant in offloading matrix multiplication service with varying matrices size. However, the size of preferences files increases by increasing the length of matrices. Therefore, the cost of T_{ps} ,

T_{pu}, T_{pd} increases accordingly. The average T_{ps} cost is examined 3294 ms for saving preferences file of matrices length 160*160, whereas the average T_{ps} cost is 91038 ms for saving preferences file of matrices length 450*450.

Hence, the T_{ps} cost increases 96.3 percent for saving preferences file of matrices length 450*450 as compared to saving the preferences file of matrices length 160*160. Similarly, it is examined that the T_{pu} cost increases according the size of preferences file. For instance, 1518 ms time is taken in uploading preferences file for matrices length 160*160; whereas 20878 ms time is taken uploading preferences file for matrices length 450*450. It shows that T_{pu} cost increases 92.7 percent in uploading the preferences file for the matrices length 450*450 as compared to the preferences file for the matrices length 160*160.

It is examined that the T_{pd} cost increases according to the size of preferences file downloaded to the local mobile device. For instance, 3400 ms time is taken in downloading the resultant preferences file for matrices length 160*160 values; whereas 23015 ms time is taken in downloading preferences file for matrices length 450*450. It shows that T_{pd} cost increases 85.2 percent in downloading the preferences file for matrices length 450*450 as compared to the preferences file for the matrices length 160*160. By using equation (3.3) the total average timing (Tc_2) is 57171 ms in offloading matrix multiplication service at runtime. Analysis of the results indicates that in traditional computational offloading, preferences saving on the local mobile device, preferences uploading, reconfiguration on the remote server node and downloading resultant file to the local increase the timing cost.

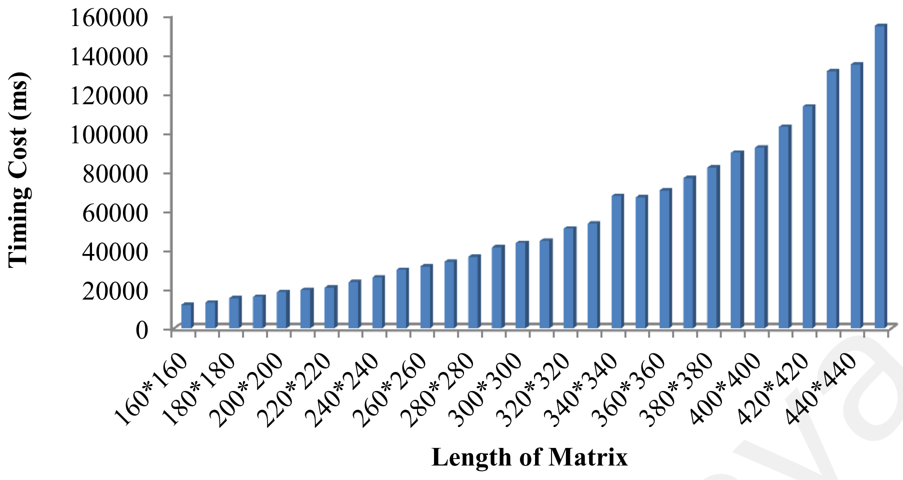

Figure 3. 4: Total Timing Cost (Tc2) in Traditional Computational Offloading for Matrix Multiplication Service

The total Turnaround Time (*TT*) in remote execution of the matrix multiplication service varies for different intensities of the matrix multiplication operation. It is examined that *TT* in the remote execution of matrix multiplication operation increases with the increase in the length of matrices. For instance, the total *TT* of multiplying matrices 160*160 length is 16431 ms, whereas the *TT* of multiplying matrices 450*450 length is 262697 ms. It shows that the *TT* increases 93.7 percent in multiplying matrices 450*450 length as compared to multiplying matrices 160*160 length. The average *TT* in offloaded execution of matrix multiplication service for multiplying matrices (160*160-450*450 length) is 91567.5 with 74.7 percent RSD.

Analysis of the results indicates that runtime computational offloading increase the *TT* of distributed application execution considerably. For instance, the additional timing cost (Tc_1) in traditional offloading of sorting service is 45 percent for sort list length 11000, 36 percent for sort list length 20000, 26 percent for sort list length 30000 and 20 percent for sort list length 40000. The average increase in the timing cost (Tc_1) of runtime computational offloading for sorting service is 31.1 percent for sort list length 11000-

40000. Similarly, the timing cost (Tc_2) in traditional offloading of matrix multiplication service is 73 percent for multiplying matrices 160*160 length, 67.5 percent for multiplying matrices 260*260 length, 61.2 percent for multiplying matrices 340*340 length and 58.8 percent for multiplying matrices 450*450 length. The average increase in the Tc_2 of runtime computational offloading for matrix multiplication service constitutes 65.2 percent for multiplying matrices 160*160-450*450 length.

We know that Tc_1 is 23779 ms (average timing cost of offloading sorting service) and Tc_2 is 57171 ms (average timing cost of offloading matrix multiplication service), hence by using equation (3.4) the total timing cost (α_t) of runtime computational offloading for the mobile application is calculated as 80950 ms, which means that in traditional computational offloading, 45.5 percent additional time is taken in offloading the components of the mobile application at runtime.

3.2.3 Analysis of the Size of Data Transmission

The Size of Data transmission (*Ds*) in runtime computational offloading involves the size of application binary file migrated at runtime (*Da*), the size of preferences file uploaded to cloud server node (*Dpu*) and the size of resultant preferences file downloaded to the local (*Dpd*). Therefore, the total size of data transmission of a single component of the mobile application which is being offloaded at runtime is given by the following equation.

$$Ds = Da + Dpu + Dpd \qquad (3.5)$$

Let D is the finite set of the size of data transmission of the components of the mobile application which are offloaded at runtime. Hence, set D is given as:

D= {the finite set of the size of data transmission of the components of the mobile application which are offloaded at runtime }

Let Ds_a represents the total size of data transmission in offloading a single component of the mobile application at runtime. Whereas, $a=1, 2,\ldots, n$

$$\therefore\ D = \{ Ds_1, Ds_2, \ldots, Ds_n \}$$

Ds_a represents the total size of data transmission in offloading a single component of the mobile application. Therefore, by using set builder notation Ds_a is represented as follows.

$$D = \{ Ds_a : Ds_a \in \mathbb{R} \wedge Ds_a > 0 \} \qquad \text{Whereas, } a=1, 2,\ldots,n$$

The total size of data transmission in offloading a single component of the mobile application belongs to the set of positive Real numbers and is greater than 0. The total size of data transmission in runtime computational offloading is the sum of size of data transmission of all the components of the application which are being offloaded at runtime. Let the total size of data transmission in runtime application offloading is represented by α_d, which is the sum of size of data transmission of all the instances $Ds_{a=1,2,\ldots,n}$ of the runtime component offloading. Therefore, α_d is represented as follows.

$$\therefore\ \alpha_d = (Ds_1 + Ds_2 + \ldots + Ds_n) \qquad \Rightarrow \forall\ Ds_a \in \mathrm{D} \wedge |\mathrm{D}| \geq 1 \quad \text{where } a=1,\ldots, n$$

By using summation notation the total size of data transmission of the runtime application offloading of the mobile application is represented as follows:

$$\alpha_d = \sum_{a=1}^{n} Ds_a \qquad \Rightarrow \forall\ Ds_a \in \mathrm{D} \wedge |\mathrm{D}| \geq 1 \qquad (3.6)$$

For all Ds_a, which denotes the size of data transmission of the single instance of runtime component offloading of the mobile application belongs to the set D and the cardinality of set D is greater than or equal to 1, whereas D is the finite set of the total size of data transmission of the components of mobile application which are offloaded at runtime. The precondition validates that D is none empty set.

The total size of data transmission in offloading sorting service (Ds_1) and total size of data transmission in offloading matrix multiplication service (Ds_2) of the application at runtime is computed by using equation (3.5). Table 3.5 shows the total size of data transmission for offloading sort service and matrix multiplication service of the application for varying length of either operation.

Table 3. 5: Size of Data Transmission in Runtime Computational Offloading for Sorting Service and Matrix Multiplication Service

Length of Sorting List	**Size of Data Transmission (*Ds*$_1$) KB**	**Length of Matrix**	**Size of Data Transmission (*Ds*$_2$) KB**
11000	752.4	160*160	5739.44
12000	820.4	170*170	6538.16
13000	888.4	180*180	7377.84
14000	950.4	190*190	8217.52
15000	1026.4	200*200	9118.64
16000	1086.4	210*210	10040.24
17000	1162.4	220*220	11023.28
18000	1230.4	230*230	12067.76
19000	1298.4	240*240	13132.72
20000	1360.4	250*250	14259.12
21000	1420.4	260*260	15426.48
22000	1480.4	270*270	16634.8
23000	1572.4	280*280	17884.08
24000	1632.4	290*290	19194.8
25000	1694.4	300*300	20526
26000	1754.4	310*310	21754.8
27000	1846.4	320*320	23393.2
28000	1914.4	330*330	24826.8
29000	1982.4	340*340	26465.2
30000	2042.4	350*350	28103.6
31000	2092.4	360*360	29742
32000	2153.84	370*370	31380.4
33000	2215.28	380*380	33223.6
34000	2276.72	390*390	34862
35000	2399.6	400*400	36705.2
36000	2399.6	410*410	38753.2
37000	2461.04	420*420	39367.6
38000	2522.48	430*430	42644.4
39000	2583.92	440*440	44692.4
40000	2645.36	450*450	46740.4

The size of data in transferring application binary code and preferences file is evaluated in 30 experiments for offloading both sorting service and matrix multiplication service of the application in 30 different experiments. It is examined that in all instances of sorting service offloading and matrix multiplication service offloading the size of binary application file (.apk) remains constant; 44.4 KB for sort service and 46 KB for matrix multiplication service. However, the size of preferences file uploaded to the cloud server node (*Dpu*) and the size of the resultant preferences file downloaded to the local mobile device (*Dpd*) varies for different length of both operations.

The size of data transmission in sorting service offloading is examined 752.4 KB for sort list length 11000, 1360.4 KB for sort list length 20000 and 2645.36 KB for sort list length 40000. It shows that the size of data transmission increases 71.6 percent in offloading sorting service with the length of sorting list 40000 as compared to the length of sorting list 11000. Similarly, the size of data transmission in matrix multiplication service is examined 5739.44 KB for matrices length 160*160, 15426.5 KB for matrices length 260*260 and 46740 KB for matrices length 450*450. It shows that the size of data transmission increases 87.8 percent for offloading matrix multiplication service with the matrices length 450*450 as compared to matrices length 160*160. The average size of data transmission (Ds_1) for offloading sorting service with the sort list length 11000-40000 is determined 1722.2 KB. Whereas, the average size of data transmission (Ds_2) for offloading matrix multiplication service with the matrices length 160*160-450*450 is determined 11474.3 KB.

The size of data transmission for offloading power compute service (Ds_3) at runtime is evaluated in 30 different experiments. It is examined that in all instances of offloading power compute service the size of binary application file (.apk) remains constant 42.7 KB , the size of preferences file uploaded to the cloud server node (*Dpu*) is 1 KB and the size of the resultant preferences file downloaded to the local (*Dpd*) is 1 KB. Hence, by using equation (3.5) the total size of data transmission in (Ds_3) is 44.7 KB for offloading power compute service as runtime. By using equation (3.6) the total size of data transmission (α_t) of runtime computational offloading for the mobile application is calculated as 13241.2 KB.

3.3 Analysis of VM Deployment for Application Processing

Virtual machine deployment based application offloading is a dominant computational outsourcing mechanism for cloud based application processing (Goyal and Carter, 2004; Satyanarayanan et al., 2009; Chun et al., 2009; Chun et al., 2011; Hung et al. 2011; Zao et al., 2011). In VM migration based application offloading, the deployment and management of VMs require additional computing resources on SMDs. The deployment of VM involves computing resources in the process of VM creation, VM configuration, VM OS setup, VM startup, and application deployment. The management of VM includes computing resources utilization in the monitoring of VM state transitions, CPU scheduling, VM migration, application processing management, VM state transitions and physical resources monitoring of computing host; allocation and de-allocation of physical resources such as CPU and memory. This section investigates the cost of VM deployment for application processing (Shiraz et al., 2013). We employ CloudSim for the evaluation of the impact of VM deployment for application processing. CloudSim is an extensible simulation framework that seamlessly models simulation and experimentation of cloud computing infrastructures and application services (Calheiros et al. 2011). The impact of VM deployment and management on the execution of the application is evaluated on the basis of the application allocation to VM time and application processing time.

The experimental model is composed of two major scenarios. First, we evaluate application allocation to VM time in different experiments. We conduct 30 different experiments to determine application allocation to VM time. The average value of each experiment is used for the analysis of application allocation to VM time. Second, the application execution time is evaluated in two different test beds. In test 1, the number of VMs equal to the number of applications in which case an individual VM is allocated to each application. In test 2, the number of VMs is reduced to half of the number of applications in which case each VM is shared by multiple applications.

In CloudSim, an application services is modeled by cloudlet. The execution of application in VM instance includes the following steps; application creation, application allocation to VM, application scheduling in VM, and application termination. In some scenarios application migration occurs in which case application is migrated to other VMs by deploying different migration policies. Application migration includes the computing cost for encapsulation of application states in VM, selection of appropriate remote host, transferring application to the remote host and allocation of application to a new VM in remote datacenter. It is examined that the average time consumed for the allocation of applications to VM increases for increasing number of VMs and applications. The allocation of two applications to two independent virtual machines takes on the average 10 ms in all instances of the experimentation. The allocation of eight applications to VM takes on the average 48 ms with the Relative Standard Deviation (RSD) 8.6 percent. Similarly, the allocation of 45 applications to VM takes on the average 430 ms with the RSD 3.1 percent.

Figure 3.6 shows increase in the time required for the allocation of application to VM. The mechanism of VM creation and application configuration in VM is time consuming. Therefore, the average time required for the allocation of the application to VM increases by increasing the number of VMs. Analysis of the results shows that allocation of application to VM require additional computing resources for configuration of VM according to the predefined specifications and the encapsulation of application in VM. As a result, it increases the execution cost and time of the application in VM based application offloading.

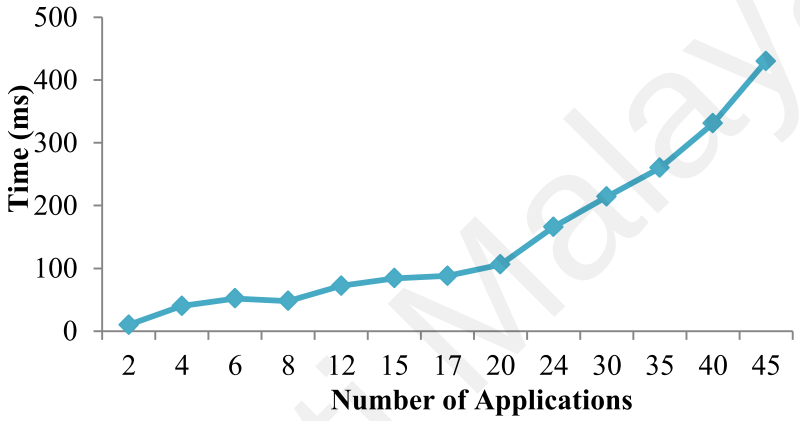

Figure 3. 5: Application Allocation to Virtual Machine Time (Shiraz et al., 2013a)

In test 1 of scenario 2, we evaluate the Application Processing Time (APT) of the application by creating an individual VM for each application. Hence, it does not involve the cost of VM scheduling. The processing time of applications (cloudlets) changes with different number of VMs. Figure 3.7 indicates the increasing trend in average APT. The average time required for the processing of application increases by increasing the number of VMs. The APT increases 28 percent for 2-5 applications, 55 percent for 15-20 applications and 65 percent for 30-15 applications. On the average the APT increases 49.8 percent with the RSD 48.2 percent for 2-45 applications.

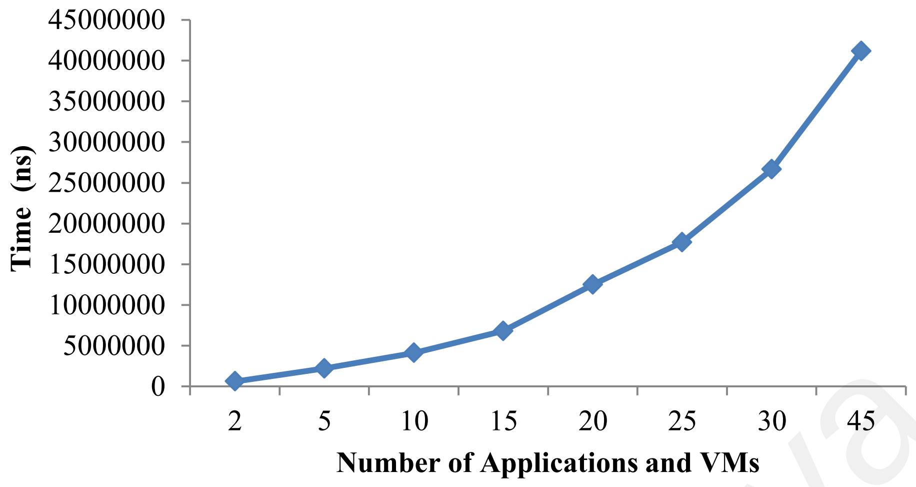

Figure 3. 6: Application Processing Time for None Shared VMs (Shiraz et al., 2013a)

In test 2 of scenario 2, we evaluate the APT of the application by allocating multiple applications to each VM which results in the sharing of VM among multiple applications. Sharing of VM involves the cost of scheduling VM resources among multiple applications which affects the processing time of the application. Figure 3.8 indicates the APT of the application for shared VM scenario. The APT of applications (cloudlets) changes differently with different number of VMs.

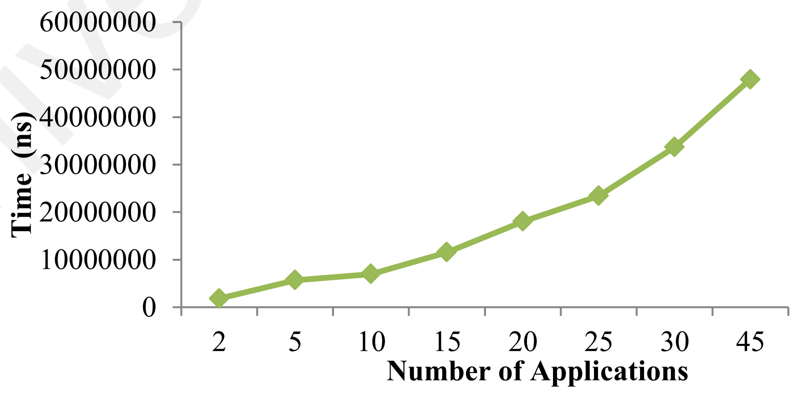

Figure 3. 7: Application Processing Time **(**APT) for Shared VMs

Analysis of the results indicates that on the average APT increases for each application with the increase in number of VMs. APT increases 32 percent for (2-5) applications, 60 percent for (10-15) applications, 77 percent for (20-25) applications, 70 percent for (30-45) applications. On average the execution time of an application increases by 64.7 percent for (2-45) applications.

Figure 3.9 compares the comparison of average application execution time in test 1 and test 2 of scenario 2. The analysis of the results in test 1 and test 2 indicates that APT of the application is 7.7 percent higher for shared VMs as compare to none shared VMs. It means that sharing of VM involves the additional cost of sharing computing resources among multiple applications.

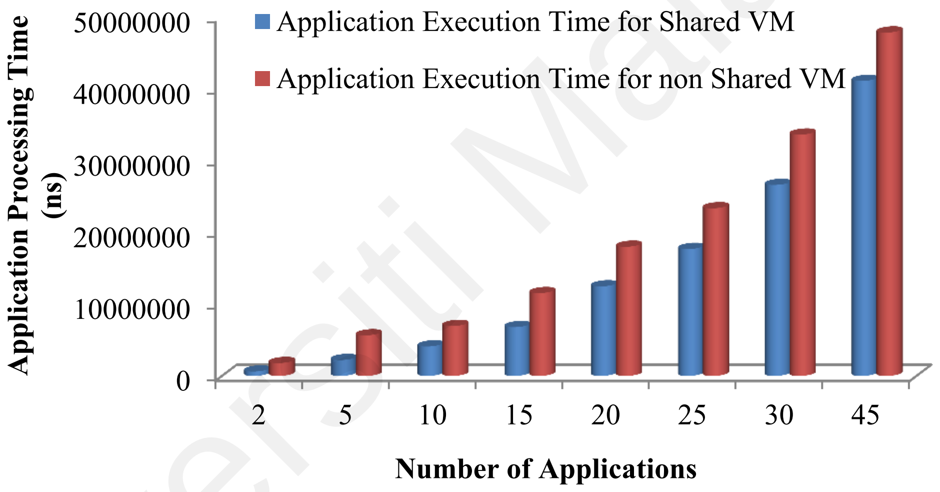

Figure 3.8: Comparison of APT for Shared VMs & None Shared VMs (Shiraz et al., 2013a)

The comparison of application processing time in different test cases shows that average processing time of the application increases for both shared and none shared VMs. It shows the additional computing resources utilization for the deployment of VM in application processing. It concludes that VM migration based application offloading requires additional computing resources on SMD for the deployment and management of VM which affects the execution cost and time of mobile application.

3.4 Conclusion

The traditional computational offloading frameworks for mobile cloud computing are the analogous extensions of earlier decentralized computational offloading frameworks for local distributed platforms. Current DAPFs are deficient in the deployment of distributed system architecture for the design and development of intensive mobile applications, which are offloaded to cloud server nodes. The traditional application offloading frameworks focus on the establishment of runtime distributed platform, which is a resources intensive mechanism. Therefore, resources intensive distributed application execution platform is established at runtime, which results in additional cost of application file and data file migration, high energy consumption cost in distributed application processing and longer turnaround time of mobile application.

It is examined that 13241.2 KB data is transmitted in traditional computational offloading of three intensive components of the mobile application, 75.2 J additional energy is consumed and 80950 ms additional time is taken in offloading the intensive components at runtime. It shows that 39.4 percent additional energy is consumed and 45.5 percent additional time is taken in offloading the components of the mobile application at runtime. Further, it is examined that the deployment for VM for application affects the execution cost of the application. Therefore, VM migration based application offloading requires additional computing resources for the deployment and management of VM on mobile device.

Traditional computational offloading frameworks lack of considering the intensity of runtime component offloading and focus on leveraging the IaaS service provisioning model for computational offloading which is resources intensive and time consuming.

Current frameworks lack of leveraging the application processing services of computational clouds by using the SaaS service provision model which provides fast and lightweight solution for cloud based application processing.

Hence, the traditional computational offloading frameworks employ heavyweight procedures for the processing of intensive applications in MCC. The resources limited features of mobile devices and the intrinsic limitations in the wireless access medium motivate for lightweight procedures for the processing of computationally intensive applications in MCC.

.

CHAPTER 4

Distributed and Elastic Application Processing (DEAP) Framework

This chapter reports on the methods and procedures for solving the problem of additional computing resources utilization in the processing of intensive mobile applications in MCC. The chapter is organized into four sections. Section 4.1 discusses introduction to the chapter. Section 4.2 explains the proposed framework for the distributed deployment of intensive mobile applications in MCC and explains the standard operation procedures of the proposed framework. Section 4.3 highlights the distinctive features of the proposed framework. Section 4.4 draws conclusive remarks with highlighting the usefulness and applicability of the proposed model.

4.1 Introduction

Traditional computational offloading frameworks employ heavyweight procedures for the processing of intensive applications in MCC. The resources limited features of mobile devices and the intrinsic limitations in the wireless access medium motivate for lightweight procedures for processing of computational intensive applications in MCC. Therefore, a lightweight framework is proposed for addressing the issue of additional computing resources utilization in the processing of intensive mobile applications in MCC. The architecture of the proposed solution is modeled and the operating procedure of the proposed framework is explained. The development of distributed applications on the basis of such lightweight framework results in substantial performance gains and enhancement in overall performance of application deployment and processing in MCC.

4.2 Proposed Lightweight Application Processing Framework

We propose a novel lightweight Distributed and Elastic Application Processing (DEAP) framework for MCC. DEAP addresses the issue of additional computing resources utilization in traditional computational offloading by focusing on minimal computing resources utilization in computational offloading and comparatively shorter turnaround time of the distributed application processing in MCC. DEAP fulfills the gap of traditional computational offloading frameworks by incorporating SaaS model with IaaS model of computational clouds for leveraging the application processing services. DEAP provides comparatively lightweight solution for the processing of intensive mobile applications in MCC.

DEAP incorporates the features of distributed application model with the elastic attributes of the traditional computational offloading frameworks. Therefore, the proposed model is distributed by design and elastic in nature. DEAP is attributed with the features of simple developmental procedures, standardized deployment principles and elastic processing management mechanism for intensive mobile applications. The distinctive features of DEAP are leveraging the SaaS model for the configuration of intensive components on the cloud server node and the incorporation of elasticity attributes for providing autonomy of mobile application and ensuring dynamic processing management on SMD. The distributed architectural attribute of DEAP framework allows mobile applications to use the application processing services of computational clouds without the additional cost of runtime application partitioning and component offloading. The issue of dependency on the preconfigured servers and autonomy of mobile application is addressed by including the elasticity features in mobile application. The elastic nature attribute of DEAP model enables mobile devices to offload the intensive components of the mobile application dynamically in critical conditions.

In traditional computational offloading frameworks computationally intensive components of the mobile application are annotated as local and remote at design time and the remotely annotated components are offloaded at runtime for remote processing (Messer et al., 2002; Giurgiu et al., 2009; Cuervo et al., 2010; Chun et al., 2011; Zhang et al. 2011), which results in additional cost of runtime component offloading. To address this issue, DEAP framework focuses on utilizing the SaaS service provision model of computational clouds for implementing preconfigured services on the cloud sever node which are accessed on demand basis. The preconfigured services include the resources intensive components of the mobile application which are not location aware and do not require user interaction.

The preconfigured services are provided access by using the on demand business model of cloud computing. The configuration of resources intensive components of the mobile application on the cloud server nodes results in minimal instances of offloading the components of mobile application at runtime. The significance of the utilization of preconfigured services in SaaS model on demand basis is twofold.

- The computational intensity of the mobile application is reduced and the intensive computation is performed on the powerful virtual machines in the cloud datacenters. Hence, the turnaround time of the application is reduced and computational resources utilization and energy consumption on the smart mobile device is minimized.
- Computational load of the mobile application is outsourced by eliminating the additional timing cost, energy consumption cost and the size of data transmission in computational offloading.

DEAP implements the two tiered architecture of distributed applications by explicitly defining the client mobile application and server mobile application. The server application is composed of the intensive components of the mobile application which are identified at design time. Client mobile application is a normal application with all the components available on local mobile device. However, two types of additional attributes are included in normal mobile application. 1) To implement the distributed features, the mobile application is enabled to switch to the client mode in the situations of accessing the services of preconfigured services on the cloud server node. 2) To implement the elastic features, mobile application is enabled to save its data states for the purpose of offloading at runtime.

The architecture and operation procedure of DEAP framework is different from the traditional client/server applications. The traditional client/server applications are called thin client applications. The client applications provide user agents for interaction with the local computer, whereas the processing logic is implemented on the remote server machines. Examples of such applications include web application, email application, social network applications such as Facebook, and video conferencing applications such as Skype application. In the traditional client/server model, the client component of the application becomes insignificant in the situations of inaccessibility of the server component. Therefore, DEAP framework is attributed with the features of offline usability, on demand access of the preconfigured cloud services and offloading computational load of the local mobile device in the situation of unavailability of sufficient resources for the processing of mobile application on local mobile device. Figure 4.1 shows the architecture of the proposed DEAP framework.

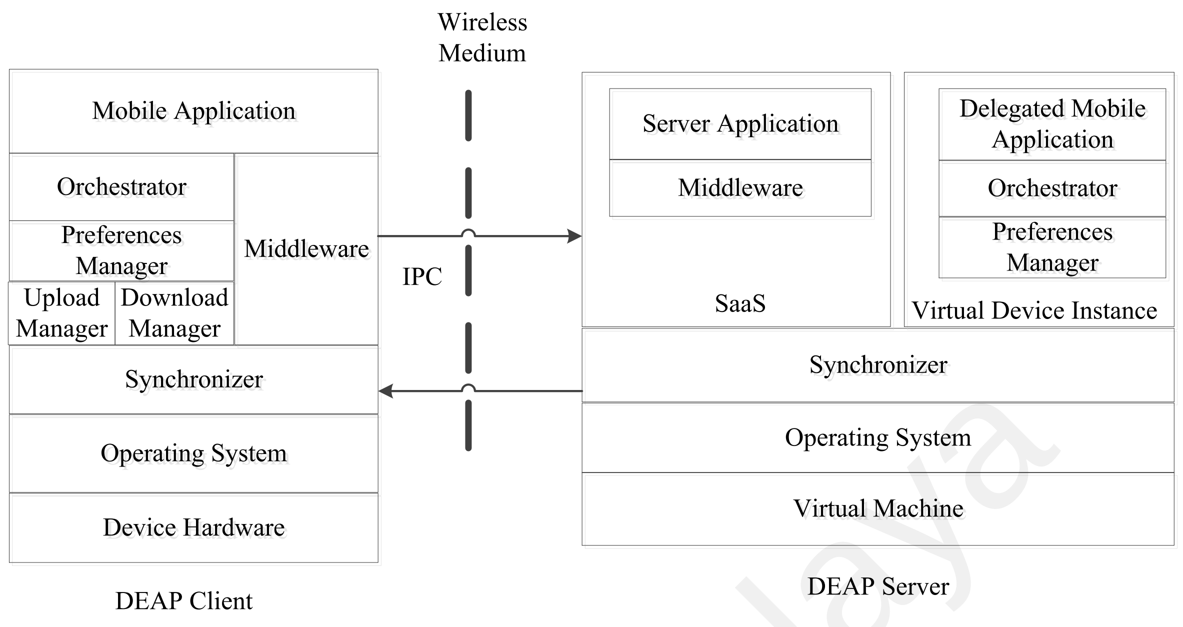

Figure 4. 1: Architecture of the Proposed DEAP Framework

DEAP configures the processing logic of the mobile application on the client mobile device and mobile application is enabled to be operated in the distributed and elastic manner on demand basis. Therefore, mobile applications are not completely dependent on the server component for application processing. The configuration of entire processing logic of the mobile application on local device assists in achieving the goals of rich user experiences and offline usability. It means that the client mobile application still remains functional in offline mode in the situations of unavailability of preconfigured services on the cloud server node. Mobile application is designed with the objectives to access the preconfigured services of DEAP server in the Primary Operating Procedure (POP) of distributed application processing and implement computational offloading at runtime in the Secondary Operation Procedure (SOP) of DEAP framework.

Considering the mobile nature and intrinsic limitations with wireless medium, DEAP is based on processing slight intensive or tightly coupled components of the mobile application on SMD which contributes to the richness and smartness of local services and offline usability of the mobile application. However in the scenarios of inaccessibility of remote servers the mobile application is capable to switch to offline mode, wherein the services of the local mobile application are activated to be executed on the local mobile device. Whenever, remote servers become accessible mobile application switches to the online mode to access the distributed services of clouds server node. The following section explains the components of the architecture of DEAP framework.

a) ***Middleware:*** DEAP client uses the services of distributed middleware for the implementation of primary operation procedure. The communication between client mobile application and server application is implemented by using Inter Process Communication (IPC) mechanism. Middleware hides the complications of the communication between DEAP client application and DEAP server application. A mobile user is provided the notion as the entire application is being processed locally on SMD. In the POP of DEAP framework, mobile application invokes the services of DEAP server application by using the distributed middleware.

b) ***Application Orchestrator***: The secondary operating procedure of DEAP client application uses the orchestrator component for the configuration of the operation modes of the client application. The orchestrator component monitors the operation of mobile application in two distinct modes on SMD; offline mode and online mode. In the offline mode the application offloading options are disabled and the components of the mobile application are executed on SMD. In the online mode the services of DEAP server are utilized in the POP for accessing the preconfigured services of the server application. Whereas, in the situation of of unavailability of preconfigured services in the DEAP server, the options of offloading application components are enabled and mobile users are capable to offload the selected components of the mobile application at runtime. In the SOP of online mode, application orchestrator activates the preferences manager component to save the data states of the running mobile application. Application orchestrator is responsible for the configuration of the mobile application on SMD and remote server node. On the cloud server node, application orchestrator configures the delegated service application on the remote server node. Application orchestrator component on the remote server node resumes the running state of the delegated mobile application by accessing the preferences files from the persistent storage. The application orchestrator component of the DEAP client arbitrates with the remote server node for offloading the selected running component of the mobile application.

c) Preferences Manager: In the SOP of the mobile application, the orchestrator component is assisted by the preferences manager component. Mobile applications are associated with a separated preferences manger which provides access to the preferences file. Preference manager reads and writes the data states from persistent storage during the activation and deactivation of mobile application. In the SOP, the preferences manager component is activated to save the data states of the running component of the client mobile application. Preferences manager saves the data states to the persistent medium. The role of preferences manager is to provide access to the preferences of the mobile application. The preferences manager components copies the preferences file to the external storage device which is directly accessible for the upload manger and download manager component. The preferences manager component of the server node is responsible for providing access to the data files which are downloaded with the delegated service application. Similarly, whenever the service application completes execution on the remote server node, the preferences manger saves the final results to the preferences file.

d) ***Upload Manager***: The upload manager component of DEAP client is responsible for uploading the preferences files of the application to remote server node in the SOP. The preferences files are stored on the persisted storage by the preferences manager which is accessible for upload manager. Whenever, the offloaded mobile application is installed on the remote virtual device instance, the synchronization manger component of the DEAP server connects to the upload manger component of the DEAP Client and make request for preferences file. Upload manager sends the requested preferences file to the synchronizer of the DEAP server.

e) ***Download Manager***: In the SOP of DEAP client, the download manager component of DEAP client is responsible for downloading the preferences files of the application from the remote server node in the SOP. Whenever, the offloaded component of the application completes execution, download manager component of the DEAP client is connected to return the resultant preferences file. Download manager receives the resultant data file and saves it to the persistent storage of the local mobile device.

f) ***Synchronizer***: The synchronizer component of the framework is responsible for the synchronization of transmission between SMD and remote server node. In POP of distributed processing, the synchronizer component is responsible for ensuring the consistency of transmission between mobile application on SMD and the server application running on the cloud server node. In SOP, whenever the states of the application are saved on the persisted medium, the synchronizer component is activated to offload the service application to remote server node. The orchestrator component searches for the configuration file of the identified intensive component on mobile device. Whenever the configuration of the service application is validated, the synchronizer component is activated to outsource the configuration files to remote server node. Synchronizer component of the remote server node is activated to receive the delegated service application. Whenever, the configuration file of the delegated service application is received successfully on the remote server node, the orchestrator component of the server node is activated to configure the delegated service application and resume the running states from the preferences file.

The synchronizer component is also responsible for the uploading and downloading of preferences files between SMD and remote server node. The synchronization manger component of the DEAP framework utilizes the services of download manager and upload manager for the synchronization of distributed application processing in the SOP DEAP framework. In the SOP DEAP client offloads the intensive components of the application to cloud datacenters which are executed on temporarily created server node. In such scenario the role of synchronizer is to coordinate between DEAP client mobile application and the offloaded components of the application. The primary responsibility of synchronizer is to ensure the consistency of transmissions between DEAP client and DEAP sever in POP and DEAP client application and temporarily allocated server node in the SOP.

Figure 4.2 shows the flowchart for the interaction of the components of DEAP framework in leveraging application processing services of cloud server node. The DEAP client mobile application executes on SMD, whereas the DEAP server application is configured on the cloud sever node. Whenever, the client application requires the services of remotely configured component in DEAP server, it activates the component by using IPC. In the online mode the services of cloud server nodes are leveraged for the processing of intensive components of the mobile application. In the POP, DEAP client access the preconfigured services of DEAP server. However, in the scenario of unavailability of the preconfigured server, the elastic features of DEAP client are used to offload the intensive components of the application to the remote server node at runtime.

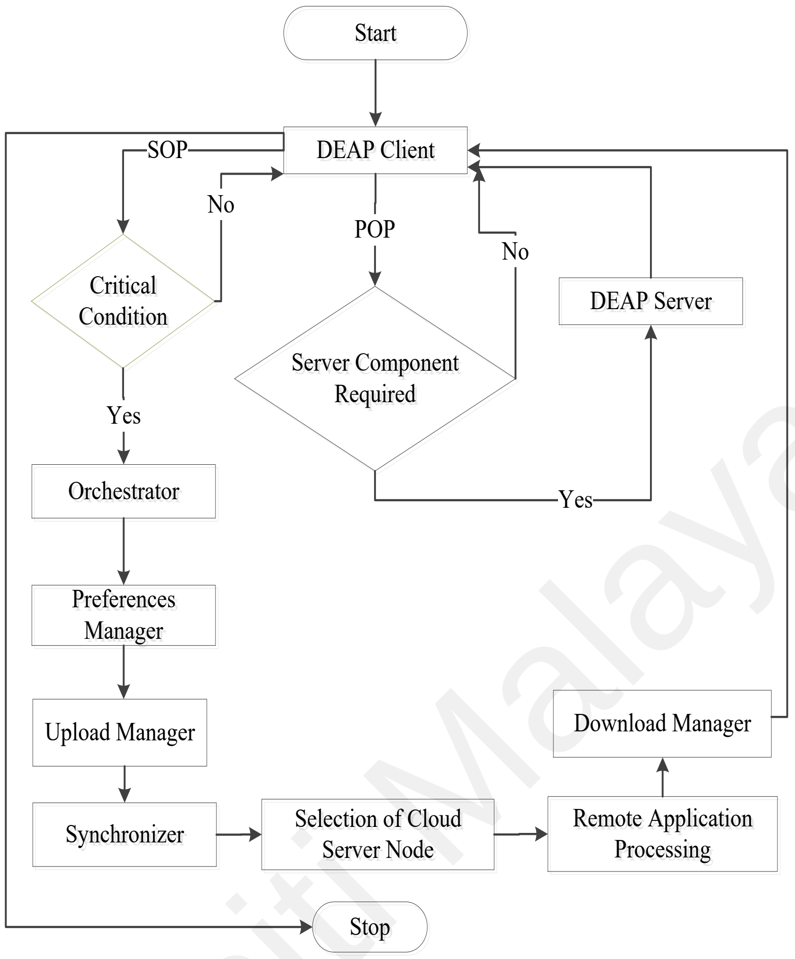

Figure 4. 2: Illustration of the Interaction of the Components of DEAP Framework in POP and SOP

DEAP framework proposes two independent operating procedures for the implementation of distributed platform of intensive mobile applications in MCC. The Primary Operating Procedure (POP) of DEAP client application implements distributed middleware for accessing the services of explicitly configured DEAP server. Mobile application activates the preconfigured services of DEAP server on demand basis. The client application uses IPC procedures for invoking the services of remote server node. The POP of DEAP client follows a simple and optimal procedure for remote processing of intensive components of the mobile application. The significant aspect is that it provides cloud based processing of mobile application without the overhead of runtime application partitioning and component offloading. Figure 4.3 shows the sequence diagram for POP of DEAP.

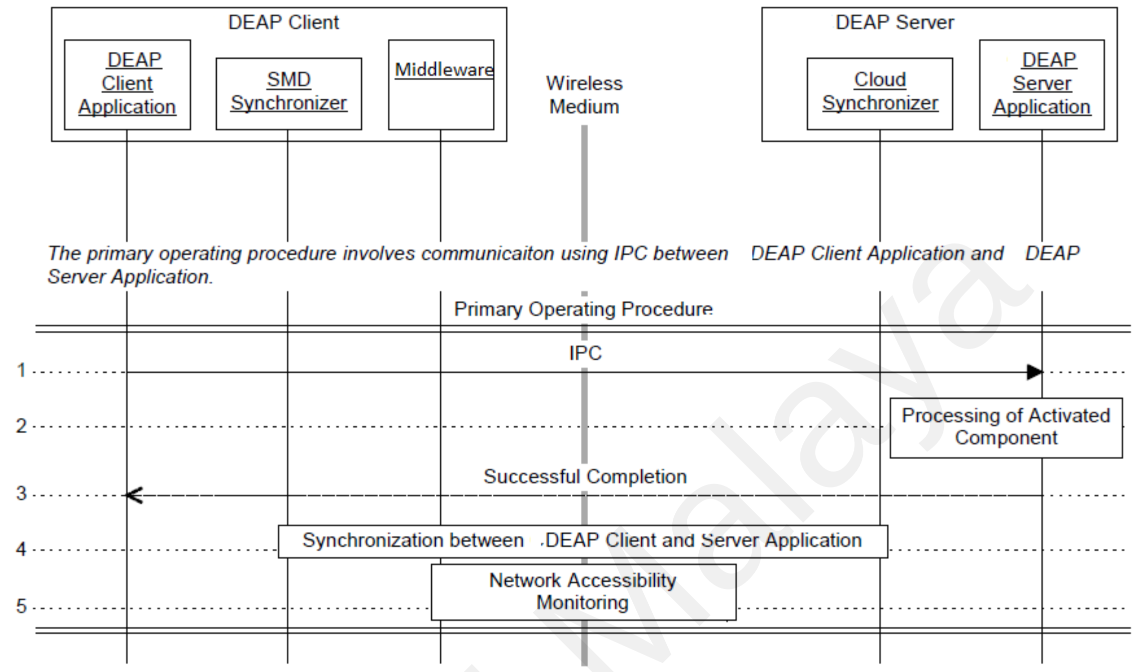
DEAP Client
DEAP Client Application
SMD Synchronizer
Middleware
Wireless Medium
DEAP Server
Cloud Synchronizer
DEAP Server Application
The primary operating procedure involves communicaiton using IPC between DEAP Client Application and DEAP Server Application.
Primary Operating Procedure
1
IPC
2
Processing of Activated Component
3
Successful Completion
4
Synchronization between DEAP Client and Server Application
5
Network Accessibility Monitoring

Figure 4. 3: Sequence Diagram for Primary Operating Procedure of DEAP Framework

The Secondary Operating Procedure (SOP) of DEAP incorporates the elasticity features for coping with the mobile application processing requirements and processing loads on SMD. DEAP client employs SOP in critical condition and online mode, wherein insufficient resources are available on local mobile device and the services of the preconfigured services are inaccessible. Therefore, DEAP client employs SOP for offloading the intensive component of mobile application at runtime. In the SOP, DEAP client follows service level granularity for offloading the intensive components of the mobile application. Further, the proposed model reduces the overhead of application outsourcing by eliminating the mechanism of runtime application profiling and solving. In SOP, DEAP client mobile application starts execution on the SMD with the activity component of mobile application. The interface of mobile application displays the operations for the operation mode of the mobile application. Figure 4.3 shows sequence diagram for the SOP of DEAP framework.

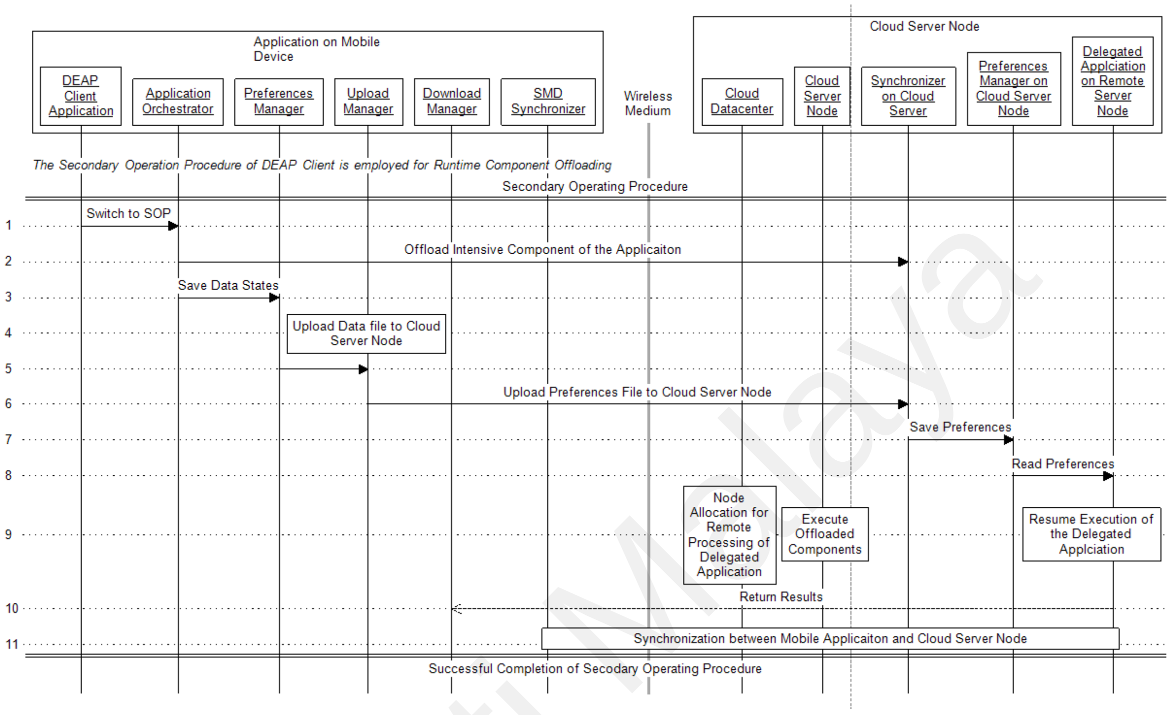
Application on Mobile Device
DEAP Client Application
Application Orchestrator
Preferences Manager
Upload Manager
Download Manager
SMD Synchronizer
Wireless Medium
Cloud Server Node
Cloud Datacenter
Cloud Server Node
Synchronizer on Cloud Server
Preferences Manager on Cloud Server Node
Delegated Applciation on Remote Server Node
The Secondary Operation Procedure of DEAP Client is employed for Runtime Component Offloading
Secondary Operating Procedure
Switch to SOP
Offload Intensive Component of the Applicaiton
Save Data States
Upload Data file to Cloud Server Node
Upload Preferences File to Cloud Server Node
Save Preferences
Read Preferences
Node Allocation for Remote Processing of Delegated Application
Execute Offloaded Components
Resume Execution of the Delegated Applciation
Return Results
Synchronization between Mobile Applicaiton and Cloud Server Node
Successful Completion of Secodary Operating Procedure

Figure 4. 4: Sequence Diagram for Secondary Operating Procedure of DEAP Framework

In the SOP of the online mode, mobile application is enabled to offloaded intensive components of the application. The synchronizer component arbitrates with the cloud datacenter for the selection of remote server node. A fresh VM instance is created on the cloud server node for the execution of delegated service application. At that time, the orchestrator on the mobile device saves the running states of the service application by activating the preferences manager and kills the selected service to release the systems resource occupied by the selected intensive service. The synchronizer component offloads the service application to remote service node. The orchestrator component configures the delegated service application and resumes the running states of the service application in the guest VM instance created on the server node. The synchronizer components of both the SMD and cloud server node communicate for the exchange of configuration and data files. On successful execution of the service application components of the mobile application on SMD results of the mobile application are saved in the preferences file and returned to the SMD. The robust nature of SOP is that it allows mobile user to switch between online mode and offline mode at any instance of mobile application execution. Figure 4.5 shows the flowchart for the operational logic of DEAP framework.

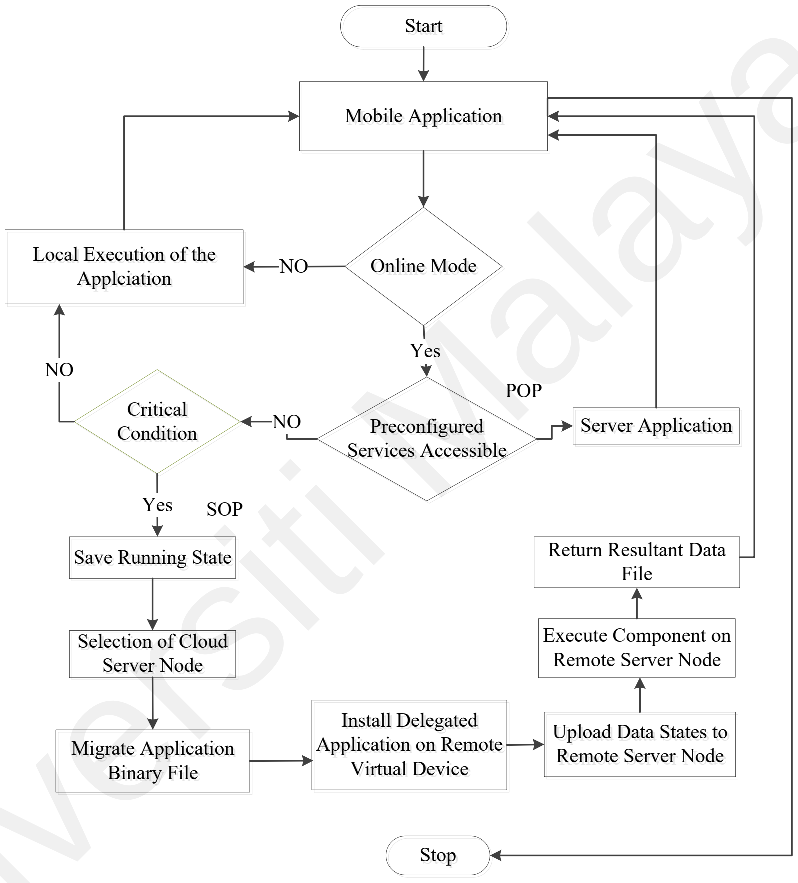
Start
Mobile Application
Local Execution of the Applciation
NO
Online Mode
Yes
NO
Critical Condition
NO
Preconfigured Services Accessible
POP
Server Application
Yes
SOP
Save Running State
Selection of Cloud Server Node
Migrate Application Binary File
Install Delegated Application on Remote Virtual Device
Upload Data States to Remote Server Node
Execute Component on Remote Server Node
Return Resultant Data File
Stop

Figure 4. 5: Illustration of the Operation Logic of DEAP Framework

The important aspect of the SOP in DEAP framework is that the service application package is transferred only once to the remote server node. However, configuration and data files require repeated transmission for each instance of remote execution of the service application. It means that at first instance the entire service application package file and the other related files are transferred to cloud server node. However, if the same service is required to be executed again on the same cloud server node, it does not require the application package to be migrated repeatedly. Instead, for the later instances of remote service execution require to upload the configuration and data files in order to synchronize the execution of service application on the remote server node. It is important to highlight that the SOP does not utilize the services of explicitly defined DEAP server. It is possible that the explicitly configured server application and online delegated components of the mobile application execute on two separate server nodes. The synchronizer component of DEAP client arbitrates with cloud servers to facilitate remote execution services on casual basis.

The distinctive aspects of DEAP framework are the design time classification of resources intensive services and the user preferences based migration of the running service application. The SOP of DEAP model employs simple developmental procedure. Unlike the traditional elastic application offloading models (Giurgiu et al., 2009; Cuervo et al., 2010; Zhang et a., 2011), DEAP model does not bound application developers to classify and annotate the application components as local or remote at finer granularity level. DEAP models entire service level granularity for the application offloading, which reduces the overhead associated with finer level granularity nature of traditional application offloading frameworks (Giurgiu et al., 2009; Cuervo et al., 2010; Zhang et a., 2011). It eradicates the cost of runtime application profiling and solving. The framework focuses on the user preferences for offloading the intensive components of the mobile application at runtime. Therefore, the service level migration is a lightweight mechanism for the establishment of distributed application processing platform at runtime. By deploying the SOP of DEAP framework mobile users have full control over the execution mode of the mobile application.

In offline mode all the components of mobile application are executed locally on SMD. On the other side, mobile user is provided with the option to offload the intensive services on demand basis. In the online mode, mobile application is enabled to access the preconfigured services of DEAP server and offload the intensive components of the application on demand basis. The dual operation modes of the DEAP client application provide robustness to the mobile applications. The applications are capable to operate with full functionalities in the situations of remote server access problems. The offloading of active service to cloud server node involves complicated mechanism. However, mobile users remain unaware of the complications of the remote execution. Mobile user is given the notion as entire components of the mobile application are executed locally on SMD. The following section highlights the distinctive features of DEAP framework.

4.3 Distinguishing Features of DEAP Framework

The following are distinguishing features of the proposed DEAP framework which make it a distinct framework for intensive mobile application processing in MCC.

4.3.1 Standardized Developmental and Deployment Procedures

The DEAP framework focuses on design time identification of the intensive components of the mobile application which provides design time support for the distributed deployment of intensive mobile application. Design time separation of the intensive mobile application reduces the developmental efforts of annotating all the individual components of the mobile application as local or remote, which makes the developmental procedures simple for the application developers. The DEAP framework explicitly defines the roles and responsibilities of the distributed components (client and server) of the mobile application. In current DAPFs the roles of distributed components participating in distributed platform remain unclear which results in complex runtime distributed platform. The DEAP framework incorporates two distinct operating procedures for inheriting the attributes of both distributed and elastic models as compared to the traditional DAPFs which employ exclusive elasticity attributes for intensive mobile applications.

4.3.2 Optimal Communication Procedures

The POP of the proposed framework minimizes the communication cost between SMD and cloud datacenters by employing IPC procedures rather than intensive partition migration. The minimization of communication overhead results in the following performance gains:

a) Communication over wireless medium is an energy starving operation, therefore minimizing data transmission overhead reduces energy consumption on SMD.
b) Application partition migration involves the issues of security in the wireless medium, therefore DEAP framework reduces the threats of network security by minimizing the migration of actual application or partitions of the application and active data file.
c) Current VM migration based DAPFs involves the overhead of VM deployment and management on SMD. However, DEAP implements application level IPC procedures for communication between DEAP client and DEAP server in POP and DEAP client and temporary cloud server in SOP which eliminates the deployment and communication overhead associated with VM deployment and migration for application transfer to the cloud servers.
d) Communication overhead over the wireless medium is highly error prone and subjected to attenuation distortion. DEAP framework focuses on minimization of runtime mobile application transmission by configuration maximum possible intensive services of the mobile application on the DEAP server which indirectly reduces the error rate of communication between SMD and cloud datacenters.

4.3.3 Elasticity and Robustness in Deployment

The proposed framework sustains the robustness and versatility of the elastic application models for coping with dynamic processing loads on SMD. The SOP of the DEAP framework is elastic and dynamic by nature. The elasticity attribute of the application enables SMD to dynamically offload computational load on the SMD to the remote server node. Current DAPFs employ runtime optimization either statically or periodically on SMD which are not appropriate for optimal deployment of distributed platform. The elasticity features of DEAP framework are employed for runtime intensive components optimization on the basis of user priorities. The SOP implements the activation of runtime optimizer whenever the critical condition occurs on SMD.

4.3.4 Convenient Application Level Deployment

DEAP model is based on the application level deployment of both the POP and SOP. The significant aspect of sustaining elasticity features in DEAP framework is the application partitioning and migration which does not require for additional operating system level support for partition migration as required in VM migration based approaches (Goyal and Carter, 2004; Satyanarayanan et al., 2009; Chun et al., 2009; Chun et al., 2011; Zao et al., 2011).

4.3.5 Offline Usability, Richness of Local Services and Smartness in Behavior

DEAP implements the two tiered architecture of distributed applications by explicitly defining the client mobile application and server mobile application. The server application is configured with the highly intensive components of the mobile application which are identified at design time. Mobile application is a normal application with all the components available on local mobile device. However, two types of additional attributes are included in normal mobile application. Mobile application is enabled to switch to the client mode in the situations of accessing the services of preconfigured services on the cloud server node. Similarly, mobile application is enabled to save its data states for the purpose of offloading at runtime.

DEAP focuses on the enrichment of services on SMD which contributes to the features of rich internet applications for MCC, smartness of client application and offline usability. DEAP framework is capable to provide local services on SMD in the failure or unavailability of internet access. However, the services DEAP server application is accessible only in online mode.

4.4 Conclusion

We propose distributed and elastic application processing framework for intensive mobile applications as a lightweight solution for the processing of intensive mobile applications in MCC. Traditional DAPFs are based on the establishment of distributed platform at runtime and lack of distributed architecture for the intensive mobile applications. The distinctive features of DEAP framework are leveraging the SaaS model for the configuration of intensive components on the cloud server node and the incorporation of elasticity attributes for providing autonomy of mobile application and ensuring dynamic processing management on SMD.

The distributed architectural attribute of DEAP framework allows mobile applications to use the application processing services of computational clouds without the additional cost of runtime application partitioning and component offloading. The issue of dependency on the preconfigured servers and autonomy of mobile application is addressed by including the elasticity features in mobile application. The elastic nature attribute of DEAP framework enables mobile users to offload the intensive components of the mobile application dynamically in critical conditions.

The incorporation of distributed model with elastic attributes of the traditional DAPFs facilitates in the simple developmental procedures, mobile application distributed by design, explicitly defined roles of the distributed components of the application, optimal deployment procedures with minimal cost in the establishment of distributed platform and comparatively minimal data transmission for the processing of intensive mobile applications in MCC. The dual operating nature of the proposed framework contributes to the versatility and robustness of the distributed and elastic model for intensive mobile application in MCC. The elasticity attributes of client mobile application enables SMD to cope with the challenges of dynamic application processing loads. Further, the elastic nature of DEAP client contributes to the objectives of offline usability, smart client and rich internet applications for MCC. It is concluded that DEAP framework provides a lightweight solution for the processing of intensive mobile applications in MCC.

CHAPTER 5

Evaluation

This chapter reports on the data collection method for the evaluation of proposed DEAP framework. It explains the tools used for testing the proposed framework, data collection technique and the statistical method used for the processing of data. The chapter is organized into seven sections. Section 5.1 presents an overview of the chapter, Section 5.2 explains the experimental setup and programming tools used for the implementation and testing of the proposed DEAP framework and the statistical method used for the compilation of empirical data. Section 5.3 presents the data collected in evaluating the execution of mobile application on local mobile device. Section 5.4 summarizes data collected in evaluating application execution in the traditional runtime component offloading. Section 5.5 presents data collected for testing the operating procedures of DEAP framework. Section 5.6 presents mapping of data by comparing experimental results in different scenarios. Finally, section 5.7 extracts conclusive remarks.

5.1 Introduction

DEAP framework provides a lightweight solution for the processing of intensive mobile applications in MCC. The significance of DEAP framework is evaluated in emulation and real mobile cloud computing environment. Synthetic workload is tested on the Android virtual device instance which is enabled to operate in the distributed mobile cloud computing environment. Experimental results are validated by benchmarking in the real distribute mobile cloud computing environment. A prototype application is developed for the Android devices, which is tested with varying computational intensities in three different experimental scenarios. The execution behavior of the application is analyzed from the perspective of resources utilization on the mobile device, size of data transmission on the wireless network medium, and turnaround time of the application in the traditional and proposed computational offloading techniques. Empirical data are collected by testing each component of the prototype application with 30 different computational intensities. The experimental results of the application are analyzed by collecting data from 30 samples. The sample mean for each sample space of 30 values is determined, which is signified by measuring the error estimate for 99% confidence interval. Finally, empirical results of testing the prototype application in the traditional and proposed technique are compared to validate the significance the proposed solution.

5.2 Evaluation of the Proposed DEAP Framework

This section presents the methodology used for the evaluation of DEAP framework. It discusses the experimental setup, prototype application used for the evaluation and the statistical method used for the compilation of results.

5.2.1 Experimental Setup

The proposed framework is evaluated by implementing synthetic workload in the emulation and real implementation on the physical mobile devices. Synthetic workload is tested on the Android virtual device instance which is enabled to operate in the distributed mobile cloud computing environment. Experimental results are validated by benchmarking in the real distribute mobile cloud computing environment. The following section describes the experimental setup for the emulation and real implementation environment.

The experimental setup for the emulation environment is composed of remote server machine and laptop computer. The server machine runs Microsoft Windows 7 Professional 32-bit operating system with Intel® core(TM) i5-2500 CPU having 3.3GHz speed and 4.0 GB RAM capacity. The laptop computer runs Microsoft Windows 7

Professional 32-bit operating system with Intel® core(TM) i5-2410M CPU having 2.30GHz speed and 4 GB RAM capacity. The emulator instance of the Android Virtual Device (AVD) runs on the laptop computer. The AVD instance runs Android 4.1-API Level-17 with ARMv7 Processor having 2389.08 BogoMIPS speed and 1GB RAM capacity. The AVD instance running on the laptop computer is connected to the D-Link Wireless Access Point providing 802.11g Wi-Fi wireless network connection of radio type 802.11g, with the available physical layer data rates of 54 Mbps.

Similarly, TP-Link wireless Wi-Fi modem is connected to the remote server machine in order to connect it to the Wi-Fi wireless network of radio type 802.11g. The experimental setup for testing the prototype application in the real wireless mobile network environment is composed of Wi-Fi wireless network of radio type 802.11g, Server machine and Samsung Galaxy SII mobile device. The Samsung smartphone runs Android 4.0.3, dual core ARMv7 Application Processor with 1.2 GHz (2389.08 BogoMIPS) speed, 16GB memory capacity and 1650mAh battery. Mobile device accesses the wireless network via Wi-Fi wireless network connection of radio type 802.11g, with the available physical layer data rates of 54Mbps.

The DEAP client application runs on the mobile device, whereas the DEAP server component of the application runs on the remote cloud server machine. The server machine is configured for the provisioning of services to the mobile device in two distinct operating modes of the application. DEAP server application utilizes the Software as a Service (SaaS) model of cloud computing for the provisioning of distributed services in the POP of DEAP client application, whereas the Infrastructure as a Service (IaaS) model (Buyya et al., 2009) is employed for the provisioning of services in the SOP of DEAP client application. We employ the DEAP server application by using Microsoft Visual Studio 2010 ASP.NET

Web Service Application tool of Visual C#, whereas kSOAP2 API (kSOAP2) is employed for the configuration of DEAP client application.

Java based Android Software development toolkit (Android Developers) is deployed for the development of DEAP client application. The Android ADB Plugin (Android Debug Bridge) is embedded in the Eclipse application development tool for the development of prototype application.

The POP of the DEAP client application is implemented by using kSOAP2 library API on the mobile devices for accessing the preconfigured services of DEAP server application. The AVD instance is created on the remote server machine by using Android emulator for the execution of offloaded components of the DEAP client mobile application in the SOP. Monitoring tools such as Android Debug Bridge (ADB) and Dalvik Debug Monitor System (DDMS) are used for the measurement of resources utilization (CPU and RAM), whereas Power Tutor tool is used for the measurement of battery power consumption in distributed application processing.

5.2.2 Prototype Application

The proposed DEAP framework is implemented by developing prototype application for Android devices. The prototype application is composed of three computational intensive service components and a single activity component. The service components implement the computational logic of the application, whereas the activity component provides Graphical User Interface (GUI) for interacting with the mobile application. The computational logic of the application includes the following service components. 1) Sorting service component implements the logic of bubble sorting for sorting liner list of integer type values. The sorting logic of the application is tested with 30 different computational intensities (11000-40000). 2) The matrix multiplication service of the application implements the logic of computing the product of 2-D array of integer type values. Matrix multiplication logic of the application is tested with 30 different computational intensities by varying the length of the 2-D array between 160*160 and 450*450). 3) The power compute service of the application implements the logic of computing b^e, whereas b is the base and e is the exponent. The power compute logic of the application is tested for 30 different computational intensities by varying the exponent between 1000000 and 200000000. The computational logic of the prototype application is tested with 30 different computational intensities. Empirical data are collected by sampling all computational intensities of the application in 30 different experiments.

5.2.3 Data Gathering and Data Processing

The primary data are collected by testing the prototype application on both the Android virtual device instance and real distributed mobile cloud environment in three different scenarios. In the first scenario, all the components of the mobile application are executed on the local mobile device to analyze resources utilization and execution time of the application on the local mobile device. In the second scenario, the intensive components of the mobile application are offloaded at runtime by implementing the traditional computational offloading technique. In this scenario, the resources utilization in distributed processing of the application, data communication over the wireless network medium, and turnaround time of the application on the virtual device instance on the remote server machine are analyzed.

In the third scenario, the prototype application is tested for the operating procedures of the proposed DEAP framework. We evaluate resources utilization and execution cost of the application in the POP and SOP of the DEAP framework. The following parameters are used for analyzing the prototype application in the local and distributed processing of the application in the emulation and real time environment.

1) CPU utilization in Millions of Instructions Per Second(MIPS),
2) Memory allocation in Mega Bytes (MB),
3) Energy consumption in Joule (J),
4) Turnaround time of the application in Millisecond (ms).
5) The size of data transmission over the wireless network in Kilo Bytes (KB).

According to the sample central limit theorem, approximately 99% of the sample means fall within 2.58 standard deviation of the population mean, provided that the sample size is greater than or equal to 30 ($n \geq 30$). Hence, the prototype application is composed of three computational intensive service components, and each component of the application is evaluated on the basis of five parameters with 30 different computational intensities. The empirical data are collected for all the computational intensities of every component of the mobile application by executing the component of the mobile application in 30 experiments. Each experiment is conducted 30 times for the evaluation of each parameter to derive the value of point estimator.

The measurement of central tendency of the data sample of each experiment is calculated by using sample mean $(\overline{X})$, for the reason that sample mean is ascertained the better point estimate of the population mean as compared to median or mode (Confidence Intervals and Sample Size, n.d.). Data sampling involves the factor of sampling error; hence the sample mean can differ from the population mean. Hence, to signify the goodness of the calculated point estimate; the interval estimate of each sample is determined. The interval estimate of a parameter represents the interval or range of values used to estimate the parameter. The confidence level of an interval estimate of a parameter indicates the probability that the interval estimate contains the parameter.

Let E represents the error estimate for 99% confidence interval, which is calculated by using the following equation.

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

Whereas, σ indicates the standard deviation in the sample values and n indicates the size of sample space. The interval estimate for each sample mean $(\overline{X})$ of the primary data is calculated with 99% confidence interval by using the following equation.

$$\text{Confidence Interval}= (\overline{X}) \pm E \tag{5.2}$$

The following section presents the data collected in different experiments for the evaluation of DEAP Framework on the Android virtual devices and real time environment. The data are presented from the perspective of three different scenarios: 1) Execution of the application on local mobile device, 2) Execution of the application by employing contemporary runtime offloading method, and 3) Execution of the application by the POP and SOP of the proposed DEAP framework.

5.3 Data Collected for Application Execution on the Local Mobile Device

In this scenario mobile application is executed on the local mobile device in order to evaluate the RAM allocation, Turnaround Time (TT), total Energy Consumption Cost (ECC) and CPU utilization in the execution of application on the local mobile device. Table 5.1 represents the statistics of RAM allocation for the execution of sorting component of the application on local mobile device. The computational length attribute indicates the computational intensity of the sorting operation on local mobile device which varies from 11000-40000 values in 30 different experiments. The sample mean attribute shows the point estimator for 30 different samples of the application execution with the identical computational intensity. The SD attribute shows the variation in the memory allocation to the application, whereas the %RSD attribute shows the percentage values of the variation in the sample space of 30 values in each experiment of evaluating RAM allocation on mobile device in sorting operation.

Table 5. 1: RAM Allocation in the Execution of Sorting Service Component of the Application on Local Mobile Device

Computational Length	Sample Mean of RAM Allocation (MB)	SD in RAM Allocation	%RSD in RAM Allocation	Confidence Interval
11000	10.148	0.013	0.1281	10.148(+/-).0061
12000	10.154	0.011	0.1083	10.154(+/-).0052
13000	10.156	0.005	0.0492	10.156(+/-).0024
14000	10.161	0.018	0.1771	10.161(+/-).0085
15000	10.167	0.004	0.0393	10.167(+/-).0019
16000	10.173	0.003	0.0295	10.173(+/-).0014
17000	10.177	0.001	0.0098	10.177(+/-).0005
18000	10.179	0.003	0.0295	10.179(+/-).0014
19000	10.185	0.004	0.0393	10.185(+/-).0019
20000	10.193	0.002	0.0196	10.193(+/-).0009
21000	10.197	0.001	0.0098	10.197(+/-).0005
22000	10.2	0.001	0.0098	10.2(+/-).0005
23000	10.204	0.001	0.0098	10.204(+/-).0005
24000	10.208	0.003	0.0294	10.208(+/-).0014
25000	10.21	0.001	0.0098	10.21(+/-)0.0098
26000	10.215	0.001	0.0098	10.215(+/-).0005
27000	10.218	0.001	0.0098	10.218(+/-).0005
28000	10.221	0.001	0.0098	10.221(+/-).0005
29000	10.224	0.001	0.0098	10.224(+/-).0005
30000	10.227	0.001	0.0098	10.227(+/-).0005
31000	10.231	0.001	0.0098	10.231(+/-).0005
32000	10.236	0.001	0.0098	10.236(+/-).00052
33000	10.238	0.001	0.0098	10.238(+/-).00005
34000	10.242	0.001	0.0098	10.242(+/-).0005
35000	10.246	0.001	0.0098	10.246(+/-).0005
36000	10.248	0.001	0.0098	10.248(+/-).0005
37000	10.254	0.001	0.0098	10.254(+/-).0005
38000	10.258	0.001	0.0097	10.258(+/-).0005
39000	10.261	0.002	0.0195	10.261(+/-).0009
40000	10.265	0.001	0.0097	10.265(+/-).0005

Table 5.2 shows the statistics of RAM allocation for matrix multiplication service component of the application on local mobile device. The computational length attribute shows the computational intensity of matrix multiplication operation in 30 different experiments. It shows the RAM allocation to matrix multiplication service with 30 different computational intensities (160*160-450*450). The sample mean attribute shows the point estimator for 30 different samples of the application execution with the identical computational intensity. The SD attribute shows the variation in the memory allocation to the application, whereas the %RSD attribute shows the percentage values of the variation in the sample space of 30 values in each experiment of evaluating RAM allocation on mobile device in matrix multiplication operation.

Table 5. 2: RAM Allocation in the Execution of the Matrix Multiplication Service Component of the Application on Local Mobile Device

Computational Length	Sample Mean of RAM Allocation (MB)	SD in RAM Allocation	%RSD in RAM Allocation	Confidence Interval
160*160	10.454	0.0029	0.0277	10.454(+/-).0014
170*170	10.4552	0.0026	0.0249	10.4552(+/-).0012
180*180	10.4967	0.0016	0.0152	10.4967(+/-).0008
190*190	10.5389	0.0024	0.0228	10.5389(+/-).011
200*200	10.5845	0.002	0.0189	10.5845(+/-).0009
210*210	10.6318	0.0009	0.0085	10.6318(+/-).0004
220*220	10.6828	0.0022	0.0206	10.6828(+/-).001
230*230	10.7346	0.0017	0.0158	10.7346(+/-).0008
240*240	10.7624	0.1406	1.3064	10.6605(+/-).1376
250*250	10.8317	0.0183	0.1689	10.8317(+/-).0086
260*260	10.9052	0.0023	0.0211	10.7986(+/-).1303
270*270	10.9605	0.007	0.0639	10.9605(+/-).0033
280*280	11.0115	0.0274	0.2488	11.0115(+/-).0129
290*290	11.1404	0.1214	1.0897	11.1404(+/-).0572
300*300	11.1617	0.0049	0.0439	11.4975(+/-).0886
310*310	11.2622	0.0447	0.3969	11.2622(+/-).0211
320*320	11.3826	0.1336	1.1737	11.3826(+/-).0629
330*330	11.756	0.1493	1.27	11.756(+/-).0703
340*340	11.78	0.178	1.511	11.78(+/-).0838

Computational Length	Sample Mean of RAM Allocation (MB)	SD in RAM Allocation	%RSD in RAM Allocation	Confidence Interval
350*350	11.7915	0.2047	1.736	11.7915(+/-).0964
360*360	11.8857	0.2342	1.9704	11.8857(+/-).1103
370*370	11.8857	0.1039	0.8742	11.8857(+/-).0498
380*380	12.3547	0.0037	0.0299	12.3547(+/-).0017
390*390	12.4562	0.1085	0.8711	12.4562(+/-).0511
400*400	12.5917	0.0057	0.0453	12.5917(+/-).0027
410*410	12.7304	0.0182	0.143	12.7304(+/-).0086
420*420	12.8516	0.0037	0.0288	12.8516(+/-).0017
430*430	12.9499	0.0374	0.2888	12.9499(+/-).0176
440*440	12.9726	0.0175	0.1349	12.656(+/-).1257
450*450	13.1003	0.1591	1.2145	13.1003(+/-).0749

Table 5.3 shows the statistics of RAM allocation for power compute service component of the application on local mobile device. The computational length attribute shows the computational length of power compute service component of the application. Memory allocation to power compute service is represented with 30 different computational intensities (2^100000-2^2000000000). The sample mean attribute shows the point estimator for 30 different samples of the application execution with the identical computational intensity. The SD attribute shows the variation in the memory allocation to power compute service, whereas the %RSD attribute shows the percentage values of the variation in the sample space of 30 values in each experiment of evaluating RAM allocation on mobile device in power computing operation.

Table 5. 3: RAM Allocation in the Execution of the Power Compute Service Component of the Application on Local

Computational Length	Sample Mean of RAM Allocation (MB)	SD in RAM Allocation	%RSD in RAM Allocation	Confidence Interval
2^100000-2^2000000000	10.11	0.0017	0.0168	10.11(+/-).00045

Table 5.4 shows the statistics of the Turnaround Time (TT) of the sorting service component of the application. The TT of the sorting service involves the execution time of completing the sorting operation and the time taken in saving the preferences file (data file) on the local mobile device. The table shows computational length for the evaluation of TT of the sorting operation in 30 different experiment, the sample mean of sample space of 30 values in each experiment, the variation in the values of the sample space, the percentage of difference in the values of the sample space of each experiment and the range of values for TT value of each experiment with 99% confidence for the sample space of 30 values.

Table 5. 4: Turnaround Time of Sorting Service on Mobile Application

Computational Length	**Sample Mean of TT (ms)**	**SD in TT**	%RSD in TT	**Confidence Interval**
11000	4876	706	14	4876(+/-)333
12000	5510	1168	21	5510(+/-)550
13000	6566	546	8	6566(+/-)257
14000	6989	697	10	6989(+/-)328
15000	7406	918	12	7406(+/-)432
16000	7450	910	12	7450(+/-)429
17000	10414	531	5	10414(+/-)250
18000	11457	693	6	11457(+/-)326
19000	11857	410	3	11857(+/-)193
20000	13221	316	2	13221(+/-)149
21000	13774	614	4	13774(+/-)289
22000	14410	641	4	14410(+/-)302
23000	15579	356	2	15579(+/-)168
24000	16059	532	3	16059(+/-)251
25000	16950	915	5	16950(+/-)431
26000	17764	412	2	17764(+/-)194
27000	18421	375	2	18421(+/-)177
28000	19176	472	2	19176(+/-)222
29000	20179	668	3	20179(+/-)315
30000	20987	501	2	20987(+/-)236
31000	21600	1467	7	21600(+/-)690
32000	22565	933	4	22565(+/-)439
33000	24701	455	2	24701(+/-)214

Computational Length	**Sample Mean of TT (ms)**	**SD in TT**	%RSD in TT	**Confidence Interval**
34000	25687	430	2	25687(+/-)203
35000	25825	670	3	25825(+/-)316
36000	26432	635	2	26432(+/-)299
37000	26910	931	3	26910(+/-)439
38000	28859	658	2	28859(+/-)310
39000	29968	1528	5	29968(+/-)720
40000	31207	1365	4	31207(+/-)643

Table 5.5 shows the statistics of the TT of the matrix multiplication component of the application. The TT of matrix multiplication service involves the execution time of completing the matrix multiplication operation and the time taken in saving the preferences file (data file) on the local mobile device. The table shows computational length for the evaluation of TT of the matrix multiplication operation in 30 different experiment, the sample mean of sample space of 30 values in each experiment, the variation in the values of the sample space, the percentage of difference in the values of the sample space of each experiment and the range of values for TT value of each experiment with 99% confidence for the sample space of 30 values.

Table 5. 5: Turnaround Time of Matrix Multiplication Service on Mobile Application

Computational Length	**Sample Mean of TT (ms)**	**SD in TT**	**%RSD in TT**	**Confidence Interval**
160*160	3653	191	5	3653(+/-)90
170*170	4276	656	15	4276(+/-)309
180*180	4781	647	14	4781(+/-)305
190*190	5030	1010	20	5030(+/-)476
200*200	6321	575	9	6321(+/-)271
210*210	7039	604	9	7039(+/-)285
220*220	7777	669	9	7777(+/-)315
230*230	8888	974	11	8888(+/-)459
240*240	10735	826	8	10735(+/-)389
250*250	13090	694	5	13090(+/-)327
260*260	13642	1182	9	13642(+/-)557

Computational Length	Sample Mean of TT (ms)	SD in TT	%RSD in TT	Confidence Interval
270*270	14471	1103	8	14471(+/-)520
280*280	16411	1221	7	16411(+/-)575
290*290	20524	341	2	20524(+/-)161
300*300	20706	1625	8	20706(+/-)765
310*310	21185	3849	18	21185(+/-)1813
320*320	27028	5813	22	27028(+/-)2738
330*330	28452	3611	13	28452(+/-)1701
340*340	39691	3562	9	39691(+/-)1678
350*350	38570	5495	14	38570(+/-)2588
360*360	40096	4208	10	40096(+/-)1982
370*370	44896	6246	14	44896(+/-)2942
380*380	48088	4351	9	48088(+/-)2050
390*390	55560	11604	21	55560(+/-)5466
400*400	57339	6236	11	57339(+/-)2937
410*410	62405	7461	12	62405(+/-)3514
420*420	63159	5580	9	63159(+/-)2628
430*430	74424	8255	11	74424(+/-)3888
440*440	78163	13626	17	78163(+/-)6418
450*450	99286	11260	11	99286(+/-)5304

Table 5.6 shows the statistics of the TT of the power compute component of the application. The TT involves the execution time of completing the power compute operation. The table shows computational length for the evaluation of TT of the power compute operation in 30 different experiment, the sample mean of sample space of 30 values in each experiment, the variation in the values of the sample space, the percentage of difference in the values of the sample space of each experiment and the range of values for TT value of each experiment with 99% confidence for the sample space of 30 values.

Table 5. 6: Turnaround Time of Power Compute Service on Mobile Application

Computational Length	Sample Mean of TT (ms)	SD in TT	%RSD in TT	Confidence Interval
2^1000000	51	10	19.6	51(+/-)5
2^2000000	80	9	11.3	80(+/-)4

Computational Length	**Sample Mean of TT (ms)**	**SD in TT**	**%RSD in TT**	**Confidence Interval**
2^3000000	110	12	10.9	110(+/-)6
2^4000000	140	11	7.9	140(+/-)6
2^5000000	176	28	15.9	176(+/-)13
2^6000000	206	16	7.8	206(+/-)8
2^7000000	233	19	8.2	233(+/-)9
2^8000000	269	25	9.3	269(+/-)12
2^9000000	293	21	7.2	293(+/-)10
2^10000000	341	30	8.8	341(+/-)14
2^20000000	373	38	10.2	373(+/-)18
2^30000000	920	43	4.7	920(+/-)20
2^40000000	1216	45	3.7	1216(+/-)21
2^50000000	1501	28	1.9	1501(+/-)13
2^60000000	1767	32	1.8	1767(+/-)15
2^70000000	2070	38	1.8	2070(+/-)18
2^80000000	2334	37	1.6	2334(+/-)17
2^90000000	2615	39	1.5	2615(+/-)18
2^100000000	2896	40	1.4	2806(+/-)19
2^200000000	6386	61	1	6386(+/-)29
2^300000000	8509	54	0.6	8509(+/-)25
2^400000000	11405	316	2.8	11405(+/-)149
2^500000000	14105	56	0.4	14105(+/-)26
2^600000000	16887	93	0.6	16887(+/-)44
2^700000000	19182	1149	6	19182(+/-)541
2^800000000	22480	160	0.7	22480(+/-)75
2^900000000	25580	1064	4.2	25580(+/-)501
2^100000000	28237	471	1.7	28237(+/-)222
2^1900000000	68365	7598	11.1	68365(+/-)3579
2^2000000000	69044	8807	12.8	69044(+/-)4148

Table 5.7 presents the Energy Consumption Cost (ECC) in processing sorting service component of the application on local mobile device. The attribute of computational length shows the computational intensity of sorting logic of the mobile application. ECC is analyzed for 30 different computational intensities of the sorting operation. The energy of mobile device is consumed in processing the application on mobile device and saving the data files of the application on mobile device.

Hence, the total energy consumption cost of sorting service component of the application is computed as the sum of ECC of processing the application on the local mobile device and the ECC of saving the preferences (data file) in the data folder of the sorting service application on the mobile device. The sample mean attribute shows the point estimator for the ECC of sorting operation in the sample space of 30 values for each experiment. Standard Deviation shows the variation in values of the sample space each experiment and %RSD indicates the percent Relative Standard Deviation in the ECC of sorting operation in each experiment. The attribute of confidence interval shows the interval estimate for ECC of sorting operation in each experiment with 99% confidence for the sample space of 30 values.

Table 5. 7: Energy Consumption Cost (ECC) of Sorting Service Operating on the Local Mobile Device

Computational Length	Sample Mean of ECC (J)	SD in ECC	%RSD in ECC	Confidence Interval
11000	16.2	2.8	17.3	16.2(+/-)1.3
12000	16.7	2.6	15.6	16.7(+/-)1.2
13000	18.1	4.1	22.7	18.1(+/-)1.9
14000	18.3	2.5	13.7	18.3(+/-)1.2
15000	18.6	2.5	13.4	18.6(+/-)1.2
16000	20.2	2.7	13.4	20.2(+/-)1.3
17000	20.8	2.8	13.5	20.8(+/-)1.3
18000	21.8	2.8	12.8	21.8(+/-)1.3
19000	22.9	6	26.2	22.9(+/-)2.8
20000	23.1	3.7	16	23.1(+/-)1.7
21000	25.2	5	19.8	25.2(+/-)2.4
22000	28	5.2	18.6	28(+/-)2.4
23000	28.1	4.5	16	28.1(+/-)2.1
24000	28.2	3.9	13.8	28.2(+/-)1.8
25000	30.4	3.3	10.9	30.4(+/-)1.6
26000	31.9	4.5	14.1	31.9(+/-)2.1
27000	33.8	3.3	9.8	33.8(+/-)1.6
28000	36.4	4.6	12.6	36.4(+/-)2.2
29000	38.3	3.3	8.6	38.3(+/-)1.6
30000	39.2	3.5	8.9	39.2(+/-)1.6

Computational Length	**Sample Mean of ECC (J)**	**SD in ECC**	**%RSD in ECC**	**Confidence Interval**
31000	40.9	3.5	8.6	40.9(+/-)1.6
32000	42.1	3.4	8.1	42.1(+/-)1.6
33000	44.3	3.1	7	44.3(+/-)1.5
34000	47.8	3.8	7.9	47.8(+/-)1.8
35000	48.4	2.9	6	48.4(+/-)1.4
36000	49.7	4.3	8.7	49.7(+/-)2
37000	51.4	2.8	5.4	51.4(+/-)1.3
38000	52.5	3.2	6.1	52.5(+/-)1.5
39000	53.2	2.4	4.5	53.2(+/-)1.1
40000	55.1	3.7	6.7	55.1(+/-)1.7

Table 5.8 presents the ECC in processing matrix multiplication component of the application on local mobile device. Energy consumption cost parameter is analyzed for 30 different computational intensities of the matrix multiplication operation. Matrix multiplication operation involves saving the results of the matrix multiplication operating in the preferences file on the mobile device. Hence, the total ECC of matrix multiplication service component of the application is computed as the energy consumption cost of processing the application on the local mobile device and the energy consumption cost of saving the preferences (data file) in the data folder of the sorting service application on the mobile device. The variation in the values of sample space for each experiment is represented with SD and the percentage difference in the sample space of each experiment is represented with %RSD. The sample mean of ECC is determined for the sample space of 30 values in each experiment and the interval estimate for each experiment is presented with 99% confidence interval for the sample space of 30 values in each experiment.

Table 5. 8: Energy Consumption Cost (ECC) of Matrix Multiplication Operation on Local Mobile Device

Computational Length	Sample Mean of ECC (J)	SD in ECC	%RSD in ECC	Confidence Interval
160*160	12.9	2.8	21.7	12.9(+/-)1.3
170*170	13.4	3	22.4	13.4(+/-)1.4
180*180	14.7	8.1	55.1	14.7(+/-)3.8
190*190	15.2	3.9	25.7	15.2(+/-)1.8
200*200	16.3	6.5	39.9	16.3(+/-)3.1
210*210	17.2	6	34.9	17.2(+/-)2.8
220*220	20	7.7	38.5	20(+/-)3.6
230*230	21.5	5.1	23.7	21.5(+/-)2.4
240*240	22	6.5	29.5	22(+/-)3.1
250*250	24.1	8.1	33.6	24.1(+/-)3.8
260*260	24.2	2.2	9.1	24.2(+/-)1
270*270	27.4	6.4	23.4	27.4(+/-)3
280*280	28.7	3.2	11.1	28.7(+/-)1.5
290*290	34.5	8.7	25.2	34.5(+/-)4.1
300*300	35.2	4	11.4	35.2(+/-)1.9
310*310	39.7	7	17.6	39.7(+/-)3.3
320*320	41.1	5.5	13.4	41.1(+/-)2.6
330*330	44.4	9.3	20.9	44.4(+/-)4.4
340*340	45.5	9.8	21.5	45.5(+/-)4.6
350*350	51.4	16.6	32.3	51.4(+/-)7.8
360*360	54.3	14.2	26.2	54.3(+/-)6.7
370*370	63.2	8.7	13.8	63.2(+/-)4.1
380*380	65.7	9.8	14.9	65.7(+/-)4.6
390*390	67	11.5	17.2	67(+/-)5.4
400*400	67.4	10.8	16	67.4(+/-)5.1
410*410	69.1	9.4	13.6	69.1(+/-)4.4
420*420	69.2	9.4	13.6	69.2(+/-)4.4
430*430	69.8	9.6	13.8	69.8(+/-)4.5
440*440	70	10.4	14.9	70(+/-)4.9
450*450	71.5	10.7	15	71.5(+/-)5

Table 5.9 presents the ECC in processing power compute component of the application on local mobile device. The variation in the ECC values of sample space for each experiment is represented with SD and the percentage difference in the sample space

of each experiment is represented with %RSD. The sample mean of ECC is determined for the sample space of 30 values in each experiment and the interval estimate for each experiment is presented with 99% confidence interval for the sample space of 30 values in each experiment.

Table 5. 9: Energy Consumption Cost of Power Compute Operation on Local Mobile Device

Computational Length	Sample Mean of ECC (J)	SD in ECC	%RSD in ECC	Confidence Interval
2^1000000	2.2	0.7	31.8	2.2(+/-)0.3
2^2000000	2.3	0.3	13	2.3(+/-)0.1
2^3000000	2.4	0.6	25	2.4(+/-)0.3
2^4000000	2.5	0.5	20	2.5(+/-)0.2
2^5000000	2.8	0.6	21.4	2.8(+/-)0.3
2^6000000	3.9	1	25.6	3.9(+/-)0.5
2^7000000	4.2	1	23.8	4.2(+/-)0.5
2^8000000	5.2	2.2	42.3	5.2(+/-)1
2^9000000	4.3	1.1	25.6	4.3(+/-)0.5
2^10000000	4.8	0.8	16.7	4.8(+/-)0.4
2^20000000	3.9	1	25.6	3.9(+/-)0.5
2^30000000	5.3	1.5	28.3	5.3(+/-)0.7
2^40000000	4.5	1.5	33.3	4.5(+/-)0.7
2^50000000	4.5	0.8	17.8	4.5(+/-)0.4
2^60000000	5.4	0.6	11.1	5.4(+/-)0.3
2^70000000	6.1	1.3	21.3	6.1(+/-)0.6
2^80000000	6.3	0.7	11.1	6.3(+/-)0.3
2^90000000	6.4	1.3	20.3	6.4(+/-)0.6
2^100000000	6.4	1.3	20.3	6.4(+/-)0.6
2^200000000	12.7	1.4	11	12.7(+/-)0.7
2^300000000	15.4	2.8	18.2	15.4(+/-)1.3
2^400000000	21.8	5.5	25.2	21.8(+/-)2.6
2^500000000	21.6	1.7	7.9	28.7(+/-)0.8
2^600000000	25.3	2.9	11.5	25.3(+/-)1.4
2^700000000	30.6	4	13.1	30.6(+/-)1.9
2^800000000	34.3	1.3	3.8	34.3(+/-)0.6
2^900000000	36.2	5.6	15.5	36.2(+/-)2.6
2^100000000	38.9	8.5	21.9	38.9(+/-)4
2^1900000000	61.7	11.9	19.3	61.7(+/-)5.6
2^2000000000	67	7.6	11.3	67(+/-)3.6

Table 5.10 summarizes the average CPU utilization on local mobile device for different components of the mobile application. The percentage of CPU utilization on the local mobile devices varies for different components of the mobile application. It depends on the length of computational logic in the component of mobile application. The sample mean of CPU utilization is determined for the sample space of 900 values for each component of the mobile application and the interval estimate for each experiment is presented with 99% confidence interval for the sample space of 900 values for each component of the mobile application. The variation in the ECC values of sample space for each experiment is represented with SD and the percentage difference in the sample space of each experiment is represented with %RSD.

Table 5. 10: Statistics of CPU Utilization on the Mobile Device in Local Application Processing

Computational Service	**Computational Length**	**% CPU Utilization**	**SD**	**% RSD**	**Confidence Interval**	**Average MIPS**
Sort	11000-40000	48.67	2.62	5.38	48.67(+/-)0.96	1163
Matrix Multiplication	160*160-4560*450	45.46	8.51	18.72	45.46(+/-)4.01	1086
Power Compute	2^1000000-2^2000000000	48.04	4.13	8.6	48.04(+/-)1.38	1148

5.4 Data Collected for Application Execution in Traditional Computational Offloading

In this scenario mobile application is executed in the distributed mobile cloud environment by offloading the components of the mobile application at runtime. The traditional runtime offloading technique is implemented for offloading the sort service and matrix multiplication service components of the mobile application. We evaluate the total TT of the application, total ECC and size of data transmission in offloading the service components of the mobile application at runtime.

Table 5.11 shows the total TT of sorting component of the application in traditional runtime computational offloading. The TT of sorting operation is the sum of the total timing cost runtime component offloading (equation 3.3) and execution time of the sorting operation on the remote server node. The sample mean of TT for sorting operation is determined for the sample space of 30 values in each experiment and the interval estimate for each experiment is presented with 99% confidence interval for the sample space of 30 values in each experiment. The variation in the TT values of sample space for each experiment is represented with SD and the percentage difference in the sample space of each experiment is represented with %RSD.

Table 5. 11: Turnaround Time of the Sorting Operation in Traditional Computational Offloading

Computational Length	Sample Mean of TT(ms)	SD in TT	%RSD in TT	Confidence Interval
11000	24331	3138	12.9	24331(+/-)1478
12000	28267	3728	13.2	28267(+/-)1756
13000	31609	2332	7.4	31609(+/-)1098
14000	35115	3007	8.6	35115(+/-)1416
15000	37010	4246	11.5	37010(+/-)2000
16000	38571	4098	10.6	38571(+/-)1930
17000	42244	3700	8.8	42244(+/-)1743
18000	47714	3434	7.2	47714(+/-)1618
19000	49481	1994	4	49481(+/-)939
20000	54599	4257	7.8	54599(+/-)2005
21000	58953	3002	5.1	58953(+/-)1414
22000	63141	3689	5.8	63141(+/-)1738
23000	69280	3983	5.7	69280(+/-)1876
24000	73368	2530	3.4	73368(+/-)1192
25000	76615	3473	4.5	76615(+/-)1636
26000	81668	3495	4.3	81668(+/-)1646
27000	87634	3691	4.2	87634(+/-)1739
28000	92439	3206	3.5	92439(+/-)1510
29000	97729	3047	3.1	97729(+/-)1435
30000	107042	4370	4.1	107042(+/-)2058
31000	119084	5179	4.3	119084(+/-)2440
32000	120380	4779	4	120380(+/-)2251

Computational Length	Sample Mean of TT(ms)	SD in TT	%RSD in TT	Confidence Interval
33000	124931	5086	4.1	124931(+/-)2396
34000	128864	3418	2.7	128864(+/-)1610
35000	135006	2938	2.2	135006(+/-)1384
36000	139564	3559	2.6	139564(+/-)1676
37000	148009	5461	3.7	148009(+/-)2572
38000	154216	5115	3.3	154216(+/-)2409
39000	157490	5862	3.7	157490(+/-)2761
40000	166457	5333	3.2	166457(+/-)2512

Table 5.12 shows the TT of matrix multiplication component of the application in traditional computational offloading. The TT of matrix multiplication operation is the sum of the total timing cost runtime component offloading (equation 3.3) and execution time of the matrix multiplication operation on the remote server node. The sample mean of TT for matrix multiplication operation is determined for the sample space of 30 values in each experiment and the interval estimate for each experiment is presented with 99% confidence interval for the sample space of 30 values in each experiment. The variation in the TT values of sample space for each experiment is represented with SD and the percentage difference in the sample space of each experiment is represented with %RSD.

Table 5. 12: Turnaround Time of the Matrix Multiplication Operation in Traditional Computational Offloading

Computational Length	Sample Mean of TT (ms)	SD in TT	%RSD in TT	Confidence Interval
160*160	16431	818	5	16431(+/-)385
170*170	18296	1837	10	18296(+/-)865
180*180	21132	2676	12.7	21132(+/-)1261
190*190	22170	2900	13.1	22170(+/-)1366
200*200	26061	2137	8.2	26061(+/-)1007
210*210	27927	1879	6.7	27927(+/-)885
220*220	29878	1918	6.4	29878(+/-)903
230*230	33920	2915	8.6	33920(+/-)1373
240*240	38052	2413	6.3	38052(+/-)1137
250*250	44310	2011	4.5	44310(+/-)947

Computational Length	Sample Mean of TT (ms)	SD in TT	%RSD in TT	Confidence Interval
260*260	46841	3423	7.3	46841(+/-)1612
270*270	50412	3498	6.9	50412(+/-)1648
280*280	55178	4054	7.3	55178(+/-)1910
290*290	64414	2116	3.3	64414(+/-)997
300*300	67346	4720	7	67346(+/-)2223
310*310	68692	8615	12.5	68692(+/-)4058
320*320	80922	12431	15.4	80922(+/-)5856
330*330	84709	8293	9.8	84709(+/-)3906
340*340	110506	8874	8	110506(+/-)4180
350*350	108677	11319	10.4	108677(+/-)5332
360*360	114062	9008	7.9	114062(+/-)4243
370*370	125149	13127	10.5	125149(+/-)6183
380*380	134100	10567	7.9	134100(+/-)4977
390*390	148747	24295	16.3	148747(+/-)11444
400*400	156489	13573	8.7	156489(+/-)6393
410*410	171252	19704	11.5	171252(+/-)9281
420*420	184056	18618	10.1	184056(+/-)8770
430*430	213507	29040	13.6	213507(+/-)13679
440*440	221092	33793	15.3	221092(+/-)15918
450*450	262697	37971	14.5	262697(+/-)17886

Table 5.13 shows the total ECC in offloaded processing of the sorting service component of the application with 30 different computational intensities. The total ECC of sorting operation in traditional computational offloading is the sum of energy consumption cost of runtime component offloading (equation 3.1) and energy consumption cost of performing sorting operation on the remote cloud server node. The sample mean of ECC for sorting operation is determined for the sample space of 30 values in each experiment and the interval estimate for each experiment is presented with 99% confidence interval for the sample space of 30 values in each experiment. The variation in the ECC values of sample space for each experiment is represented with SD and the percentage difference in the sample space of each experiment is represented with %RSD.

Table 5. 13: Energy Consumption Cost (ECC) of Sorting Operation in Traditional Computational Offloading

Computational Length	Sample Mean of ECC (J)	SD in ECC	%RTD in ECC	Confidence Interval
11000	49.7749	7.3245	14.7	49.7749(+/-)3.4501
12000	53.7829	6.8753	12.8	53.7829(+/-)3.2386
13000	56.4835	6.5346	11.6	56.4835(+/-)3.0781
14000	60.5911	7.0763	11.7	60.5911(+/-)3.332
15000	61.8917	6.4377	10.4	61.8917(+/-)3.0324
16000	64.5919	6.3711	9.9	64.5919(+/-)3.0011
17000	70.6915	6.5094	9.2	70.6915(+/-)3.0662
18000	73.7923	6.335	8.6	73.7923(+/-)2.984
19000	76.692	11.6244	15.2	76.692(+/-)5.4756
20000	75.5971	8.3274	11	75.5971(+/-)3.9226
21000	79.6989	8.3845	10.5	79.6989(+/-)3.9494
22000	82.0987	7.2209	8.8	82.0987(+/-)3.4013
23000	87.7002	7.0997	8.1	87.7002(+/-)3.3443
24000	90.3008	8.1183	9	90.3008(+/-)3.8241
25000	95.4014	8.514	8.9	95.4014(+/-)4.0104
26000	105.702	8.0472	7.6	105.702(+/-)3.7906
27000	109.1036	9.0645	8.3	109.1036(+/-)4.2698
28000	114.9034	8.1614	7.1	114.9034(+/-)3.8444
29000	120.505	8.1308	6.7	120.505(+/-)3.8299
30000	125.1055	10.8596	8.7	125.1055(+/-)5.1153
31000	131.607	6.916	5.3	131.607(+/-)3.2577
32000	136.3078	18.8652	13.8	136.3078(+/-)8.8863
33000	150.3086	10.5245	7	150.3086(+/-)4.9575
34000	159.4095	10.6692	6.7	159.4095(+/-)5.0256
35000	165.3109	10.7827	6.5	165.3109(+/-)5.0791
36000	173.8131	8.0774	4.6	173.8131(+/-)3.8048
37000	183.0151	9.1271	5	183.0151(+/-)4.2992
38000	186.8154	10.2068	5.5	186.8154(+/-)4.8078
39000	195.9168	9.5538	4.9	195.9168(+/-)4.5002
40000	201.4191	12.5839	6.2	201.4191(+/-)5.9275

Table 5.14 shows the ECC in offloaded processing of the matrix multiplication service component of the application. The total ECC of matrix multiplication operation in traditional computational offloading is the sum of energy consumption cost of runtime component offloading (equation 3.1) and energy consumption cost of performing matrix

multiplication operation on the remote cloud server node. The sample mean of ECC for matrix multiplication operation is determined for the sample space of 30 values in each experiment and the interval estimate for each experiment is presented with 99% confidence interval for the sample space of 30 values in each experiment. The variation in the ECC values of sample space for each experiment is represented with SD and the percentage difference in the sample space of each experiment is represented with %RSD.

Table 5. 14: Energy Consumption Cost (ECC) of Matrix Multiplication Operation in Traditional Runtime Offloading

Computational Length	Sample Mean of ECC(J)	SD in ECC	%RSD in ECC	Confidence Interval
160*160	39.9898	11.7609	29.4	39.9898(+/-)5.5399
170*170	40.9927	13.1147	32	40.9927(+/-)6.1776
180*180	43.998	16.5939	37.7	43.998(+/-)7.8164
190*190	44.5995	12.1789	27.3	44.5995(+/-)5.7368
200*200	44.4027	15.4921	34.9	44.4027(+/-)7.2974
210*210	46.5297	16.1287	34.7	46.5297(+/-)7.5973
220*220	49.3349	15.8277	32.1	49.3349(+/-)7.4555
230*230	52.1504	13.9581	26.8	52.1504(+/-)6.5748
240*240	52.8534	14.8811	28.2	52.8534(+/-)7.0096
250*250	58.0603	15.2977	26.3	58.0603(+/-)7.2059
260*260	59.1631	9.6401	16.3	59.1631(+/-)4.5409
270*270	64.5291	13.9521	21.6	64.5291(+/-)6.572
280*280	65.4867	10.2341	15.6	65.4867(+/-)4.8207
290*290	71.8974	13.3375	18.6	71.8974(+/-)6.2825
300*300	71.9067	10.3129	14.3	71.9067(+/-)4.8578
310*310	76.7187	13.2817	17.3	76.7187(+/-)6.2562
320*320	79.2983	11.0075	13.9	79.2983(+/-)5.185
330*330	82.7353	16.8737	20.4	82.7353(+/-)7.9482
340*340	85.8619	17.5749	20.5	85.8619(+/-)8.2785
350*350	93.1617	22.3729	24	93.1617(+/-)10.5386
360*360	95.7239	21.1127	22.1	95.7239(+/-)9.945
370*370	105.5589	14.7985	14	105.5589(+/-)6.9707
380*380	109.1943	16.1279	14.8	109.1943(+/-)7.5969
390*390	109.8383	16.0029	14.6	109.8383(+/-)7.538
400*400	114.1728	16.5943	14.5	114.1728(+/-)7.8166
410*410	115.1444	17.1541	14.9	115.1444(+/-)8.0803
420*420	119.0066	16.5657	13.9	119.0066(+/-)7.8031

Computational Length	Sample Mean of ECC(J)	SD in ECC	%RSD in ECC	Confidence Interval
430*430	121.4333	16.5499	13.6	121.4333(+/-)7.7957
440*440	126.8541	18.4795	14.6	126.8541(+/-)8.7046
450*450	131.6952	16.5959	12.6	131.6952(+/-)7.8174

Table 5.15 shows the size of data transmission over the wireless network medium in offloading sort service component of the application with 30 different computational intensities at runtime. Total data size for either instance of list size includes the data size of application binary file, the size of preferences file uploaded to the remote server node at runtime and the size of preferences file downloaded to the mobile devices for returning results. The length of sorting list attribute shows the length of one dimensional array. The attribute of preferences file size shows the size of data file size of the sorting component of the application. The attribute of total data size represent the amount of total data transmitted in each experiment of offloading sort service component of the application. The goodput attribute shows the number of bits delivered by the network to the remote cloud server node in the unit time. The goodput attribute is represented in the units of Kilo Bits per Second (Kbps). It indicates the application layer throughput of the data transmission irrespective of the transmission overhead of the underlying layers of the TCP/IP protocol stack.

Table 5. 15: The Size of Data Transmission over the Wireless Network Medium for Sorting Component in Traditional Computational Offloading

Length of Sort List	Preferences Size (KB)	Total Data Size (KB)	Goodput (Kbps)
11000	354	752.4	843.6
12000	388	820.4	841.1
13000	422	888.4	813.8
14000	453	950.4	826.5
15000	491	1026.4	823.3
16000	521	1086.4	868.3
17000	559	1162.4	869.1
18000	593	1230.4	810.3

Length of Sort List	Preferences Size (KB)	Total Data Size (KB)	Goodput (Kbps)
19000	627	1298.4	782
20000	658	1360.4	851.8
21000	688	1420.4	844.9
22000	718	1480.4	774.6
23000	764	1572.4	769.8
24000	794	1632.4	786.7
25000	825	1694.4	789.3
26000	855	1754.4	801.5
27000	901	1846.4	832
28000	935	1914.4	828.3
29000	969	1982.4	842.9
30000	999	2042.4	841.1
31000	1024	2092.4	860.5
32000	1054.72	2153.84	860.6
33000	1085.44	2215.28	856.2
34000	1116.16	2276.72	867.3
35000	1177.6	2399.6	908.6
36000	1177.6	2399.6	877.2
37000	1208.32	2461.04	875.1
38000	1239.04	2522.48	888.4
39000	1269.76	2583.92	903.1
40000	1300.48	2645.36	915.7

Table 5.16 shows the size of data transmission over the wireless network medium in offloading matrix multiplication component of the application with 30 different computational intensities at runtime. The length of matrices attribute shows the length of two dimensional arrays used in the matrix multiplication. The attribute of preferences file size shows the size of data file size of the matrix multiplication component of the application. The attribute of total data size represent the amount of total data transmitted with either size 2-D array in matrix multiplication. Total data size for either instance of list size includes the data size of application binary file, the size of preferences file uploaded to the remote server node at runtime and the size of preferences file downloaded to the mobile devices for returning results. The goodput attribute shows the number of bits delivered by

the network to the remote cloud server node in the unit time for matrix multiplication component.

Table 5. 16: The Size of Data Transmission over the Wireless Network Medium for Matrix Multiplication Component in Traditional Computational Offloading

Length of Matrices	Preferences Size (KB)	Data Transmission (KB)	Goodput(Kbps)
160*160	2846.72	5739.44	9238.5
170*170	3246.08	6538.16	9919.5
180*180	3665.92	7377.84	9172.1
190*190	4085.76	8217.52	9816.4
200*200	4536.32	9118.64	9460.4
210*210	4997.12	10040.24	10332.1
220*220	5488.64	11023.28	11084.2
230*230	6010.88	12067.76	10363
240*240	6543.36	13132.72	10858
250*250	7106.56	14259.12	10400.5
260*260	7690.24	15426.48	10427.7
270*270	8294.4	16634.8	10112.3
280*280	8919.04	17884.08	10525.5
290*290	9574.4	19194.8	10954.4
300*300	10240	20526	10402.1
310*310	10854.4	21754.8	10754.4
320*320	11673.6	23393.2	11166.9
330*330	12390.4	24826.8	11009.1
340*340	13209.6	26465.2	11351.8
350*350	14028.8	28103.6	11749.6
360*360	14848	29742	11867.1
370*370	15667.2	31380.4	11612.2
380*380	16588.8	33223.6	11111.1
390*390	17408	34862	11448.5
400*400	18329.6	36705.2	11786.7
410*410	19353.6	38753.2	10124.6
420*420	19660.8	39367.6	8016
430*430	21299.2	42644.4	7439.1
440*440	22323.2	44692.4	7863
450*450	23347.2	46740.4	8508.9

5.5 Data Collected for Application Execution by Employing DEAP Framework

In this scenario the prototype mobile application is tested by implementing DEAP framework. Application execution is evaluated in the Primary Operating Procedure (POP) and Secondary Operating Procedure (SOP) of the DEAP framework. The sorting service and matrix multiplication service components of the application are computational and data intensive. Therefore, sorting logic and matrix multiplication logic of the application is configured explicitly in the DEAP server. The POP of the DEAP client application is used to access the preconfigured services by employing SaaS cloud service provision model of computational clouds. However, the power compute service component of the application is offloaded at runtime in the SOP of the DEAP client application which utilizes IaaS service provisioning model of computational clouds. We evaluate the TT of the application, ECC in DEAP based processing of the application, size of data transmission in POP and SOP of the DEAP client application, RAM allocation and CPU utilization on local mobile device in leveraging the services of DEAP server.

Table 5.17 shows the TT of sorting operation in POP of the DEAP client application. Turnaround time of the sorting operation in POP of DEAP client application is the sum of application processing time on the remote server node and the time taken in saving resultant data on the local mobile device. The sample mean of TT for sorting operation is determined for the sample space of 30 values in each experiment and the interval estimate for each experiment is presented with 99% confidence interval for the sample space of 30 values in each experiment. The variation in the TT values of sample space for each experiment is represented with SD and the percentage difference in the sample space of each experiment is represented with %RSD.

Table 5. 17: Turnaround Time for Sorting Operation in the POP of the DEAP client Application

Computational Length	Sample Mean of TT (ms)	SD in TT	%RSD in TT	Confidence Interval
11000	2559	446	17.4	2559(+/-)210
12000	2902	715	24.6	3902(+/-)337
13000	3132	454	14.5	3132(+/-)214
14000	3345	308	9.2	3345(+/-)145
15000	3494	519	14.9	3494(+/-)244
16000	3757	694	18.5	3757(+/-)327
17000	3888	402	10.3	3888(+/-)189
18000	4379	316	7.2	4379(+/-)149
19000	4579	587	12.8	4579(+/-)277
20000	4864	238	4.9	4864(+/-)112
21000	5222	308	5.9	5222(+/-)145
22000	5461	227	4.2	5461(+/-)107
23000	5838	444	7.6	5838(+/-)209
24000	6171	248	4	6171(+/-)117
25000	6770	334	4.9	6770(+/-)157
26000	6844	388	5.7	6844(+/-)183
27000	7377	194	2.6	7377(+/-)91
28000	7885	167	2.1	7885(+/-)79
29000	8436	667	7.9	8436(+/-)314
30000	8525	861	10.1	8525(+/-)406
31000	8989	369	4.1	8989(+/-)174
32000	9378	3874	41.3	9378(+/-)1825
33000	10042	949	9.5	10042(+/-)447
34000	10328	630	6.1	10328(+/-)297
35000	10790	474	4.4	10790(+/-)223
36000	10914	1645	15.1	10914(+/-)775
37000	11704	551	4.7	11704(+/-)260
38000	12537	619	4.9	12537(+/-)292
39000	13182	1133	8.6	13182(+/-)534
40000	13416	598	4.5	13416(+/-)282

Table 5.18 shows the TT of matrix multiplication in POP of the DEAP client application. Turnaround time of the matrix multiplication operation in POP of DEAP client application is the sum of matrix multiplication time on the remote server node and the time taken in saving resultant data on the local mobile device. The sample mean of TT for

matrix multiplication operation is determined for the sample space of 30 values in each experiment and the interval estimate for each experiment is presented with 99% confidence interval for the sample space of 30 values in each experiment. The variation in the TT values of sample space for each experiment is represented with SD and the percentage difference in the sample space of each experiment is represented with %RSD.

Table 5. 18: Turnaround Time for Matrix Multiplication Operation in the POP of the DEAP Client Application

Computational Intensity	Sample Mean of TT (ms)	SD in TT	% RSD in TT	Confidence Interval of TT
160*160	4241	207	4.9	4241(+/-)98
170*170	4983	701	14.1	4983(+/-)330
180*180	5481	637	11.6	5481(+/-)300
190*190	5676	1010	17.8	5676(+/-)476
200*200	7038	579	8.2	7038(+/-)273
210*210	7795	655	8.4	7795(+/-)309
220*220	8560	598	7	8560(+/-)282
230*230	9661	1009	10.4	9661(+/-)475
240*240	11521	830	7.2	11521(+/-)391
250*250	13766	578	4.2	13766(+/-)272
260*260	14442	1192	8.3	14442(+/-)561
270*270	15174	1149	7.6	15174(+/-)541
280*280	17494	2238	12.8	17494(+/-)1054
290*290	21148	597	2.8	21148(+/-)281
300*300	21476	593	2.8	21476(+/-)279
310*310	22025	3868	17.6	22025(+/-)1822
320*320	27875	5953	21.4	27875(+/-)2804
330*330	28900	3563	12.3	28900(+/-)1678
340*340	38435	3987	10.4	38435(+/-)1878
350*350	39074	5117	13.1	39074(+/-)2410
360*360	40649	3562	8.8	40649(+/-)1678
370*370	45182	5611	12.4	45182(+/-)2643
380*380	48164	4250	8.8	48164(+/-)2002
390*390	55381	11854	21.4	55381(+/-)5584
400*400	57224	6180	10.8	57224(+/-)2911
410*410	61358	7383	12	61358(+/-)3478
420*420	62199	6288	10.1	62199(+/-)2962
430*430	73466	8248	11.2	73466(+/-)3885

Computational Intensity	Sample Mean of TT (ms)	SD in TT	% RSD in TT	Confidence Interval of TT
440*440	77068	13544	17.6	77068(+/-)6380
450*450	97887	11002	11.2	97887(+/-)5182

The ECC for accessing sorting service of DEAP server is the sum of energy consumed in accessing the sorting service and energy consumed in saving the resultant data on the local mobile device. The sorting operation service of the of the DEAP server application in the POP of the DEAP client application is evaluated for 30 different computational intensities, and data are collected for the sample space of 30 values for either computational intensity of the sorting operation.

Table 5.19 shows the ECC of sorting operation in POP of the DEAP client application. The sample mean of ECC for sorting operation in POP of DEAP framework is determined for the sample space of 30 values in each experiment and the interval estimate for each experiment is presented with 99% confidence interval for the sample space of 30 values in each experiment. The variation in the ECC values of sample space for each experiment is represented with SD and the percentage difference in the sample space of each experiment is represented with %RSD.

Table 5. 19: Energy Consumption Cost (ECC) for Sorting Operation in the POP of the DEAP Framework

Computational Length	Sample Mean of ECC (J)	SD in ECC	%RSD in ECC	Confidence Interval in ECC
11000	7.4	1.3	17.6	7.4(+/-).6
12000	8.3	2	24.1	8.3(+/-)1.8
13000	9	1.8	20	9(+/-)1.6
14000	9.3	2.1	22.6	9.3(+/-).9
15000	9.3	2	21.5	9.3(+/-)1.8
16000	9.5	1.6	16.8	9.5(+/-).7
17000	10.7	2	18.7	10.7(+/-).9
18000	10.8	1.6	14.8	10.8(+/-).7

Computational Length	**Sample Mean of ECC (J)**	**SD in ECC**	**%RSD in ECC**	**Confidence Interval in ECC**
19000	11	4.1	37.3	11(+/-)1.9
20000	11.2	3.7	33	11.2(+/-)1.7
21000	12.3	2.3	18.7	12.3(+/-)1.1
22000	12.5	2.7	21.6	12.5(+/-)1.3
23000	12.7	2.4	18.9	12.7(+/-)1.1
24000	12.7	1.8	14.2	12.7(+/-).8
25000	14.1	3.2	22.7	14.1(+/-)1.5
26000	14.6	2.8	19.2	14.6(+/-)1.3
27000	14.8	2.8	18.9	14.8(+/-)1.6
28000	14.8	2.1	14.2	14.8(+/-)1
29000	14.9	3.1	20.8	14.9(+/-)1.4
30000	15.3	2.2	14.4	15.3(+/-)1
31000	15.3	1.8	11.8	15.3(+/-).8
32000	15.4	2.5	16.2	15.4(+/-)1.2
33000	17	3.6	21.2	17(+/-)1.7
34000	17	2.6	15.3	17.2(+/-)1.2
35000	17.3	3.7	21.4	17.3
36000	17.5	2.8	16	17.5(+/-)1.3
37000	17.6	2.1	11.9	17.6(+/-)1
38000	18.5	3.4	18.4	18.5(+/-)1.6
39000	21.8	3	13.8	21.8(+/-)1.4
40000	23	5.6	24.3	23(+/-)2.6

The ECC of matrix multiplication operation in the POP of DEAP client application is the sum of energy consumed in accessing the matrix multiplication operation service of the DEAP server application and energy consumed in saving the resultant preferences (data file) on the local mobile device. Table 5.20 shows the ECC of matrix multiplication operation in POP of the DEAP client application. The sample mean of ECC for matrix multiplication operation in POP of DEAP framework is determined for the sample space of 30 values in each experiment and the interval estimate for each experiment is presented with 99% confidence interval for the sample space of 30 values in each experiment. The variation in the ECC values of sample space for each experiment is represented with SD and the percentage difference in the sample space of each experiment is represented with %RSD.

Table 5. 20: Energy Consumption Cost (ECC) for Matrix Multiplication Operation in the POP of the DEAP Client Application

Computational Length	Sample Mean of ECC	SD in ECC	%RSD in ECC	Confidence Interval
160*160	10.8	3.8	35.2	10.8(+/-)1.8
170*170	11.2	3.5	31.3	11.2(+/-)1.6
180*180	11.9	5.2	43.7	11.9(+/-)2.4
190*190	12	3.4	28.3	12(+/-)1.6
200*200	12.8	7	54.7	12.8(+/-)3.3
210*210	13	4.7	36.2	13(+/-)2.2
220*220	14.5	5.3	36.6	14.5(+/-)2.5
230*230	15.5	4.2	27.1	15.5(+/-)2
240*240	16.2	5.2	32.1	16.2(+/-)2.4
250*250	17.3	5.5	31.8	17.3(+/-)2.6
260*260	18.5	5.2	28.1	18.5(+/-)2.4
270*270	19.3	6.8	35.2	19.3(+/-)3.2
280*280	20.3	5.6	27.6	20.3(+/-)2.6
290*290	23	6.9	30	23(+/-)3.3
300*300	24.2	8.2	33.9	24.2(+/-)3.9
310*310	25.6	10	39.1	25.6(+/-)4.7
320*320	29.3	8.5	29	29.3(+/-)4
330*330	31.5	12.3	39	31.5(+/-)5.8
340*340	32.6	12	36.8	32.6(+/-)5.7
350*350	35.7	14.9	41.7	35.7(+/-)7
360*360	37	13.6	36.8	37(+/-)6.4
370*370	41.6	7.6	18.3	41.6(+/-)3.6
380*380	45	11	24.4	45(+/-)5.2
390*390	47.3	13.5	28.5	47.3(+/-)6.4
400*400	49.3	19.6	39.8	49.3(+/-)9.2
410*410	53.7	11.4	21.2	53.7(+/-)5.4
420*420	56.8	17.4	30.6	56.8(+/-)8.2
430*430	57.1	10.2	17.9	57.1(+/-)4.8
440*440	61.6	12.3	20	61.6(+/-)5.8
450*450	65.3	10.8	16.5	65.3(+/-)5.1

The power compute service component of the application is offloaded at runtime by employing the SOP of the DEAP framework. Table 5.21 shows the time taken in of offloading power compute service in the SOP of DEAP client application. The sample mean of TT for power compute operation is determined for the sample space of 30 values and the interval estimate for each experiment is presented with 99% confidence interval for the sample space of 30 values. The variation in the TT values of sample space for each experiment is represented with SD and the percentage difference in the sample space of each experiment is represented with %RSD.

Table 5. 21: Time Taken in Offloading Power Compute in the SOP of DEAP client Application

	Sample Mean of Time (ms)	**SD in Time**	**%RSD**	**Confidence Interval**
Offloading Time	52	9	17.3	52(+/-)4
Download Time to Remote Virtual Device	212	39	18.4	212(+/-)18
Reconfiguration Time on the on Remote virtual Device	6349	663	10.4	6349(+/-)312

Table 5.22 shows the total time taken in the execution of power compute service component of the application in the SOP of DEAP client application. The sample mean of TT for power compute operation is determined for the sample space of 30 values in each experiment and the interval estimate for each experiment is presented with 99% confidence interval for the sample space of 30 values in each experiment. The variation in the TT values of sample space for each experiment is represented with SD and the percentage difference in the sample space of each experiment is represented with %RSD.

Table 5. 22: Turnaround Time of Offloading Power Compute in the SOP of DEAP client Application

Computational Length	**Sample Mean for TT**	**SD in TT**	**%RSD in TT**	**Confidence Interval**
2^1000000	7175	721.1	10.1	7175(+/-)340
2^2000000	7587	717	9.5	7587(+/-)338
2^3000000	7751	726.5	9.4	7751(+/-)342
2^4000000	7779	719.7	9.3	7779(+/-)339
2^5000000	7965	699	8.8	7965(+/-)329
2^6000000	8071	692.7	8.6	8071(+/-)326
2^7000000	8183	690.1	8.4	8183(+/-)325
2^8000000	8283	690.5	8.3	8283(+/-)325
2^9000000	8471	684.2	8.1	8471(+/-)322
2^10000000	8658	724.3	8.4	8658(+/-)341
2^20000000	9877	674.9	6.8	9877(+/-)318
2^30000000	12029	1398.2	11.6	12029(+/-)659
2^40000000	12999	719.5	5.5	12999(+/-)339
2^50000000	13444	1212.1	9	13444(+/-)571
2^60000000	15356	804.1	5.2	15356(+/-)379
2^70000000	17067	918.7	5.4	17067(+/-)433
2^80000000	18092	831.5	4.6	18092(+/-)392
2^90000000	19188	1115	5.8	19188(+/-)525
2^100000000	20389	1045.5	5.1	20389(+/-)492
2^200000000	32645	1367.5	4.2	32645(+/-)644
2^300000000	46475	876.3	1.9	46475(+/-)413
2^400000000	61280	2107.4	3.4	61280(+/-)913
2^500000000	75534	2123.2	2.8	75534(+/-)1000
2^600000000	84686	1944.1	2.3	84686(+/-)916
2^700000000	112309	5497.9	4.9	112309(+/-)2590
2^800000000	126616	2987.1	2.4	126616(+/-)1407
2^900000000	135190	2297.8	1.7	135190(+/-)1082
2^100000000	138851	1379.6	1	138851(+/-)650
2^1900000000	139471	3875.7	2.8	139471(+/-)1826
2^2000000000	265724	5274.6	2	265724(+/-)2485

Table 5.23 shows the total ECC in the execution of power compute service component of the application in the SOP of DEAP client application. The sample mean of ECC for matrix multiplication operation is determined for the sample space of 30 values in each experiment and the interval estimate for each experiment is presented with 99%

confidence interval for the sample space of 30 values in each experiment. The variation in the ECC values of sample space for each experiment is represented with SD and the percentage difference in the sample space of each experiment is represented with %RSD.

Table 5. 23: Total Energy Consumption Cost in Offloading of Power Compute Service in the SOP of DEAP Client Application

Compute Size	Sample Mean of ECC(J)	SD in ECC	%RSD	Confidence Interval
2^1000000	5.4	1.4	25.9	5.4(+/-).7
2^2000000	6.5	2.6	40	6.5(+/.-)1.2
2^3000000	6.8	1	14.7	6.8(+/-)0.5
2^4000000	7.9	3.2	40.5	7.9(+/-)1.5
2^5000000	8.1	2.3	28.4	8.1(+/-)1.1
2^6000000	8.6	2.3	26.7	8.6(+/-)1.1
2^7000000	9.5	3.3	34.7	9.5(+/-)1.6
2^8000000	9.6	1.6	16.7	9.6(+/-)0.8
2^9000000	10	1.4	14	10(+/-)0.7
2^10000000	10.5	2	19	10.5(+/-)0.9
2^20000000	13.2	2.3	17.4	13.2(+/-)1.1
2^30000000	15.1	1.9	12.6	15.1(+/-)0.9
2^40000000	17.1	1.9	11.1	17.1(+/-)0.9
2^50000000	19.8	1.3	6.6	19.8(+/-)0.6
2^60000000	23.2	2.8	12.1	23.2(+/-)1.3
2^70000000	25.1	1.9	7.6	25.1(+/-)0.9
2^80000000	26.8	2.6	9.7	26.8(+/-)1.2
2^90000000	30.2	1.1	3.6	30.2(+/-)0.5
2^100000000	31.3	2.1	6.7	31.3(+/-)1
2^200000000	56.1	2.3	4.1	56.1(+/-)1.1
2^300000000	72.7	4.2	5.8	72.7(+/-)2
2^400000000	79.6	4.4	5.5	79.6(+/-)2.1
2^500000000	88.2	2.7	3.1	88.2(+/-)1.3
2^600000000	106.4	1.6	1.5	106.4(+/-)0.8
2^700000000	126.4	4.9	3.9	126.4(+/-)2.3
2^800000000	143.3	5.9	4.1	143.3(+/-)2.8
2^900000000	164.9	3.4	2.1	164.9(+/-)1.6
2^100000000	181.1	6.5	3.6	181.1(+/-)3.1
2^1900000000	343.6	5.9	1.7	343.6(+/-)1.7
2^2000000000	351	15.7	4.5	351(+/-)4.5

Table 5.24 represents the statistics of RAM allocation on local mobile device for accessing the sorting service of DEAP server application. The sample mean of RAM allocation for matrix multiplication operation is determined for the sample space of 30 values in each experiment and the interval estimate for each experiment is presented with 99% confidence interval for the sample space of 30 values in each experiment. The variation in the RAM allocation values of sample space for each experiment is represented with SD and the percentage difference in the sample space of each experiment is represented with %RSD.

Table 5. 24: RAM Allocation on Local Mobile Device in Accessing Sorting Service of DEAP server Application

Computational length	**Sample Mean for RAM Allocation (MB)**	**SD in RAM Allocation**	**%RSD in RAM Allocation**	**Confidence Interval**
11000	1.165	0.06	5.2	1.165(+/-)0.028
12000	1.232	0.027	2.2	1.232(+/-)0.013
13000	1.248	0.011	0.9	1.248(+/-)0.005
14000	1.442	0.139	9.6	1.442(+/-)0.065
15000	1.559	0.021	1.3	1.559(+/-)0.01
16000	1.566	0.019	1.2	1.566(+/-)0.009
17000	1.654	0.111	6.7	1.654(+/-)0.052
18000	1.925	0.024	1.2	1.925(+/-)0.011
19000	1.938	0.006	0.3	1.938(+/-)0.003
20000	1.938	0.002	0.1	1.938(+/-)0.001
21000	2.324	0.002	0.1	2.324(+/-)0.001
22000	2.355	0.008	0.3	2.355(+/-)0.004
23000	2.356	0.085	3.7	2.314(+/-)0.04
24000	2.376	0.055	2.3	2.376(+/-)0.026
25000	2.382	0.043	1.8	2.382(+/-)0.02
26000	2.426	1.871	77.1	2.426(+/-)0.881
27000	2.854	0.077	2.7	2.854(+/-)0.036
28000	2.883	0.037	1.3	2.883(+/-)0.017
29000	2.891	0.091	3.1	2.891(+/-)0.043
30000	2.898	0.008	0.3	2.898(+/-)0.004
31000	2.922	0.026	0.9	2.922(+/-)0.012
32000	3.186	0.1	3.1	3.186(+/-)0.047

Computational length	Sample Mean for RAM Allocation (MB)	SD in RAM Allocation	%RSD in RAM Allocation	Confidence Interval
33000	3.518	0.001	0	3.518(+/-)0.0005
34000	3.531	0.01	0.3	3.531(+/-)0.005
35000	3.553	0.232	6.5	3.553(+/-)0.109
36000	3.559	0.176	4.9	3.559(+/-)0.083
37000	3.664	0.393	10.7	3.664(+/-)0.185
38000	3.68	0.109	3	3.68(+/-)0.051
39000	3.743	0.146	3.9	3.743(+/-)0.069
40000	4.344	0.376	8.7	4.344(+/-)0.177

Table 5.25 represents the statistics of RAM allocation on local mobile device for accessing the matrix multiplication service of the application on DEAP server application. The computational length of the matrix service varies in 30 different experiments (160*160-450*450), whereas the variation in the values of sample space in either computational intensity is shown with SD and %RSD. The confidence interval attribute shows the interval estimate of the sample mean with 99% confidence.

Table 5. 25: RAM Allocation on Local Mobile Device in Accessing Matrix Multiplication Service of DEAP Server Application

Computational Length	Sample Mean for RAM Allocation (MB)	SD in RAM Allocation	%RSD in RAM Allocation	Confidence Interval
160*160	1.695	0.16	9.4	1.695(+/-).08
170*170	1.865	0.068	3.6	1.865(+/-)0.03
180*180	2.094	0.068	3.2	2.094(+/-)0.03
190*190	2.36	0.198	8.4	2.36(+/-)0.09
200*200	2.438	0.071	2.9	2.438(+/-)0.03
210*210	2.887	0.13	4.5	2.887(+/-)0.06
220*220	3.119	0.162	5.2	3.119(+/-)0.08
230*230	3.524	0.091	2.6	3.524(+/-)0.04
240*240	3.576	0.12	3.4	3.576(+/-)0.06
250*250	3.752	0.272	7.2	3.752(+/-)0.13
260*260	4.193	0.13	3.1	4.193(+/-)0.06

Computational Length	Sample Mean for RAM Allocation (MB)	SD in RAM Allocation	%RSD in RAM Allocation	Confidence Interval
270*270	4.448	0.266	6	4.448(+/-)0.13
280*280	4.842	0.238	4.9	4.842(+/-)0.13
290*290	5.616	0.308	5.5	5.616(+/-)0.15
300*300	5.888	0.076	1.3	5.888(+/-)0.04
310*310	6.335	0.136	2.1	6.335(+/-)0.06
320*320	6.498	0.279	4.3	6.498(+/-)0.13
330*330	7.171	0.211	2.9	7.171(+/-)0.1
340*340	7.344	0.226	3.1	7.344(+/-)0.11
350*350	7.696	0.279	3.6	7.696(+/-)0.13
360*360	8.311	0.351	4.2	8.311(+/-)0.17
370*370	8.319	0.326	3.9	8.319(+/-)0.15
380*380	9.141	0.197	2.2	9.141(+/-)0.09
390*390	9.754	0.17	1.7	9.754(+/-)0.08
400*400	9.88	0.082	0.8	9.88(+/-)0.04
410*410	11.476	0.354	3.1	11.476(+/-)0.17
420*420	11.912	0.35	2.9	11.912(+/-)0.16
430*430	12.263	0.237	1.9	12.263(+/-)0.11
440*440	12.549	0.209	1.7	12.549(+/-)0.1
450*450	13.056	0.653	5	13.056(+/-)0.31

The percentage of CPU utilization on the local mobile devices in DEAP based processing varies for different components of the mobile application. In the POP of DEAP framework, the processing logic of the service components is executed on the DEAP server application. However, a certain amount of CPU is still utilized for accessing the services of DEAP server application on the remote server node. Table 5.26 summarizes the average CPU utilization on local mobile device for accessing the services of DEAP server application.

Table 5. 26: Statistics of CPU Utilization on the Mobile Device in DEAP Based Application Processing

Computational Service	**Computational Length**	**% CPU Utilization**	**SD in CPU Utilization**	**%RSD in CPU Utilization**	**Confidence Interval**	**Average MIPS**
Sort	11000-40000	25.5	14.00375	54.9	25.5(+/-)5.64	609.2
Matrix Multiplication	160*160-4560*450	35.4	17.8	50.3	35.4(+/-)8.38	845.7
Power Compute	2^1000000-2^2000000000	3	0.8	26.7	3(+/-)0.38	71. 7

Table 5.27 shows the size of data transmission over the wireless network medium in accessing the sorting service and matrix multiplication service on DEAP server application. The attribute of data size represent the amount of data transmitted with either list size of the sort array and matrix length.

Table 5. 27: Data Transmission in the POP of DEAP client Application

Length of Sorting List	**Data Transmission (KB) (Sorting Service)**	**Length of Matrices**	**Data Transmission (KB) (Matrix Service)**
11000	183	160*160	463
12000	200	170*170	528
13000	218	180*180	595
14000	235	190*190	664
15000	253	200*200	639
16000	270	210*210	705
17000	288	220*220	774
18000	306	230*230	847
19000	323	240*240	923
20000	341	250*250	1002
21000	358	260*260	1084
22000	376	270*270	1169
23000	393	280*280	1258
24000	411	290*290	1350
25000	429	300*300	1445
26000	446	310*310	1543

Length of Sorting List	Data Transmission (KB) (Sorting Service)	Length of Matrices	Data Transmission (KB) (Matrix Service)
27000	464	320*320	1646
28000	481	330*330	1754
29000	499	340*340	1865
30000	516	350*350	1979
31000	534	360*360	21629
32000	552	370*370	2219
33000	568	380*380	2343
34000	587	390*390	2471
35000	604	400*400	2602
36000	622	410*410	2737
37000	639	420*420	2874
38000	657	430*430	3015
39000	675	440*440	3160
40000	692	450*450	3308

5.6 Comparison of Experimental Results

This section presents the comparison of resources utilization in the application processing on the local device and remote server node. The comparison of experimental results is presented for Android virtual device and real time experimentation. Table 5.28 shows the TT of the sorting service on the local virtual device and execution time in the performing sorting operation on the remote server using the POP of DEAP client application. Similarly, the table compares the ECC of sort service execution on local virtual device and ECC of DEAP client application for accessing sorting service on DEAP server application.

Table 5. 28: Comparison of Sorting Service Execution on Local Android Virtual Device and POP of DEAP client Application

Computational Length	TT (ms) on Local AVD	TT (ms) in DEAP	ECC (J) on Local AVD	ECC (J) in DEAP
11000	13400	3246	20.1	13.3
12000	16217	3475	22.9	13.6

Computational Length	**TT (ms) on Local AVD**	**TT (ms) in DEAP**	**ECC (J) on Local AVD**	**ECC (J) in DEAP**
13000	18423	3782	25.4	15
14000	20918	3954	29.1	15.4
15000	22021	4541	30	16.2
16000	23425	4816	32.7	16.2
17000	26109	4928	38.2	16.3
18000	29743	5476	40.9	16.4
19000	30777	5827	43.6	16.5
20000	34911	6215	42.3	16.7
21000	38949	6237	46.4	17.8
22000	41064	6839	48.4	17.4
23000	45905	7571	53.4	18.7
24000	49591	7659	55.6	18.7
25000	51808	8081	60.5	18.9
26000	56479	8221	70.2	20
27000	61963	8560	73.4	20.2
28000	65919	8717	78.8	20.8
29000	70729	9435	83.8	20.9
30000	79087	10283	87.4	21.5
31000	90472	10564	93.1	21.3
32000	91096	10883	96.6	22.2
33000	94886	10899	108.8	22.9
34000	98462	10913	116.9	23.6
35000	104213	11308	122.6	24
36000	107928	12179	130.7	24.9
37000	115239	12689	138.9	26.9
38000	121113	13216	142.5	28.2
39000	123935	13435	151.4	31.2
40000	132661	14093	156.1	32.5

Table 5.29 compares execution time and energy consumption in the execution of matrix multiplication operation on local virtual device instance and DEAP based service execution by using Android virtual device. The TT of the matrix multiplication operation is compared for local execution on AVD and remote execution on remote server node. Similarly, the table compares the Energy Consumption Cost (ECC) of matrix multiplication

service execution on local virtual device and ECC of DEAP client application for accessing matrix multiplication service on DEAP server application.

Table 5. 29: Comparison of Matrix Multiplication Service Execution on Local Android Virtual Device and POP of DEAP client Application

Computational Length	**TT (ms) on Local AVD**	**TT (ms) in DEAP**	**ECC (J) on Local AVD**	**ECC (J) in DEAP**
160*160	4430	4488	11.2	11.8
170*170	5146	4587	12	13.5
180*180	5762	5439	12.8	14.2
190*190	6034	5668	13	14.3
200*200	7500	6120	13.6	14.3
210*210	8264	6838	14.7	15.1
220*220	9003	7485	17.5	15.7
230*230	10243	8358	18.7	15.9
240*240	12010	9069	19.4	16.5
250*250	14429	9901	21.8	16.7
260*260	15237	10101	22.3	17.4
270*270	16429	10493	25.6	18.3
280*280	18657	10666	26.9	18.8
290*290	22963	11869	32.9	19.9
300*300	23665	12228	33.9	20
310*310	23973	13705	37.7	20
320*320	29914	13724	39.2	22.2
330*330	31121	14664	43.2	22.4
340*340	42864	15405	45	22.4
350*350	41778	15632	50.8	23.8
360*360	43578	15938	53.9	25.8
370*370	48387	18822	63.5	27.2
380*380	51952	19010	66.8	27
390*390	58945	18998	67.2	28.2
400*400	64100	20828	71.9	28.2
410*410	68265	21287	72.5	29.1
420*420	70675	22200	74.7	30
430*430	82111	24789	75.8	35.2
440*440	86244	23674	77.1	35.7
450*450	108202	24994	79	43.7

Table 5.30 compares execution time and energy consumption in the execution of power compute service on local AVD and remote DEAP server application by using Android virtual device. It compares the TT of the power compute service on the local virtual device and execution time in performing power compute operation on the remote server using the POP of DEAP client application. Similarly, the table compares the ECC of power compute service execution on local virtual device and ECC of DEAP client application for accessing power compute service on DEAP server application.

Table 5. 30: Comparison of Power Compute Service Execution on Local Android Virtual Device and POP of DEAP client Application

Computational Length	TT (ms) on Local AVD	TT (ms) in DEAP	ECC (J) on Local AVD	ECC (J) in DEAP
2^1000000	562	173	3.1	2.9
2^2000000	974	182	4.1	3.6
2^3000000	1138	186	4.3	3.7
2^4000000	1166	191	5.1	3.8
2^5000000	1352	206	5.2	4.1
2^6000000	1458	207	5.6	4.5
2^7000000	1570	230	6.4	4.6
2^8000000	1670	234	6.5	5.9
2^9000000	1858	260	6.8	6.1
2^10000000	2045	263	7.2	6.6
2^20000000	3264	301	9.7	8.5
2^30000000	5416	346	11.6	8.1
2^40000000	6386	394	13.6	8.2
2^50000000	6831	475	16.3	8.2
2^60000000	8743	521	19.7	8.3
2^70000000	10454	524	21.5	8.4
2^80000000	11479	549	23.1	8.6
2^90000000	12575	569	26.3	9.4
2^100000000	13776	608	27.3	10.6
2^200000000	26032	1108	51.5	10.6
2^300000000	39862	1171	67.8	11.3
2^400000000	54667	1445	74.6	11.3
2^500000000	68921	2501	83	11.4
2^600000000	78073	2948	100.5	13.4

2^700000000	105696	3312	119.5	13.5
2^800000000	120003	4369	135.5	13.9
2^900000000	128577	4846	156.8	14.1
2^100000000	132238	5134	172.7	14.2
2^1900000000	132858	9146	333.3	14.3
2^2000000000	259111	9606	339.9	14.5

Table 5.31 summarizes the comparison of the TT of sorting operating of the application in different scenarios of the real distributed mobile cloud computing environment. The contemporary approaches for application offloading implement runtime application profiling technique for outsourcing the computational load at runtime. Therefore, Local execution time is evaluated from two perspective.1) The execution time of the application is evaluated without involving the runtime profiling mechanism. 2) In order to evaluate the impact of runtime profiling, the execution time of the application is evaluated by including the runtime profiling mechanism. The TT in DEAP attribute shows total time taken in the executing the sort service on DEAP server application and returning the results to the local mobile device. The TT of the application in traditional offloading is represented from two perspectives. (a) The TT in traditional offloading without profiling shows the turnaround time of the sorting service operation by offloading the component without using runtime profiling on the local mobile. (b) Whereas, the TT in traditional offloading with profiling attribute represents the turnaround time of the application by including the profiling mechanism on local mobile device.

Table 5. 31: Comparison of Turnaround Time (ms) of Sorting Operation in Local and Remote Execution

Computational Length	**TT on Local SMD (Without Profiling)**	**TT on Local SMD (Including Profiling)**	**TT in POP of DEAP**	**TT in Traditional Offloading (without Profiling)**	**TT in Traditional Offloading (Including Profiling)**
11000	4876	20756	2559	24331	40211

Computational Length	TT on Local SMD (Without Profiling)	TT on Local SMD (Including Profiling)	TT in POP of DEAP	TT in Traditional Offloading (without Profiling)	TT in Traditional Offloading (Including Profiling)
12000	5510	25160	2902	28267	47917
13000	6566	29298	3132	31609	54341
14000	6989	32718	3345	35115	60844
15000	7406	38820	3494	37010	68424
16000	7450	47778	3757	38571	78899
17000	10414	58773	3888	42244	90603
18000	11457	68132	4379	47714	104389
19000	11857	82182	4579	49481	119806
20000	13221	92106	4864	54599	133484
21000	13774	98885	5222	58953	144064
22000	14410	107419	5461	63141	156150
23000	15579	108969	5838	69280	162670
24000	16059	118121	6171	73368	175430
25000	16950	126903	6770	76615	186568
26000	17764	134495	6844	81668	198399
27000	18421	142275	7377	87634	211488
28000	19176	150483	7885	92439	223746
29000	20179	159581	8436	97729	237131
30000	20987	169385	8525	107042	255440
31000	21600	179592	8989	119084	277076
32000	22565	190058	9378	120380	287873
33000	24701	197576	10042	124931	297806
34000	25687	205867	10328	128864	309044
35000	25825	229236	10790	135006	338417
36000	26432	230547	10914	139564	343679
37000	26910	246572	11704	148009	367671
38000	28859	252496	12537	154216	377853
39000	29968	264108	13182	157490	391630
40000	31207	275148	13416	166457	410398

Table 5.32 compares the TT of the matrix multiplication operation of the application in different scenarios. The execution time on the local device represents the TT of the matrix multiplication operation and saving the preferences file on the local mobile device. The TT in DEAP attribute shows total time taken in the executing the matrix

multiplication operation on DEAP server application and returning the results to the local mobile device. The TT of the application in traditional offloading is represented from two perspectives. The TT in traditional offloading without profiling shows the turnaround time of the matrix multiplication operation by offloading the component without using runtime profiling on the local mobile. The TT in traditional offloading with profiling attribute represents the turnaround time of the application by including the profiling mechanism on local mobile device.

Table 5. 32: Comparison of the Turnaround Time of the Matrix Multiplication Operation in Local and Remote Execution

Computational Length	**TT on Local SMD (Without Profiling)**	**TT on Local SMD (Including Profiling)**	**TT in DEAP**	**TT in Traditional Offloading (without Profiling)**	**TT in Traditional Offloading (Including Profiling)**
160*160	3653	4254	3294	16431	20326
170*170	4276	5038	3889	18296	22947
180*180	4781	5675	4308	21132	26334
190*190	5030	5930	4451	22170	27521
200*200	6321	7455	5661	26061	32856
210*210	7039	8346	6287	27927	35521
220*220	7777	9408	6929	29878	38438
230*230	8888	10639	7916	33920	43587
240*240	10735	12753	9589	38052	49659
250*250	13090	15333	11724	44310	58277
260*260	13642	16375	12177	46841	61751
270*270	14471	17279	12805	50412	66025
280*280	16411	19360	14546	55178	72673
290*290	20524	24018	18448	64414	86481
300*300	20706	24721	18599	67346	89960
310*310	21185	25656	18883	68692	92046
320*320	27028	31939	24444	80922	110277
330*330	28452	33760	25317	84709	115334
340*340	39691	45333	34666	110506	152657
350*350	38570	44698	35016	108677	149821
360*360	40096	46936	36406	114062	157308
370*370	44896	52192	40720	125149	173165
380*380	48088	55901	43402	134100	185315
390*390	55560	62952	50381	148747	206520

Computational Length	TT on Local SMD (Without Profiling)	TT on Local SMD (Including Profiling)	TT in DEAP	TT in Traditional Offloading (without Profiling)	TT in Traditional Offloading (Including Profiling)
400*400	57339	64397	51775	156489	215322
410*410	62405	73252	55762	171252	237861
420*420	63159	74057	56467	184056	251421
430*430	74424	85789	67278	213507	292150
440*440	78163	90390	70400	221092	303719
450*450	99286	112628	91038	262697	367077

Table 5.33 compares the execution time of the power compute operation of the application in local and remote execution scenarios. The execution time on the local device represents the turnaround time of the power compute operation on the local mobile device. The TT in POP of DEAP attribute shows total time taken in the executing power compute operation on DEAP server application and returning the results to the local mobile device. The TT in SOP of DEAP shows the turnaround time of the power compute operation by offloading the component at runtime.

Table 5. 33: Comparison of the Turnaround Time (ms) of the Power Compute Operation in Local and Remote Execution

Computational Length	TT on Local SMD	TT in Traditional Computational Offloading without Profiling	TT in SOP of DEAP Client including Profiling
2^1000000	51	7175	7284
2^2000000	80	7587	7786
2^3000000	110	7751	8023
2^4000000	140	7779	8150
2^5000000	176	7965	8437
2^6000000	206	8071	8617
2^7000000	233	8183	8802
2^8000000	269	8283	8980
2^9000000	293	8471	9321
2^10000000	341	8658	9582
2^20000000	373	9877	11961

2^30000000	920	12029	14665
2^40000000	1216	12999	16529
2^50000000	1501	13444	17828
2^60000000	1767	15356	20629
2^70000000	2070	17067	23250
2^80000000	2334	18092	25117
2^90000000	2615	19188	27066
2^100000000	2896	20389	29124
2^200000000	6386	32645	49549
2^300000000	8509	46475	73446
2^400000000	11405	61280	108268
2^500000000	14105	75534	142978
2^600000000	16887	84686	173497
2^700000000	19182	112309	220728
2^800000000	22480	126616	223406
2^900000000	25580	135190	282353
2^100000000	28237	138851	304233
2^1900000000	68365	139471	473860
2^2000000000	69044	265724	622062

Table 5.34 compares the ECC of sorting operating of the application in different scenarios. The ECC represents the energy consumed by executing the sorting service components of the application on mobile device and saving the resultant preferences file on the local mobile device. The ECC of the application is evaluated without involving the runtime profiling mechanism and by including the runtime profiling mechanism. The ECC in DEAP attribute shows total ECC of DEAP client application in accessing the sort service on DEAP server application and saving the resultant preferences file on the local mobile device. The ECC of the application in traditional offloading without profiling attribute shows the turnaround time of the sorting service operation by offloading the component without using runtime profiling on the local mobile. The ECC in traditional offloading with profiling attribute represents the total ECC of the application by including the profiling mechanism on local mobile device.

Table 5. 34: Comparison of Energy Consumption Cost of Sorting Operation in Local and Remote Execution

Computational Length	ECC on Local SMD (Without Profiling)	ECC on Local SMD (Including Runtime Profiling)	ECC in POP of DEAP	ECC in Traditional Offloading (Without Profiling)	ECC in Traditional Offloading (Including Profiling)
11000	21.1	21.6	7.4	49.8	50.3
12000	23.3	30.9	8.3	53.8	61.4
13000	25.1	35.4	9	56.5	66.8
14000	25.3	36.4	9.3	60.6	71.7
15000	25.3	43.8	9.3	61.9	80.4
16000	27.2	46.5	9.5	64.6	83.9
17000	28.9	53.8	10.7	70.7	95.6
18000	29.4	59.5	10.8	73.8	103.9
19000	30.4	64.6	11	76.7	110.9
20000	30.8	70.5	11.2	75.6	115.3
21000	33.1	76.4	12.3	79.7	123
22000	35.9	83.9	12.5	82.1	130.1
23000	36.5	90.8	12.7	87.7	142
24000	34.8	97.5	12.7	90.3	153
25000	37.1	104.7	14.1	95.4	163
26000	38.9	112.9	14.6	105.7	179.7
27000	40	120.2	14.8	109.1	189.3
28000	43.4	129.1	14.8	114.9	200.6
29000	45	137.8	14.9	120.5	213.3
30000	46	147.4	15.3	125.1	226.5
31000	47.6	154.8	15.3	131.6	238.8
32000	48.9	165.4	15.4	136.3	252.8
33000	52.8	175.4	17	150.3	272.9
34000	56.6	186.1	17	159.4	288.9
35000	56.9	196.3	17.3	165.3	304.7
36000	57.9	206.5	17.5	173.8	322.4
37000	59.7	217.9	17.6	183	341.2
38000	61.7	230	18.5	186.8	355.1
39000	65.3	241.1	21.8	195.9	371.7
40000	68.6	253.6	23	201.4	386.4

Table 5.35 compares the ECC of matrix multiplication operating of the application in in local and remote execution. The ECC represents the energy consumed by executing

the matrix multiplication components of the application on mobile device and saving the resultant preferences file on the local mobile device. The ECC of the application is evaluated without involving the runtime profiling mechanism and by including the runtime profiling mechanism. The ECC in DEAP attribute shows total ECC of DEAP client application in accessing the matrix multiplication operating on DEAP server application and saving the resultant preferences file on the local mobile device. The ECC of the application in traditional offloading without profiling attribute shows the total ECC of the matrix multiplication operating by offloading the component without using runtime profiling on the local mobile. The ECC in traditional offloading with profiling attribute represents the total ECC of the application by including the profiling mechanism on local mobile device.

Table 5. 35: Comparison of Energy Consumption Cost of Matrix Operation in Local and Remote Execution

Computational Length	ECC on Local SMD (Without Profiling)	ECC on Local SMD (Including Runtime Profiling)	ECC in POP of DEAP	ECC in Traditional Offloading (Without Profiling)	ECC in Traditional Computational Offloading (Including Profiling)
160*160	12.9	16	10.8	39.9898	43.1
170*170	13.4	18.1	11.2	40.9927	45.7
180*180	14.7	18.9	11.9	43.998	48.2
190*190	15.2	19.3	12	44.5995	48.7
200*200	16.3	19.9	12.8	44.4027	48
210*210	17.2	21.4	13	46.5297	50.7
220*220	20	25.1	14.5	49.3349	54.4
230*230	21.5	26.5	15.5	52.1504	57.2
240*240	22	27.3	16.2	52.8534	58.2
250*250	24.1	29.8	17.3	58.0603	63.8
260*260	24.2	32.4	18.5	59.1631	67.4
270*270	27.4	35.9	19.3	64.5291	73
280*280	28.7	37	20.3	65.4867	73.8
290*290	34.5	42.8	23	71.8974	80.2
300*300	35.2	45.7	24.2	71.9067	82.4
310*310	39.7	50.2	25.6	76.7187	87.2

Computational Length	ECC on Local SMD (Without Profiling)	ECC on Local SMD (Including Runtime Profiling)	ECC in POP of DEAP	ECC in Traditional Offloading (Without Profiling)	ECC in Traditional Computational Offloading (Including Profiling)
320*320	41.1	51.8	29.3	79.2983	90
330*330	44.4	55.3	31.5	82.7353	93.6
340*340	45.5	57.3	32.6	85.8619	97.7
350*350	51.4	63.4	35.7	93.1617	105.2
360*360	54.3	66.6	37	95.7239	108
370*370	63.2	78.5	41.6	105.5589	120.9
380*380	65.7	81.2	45	109.1943	124.7
390*390	67	82.6	47.3	109.8383	125.4
400*400	67.4	83.5	49.3	114.1728	130.3
410*410	69.1	83.3	53.7	115.1444	129.3
420*420	69.2	83.8	56.8	119.0066	133.6
430*430	69.8	87.8	57.1	121.4333	139.4
440*440	70	90.6	61.6	126.8541	147.5
450*450	71.5	91.8	65.3	131.6952	152

Table 5.36 compares the ECC of power compute operating of the application in different scenarios. The ECC of the application is evaluated without involving the runtime profiling mechanism and by including the runtime profiling mechanism. The ECC of the application in SOP of DEAP client application attribute shows the total ECC of the power compute operating by offloading the component at runtime.

Table 5. 36: Comparison of Energy Consumption Cost of Power Compute Operation in Local and Execution

Compute length	ECC on Local SMD	ECC in the SOP of Runtime Offloading (Without Profiling)	ECC in the SOP of DEAP client Application (Including Profiling)
2^1000000	2.2	5.4	5.4
2^2000000	2.3	6.5	6.8
2^3000000	2.4	6.8	6.9
2^4000000	2.5	7.9	7.9
2^5000000	2.8	8.1	8.3
2^6000000	3.9	8.6	8.9
2^7000000	4.2	9.5	9.9
2^8000000	5.2	9.6	9.9

Compute length	**ECC on Local SMD**	**ECC in the SOP of Runtime Offloading (Without Profiling)**	**ECC in the SOP of DEAP client Application (Including Profiling)**
2^9000000	4.3	10	11.3
2^10000000	4.8	10.5	11.4
2^20000000	3.9	13.2	15.1
2^30000000	5.3	15.1	16.2
2^40000000	4.5	17.1	20
2^50000000	4.5	19.8	25.6
2^60000000	5.4	23.2	28.6
2^70000000	6.1	25.1	30.8
2^80000000	6.3	26.8	34.1
2^90000000	6.4	30.2	43.8
2^100000000	6.4	31.3	45.6
2^200000000	12.7	56.1	78.5
2^300000000	15.4	72.7	101.8
2^400000000	21.8	79.6	109.2
2^500000000	21.6	88.2	128
2^600000000	25.3	106.4	149.7
2^700000000	30.6	126.4	173.3
2^800000000	34.3	143.3	193.3
2^900000000	36.2	164.9	218.9
2^100000000	38.9	181.1	249.8
2^1900000000	61.7	343.6	447.6
2^2000000000	67	351	460.7

Table 5.37 compares the RAM utilization for sorting operating of the application in different scenarios. The sorting length attribute shows the length of array which is being sorted in each instance of application execution. The RAM in local application execution attribute shows the amount of memory allocated to the sorting service component of the application on local mobile device. The attribute of RAM in remote application execution shows the amount of memory allocated to DEAP client application on local mobile device for accessing the sorting service on the remote DEAP server application.

Table 5. 37: Comparison of RAM Allocation to Sorting Service in Local Execution and POP of DEAP client Application

Length of Sort List	RAM in Local Execution (MB)	RAM in DEAP Client Application (MB)
11000	10.148	1.165
12000	10.154	1.232
13000	10.15	1.248
14000	10.209	1.442
15000	10.167	1.559
16000	10.173	1.566
17000	10.177	1.654
18000	10.179	1.925
19000	10.185	1.938
20000	10.193	1.938
21000	10.197	2.324
22000	10.2	2.355
23000	10.204	2.314
24000	10.213	2.376
25000	10.21	2.382
26000	10.215	2.426
27000	10.218	2.854
28000	10.221	2.883
29000	10.224	2.891
30000	10.227	2.898
31000	10.231	2.922
32000	10.236	3.186
33000	10.257	3.518
34000	10.242	3.531
35000	10.246	3.553
36000	10.248	3.559
37000	10.254	3.664
38000	10.258	3.68
39000	10.261	3.743
40000	10.265	5.344

Table 5.38 compares the RAM utilization for matrix multiplication operation of the application in different scenarios. The matrix size attribute represents the size of 2-D array which are used in the matrix multiplication operation. The RAM in local application

execution attribute shows the amount of memory allocated to the matrix multiplication service component of the application on local mobile device. The attribute of RAM in remote application execution shows the amount of memory allocated to DEAP client application on local mobile device for accessing the sorting service on the remote DEAP server application.

Table 5. 38: Comparison of RAM Allocation to Matrix Multiplication Service in Local Execution and POP of DEAP client Application

Length of Matrices	**RAM in Local Execution**	**RAM in DEAP Client Application**
160*160	2.78	1.695
170*170	3.17	1.865
180*180	3.58	2.094
190*190	3.99	2.36
200*200	4.43	2.438
210*210	4.88	2.887
220*220	5.36	3.119
230*230	5.87	3.524
240*240	6.39	3.576
250*250	6.94	3.752
260*260	7.51	4.193
270*270	8.1	4.448
280*280	8.71	4.842
290*290	9.35	5.616
300*300	10	5.888
310*310	10.6	6.335
320*320	11.4	6.498
330*330	12.1	7.171
340*340	12.9	7.344
350*350	13.7	7.696
360*360	14.5	8.311
370*370	15.3	8.319
380*380	16.2	9.141
390*390	17	9.754
400*400	17.9	9.88
410*410	18.9	11.476
420*420	19.2	11.912
430*430	20.8	12.263
440*440	21.8	12.549

Length of Matrices	**RAM in Local Execution**	**RAM in DEAP Client Application**
450*450	22.8	13.056

The power compute component of the application is offloaded to remote server node in the SOP of DEAP client application. Therefore, the execution takes place on the remote virtual device instance and therefore the cost of RAM allocation is eliminated on the local mobile device. Table 5.39 compares the CPU utilization in execution of mobile application in different scenarios. The computational length attribute shows the range of computational intensities for which the CPU utilization is evaluated. Percentage of CPU utilization in local application execution, and percentage of CPU utilization for DEAP client application in the SOP and POP of DEAP server application are presented. The attribute of MIPS utilization shows CPU utilization in the units of Millions of Instruction Per Second on the mobile device for executing the application on local mobile devise and accessing the services of DEAP server application.

Table 5. 39: Comparison of CPU Utilization in Local Application Execution and Remote DEAP Based Remote Application Execution

Computational Service	**Computational Length**	**%CPU in Local Execution**	**Average MIPS in Local Execution**	**%CPU in DEAP Based Execution**	**Average MIPS in DEAP Based Execution**
Sort	11000-40000	48.67	1163	25.5	609.2
Matrix Multiplication	160*160-4560*450	45.46	1086	35.4	845.7
Power Compute	2^1000000-2^2000000000	48.04	1148	3	71. 7

Table 5.40 compares the size of data transmitted over the wireless network medium for sorting service in offloading computational load in the proposed DEAP framework and traditional application offloading technique.

Table 5. 40: Comparison of the Data Transmission Using Traditional Offloading Technique and DEAP Framework for Sorting Service

Length of Sort List	Data Transmission in Traditional Offloading (KB)	Data Transmission in DEAP Based Offloading (KB)
11000	752.4	183
12000	820.4	200
13000	888.4	218
14000	950.4	235
15000	1026.4	253
16000	1086.4	270
17000	1162.4	288
18000	1230.4	306
19000	1298.4	323
20000	1360.4	341
21000	1420.4	358
22000	1480.4	376
23000	1572.4	393
24000	1632.4	411
25000	1694.4	429
26000	1754.4	446
27000	1846.4	464
28000	1914.4	481
29000	1982.4	499
30000	2042.4	516
31000	2092.4	534
32000	2153.84	552
33000	2215.28	568
34000	2276.72	587
35000	2399.6	604
36000	2399.6	622
37000	2461.04	639
38000	2522.48	657
39000	2583.92	675
40000	2645.36	692

Table 5.41 compares the size of data transmitted over the wireless network medium for matrix multiplication service in offloading computational load in the proposed DEAP framework and traditional application offloading technique.

Table 5. 41: Comparison of the Data Transmission Using Traditional Offloading Technique and DEAP Framework for Matrix Multiplication Operation

Length of Matrices	**Data Transmission in Traditional Offloading (KB)**	**Data Transmission in DEAP Framework Based Offloading (KB)**
160*160	5739.44	463
170*170	6538.16	528
180*180	7377.84	595
190*190	8217.52	664
200*200	9118.64	639
210*210	10040.24	705
220*220	11023.28	774
230*230	12067.76	847
240*240	13132.72	923
250*250	14259.12	1002
260*260	15426.48	1084
270*270	16634.8	1169
280*280	17884.08	1258
290*290	19194.8	1350
300*300	20526	1445
310*310	21754.8	1543
320*320	23393.2	1646
330*330	24826.8	1754
340*340	26465.2	1865
350*350	28103.6	1979
360*360	29742	21629
370*370	31380.4	2219
380*380	33223.6	2343
390*390	34862	2471
400*400	36705.2	2602
410*410	38753.2	2737
420*420	39367.6	2874
430*430	42644.4	3015
440*440	44692.4	3160
450*450	46740.4	3308

5.7 Conclusion

The proposed framework is tested on the Android virtual device and benchmarking is done by evaluating the prototype application on the real mobile device. Data are collected by sampling the evaluation parameters with 30 different computational intensities of mobile application. The point estimator of each experiment is determined by finding the sample mean of the sample space of 30 values in each experiment. The value of sample mean is signified by finding the interval estimate with 99% confidence for the sample space of 30 values in each experiment.

It is concluded that DEAP successfully leveraged the application processing services of computational cloud for outsourcing the resource intensive logic of mobile applications. DEAP framework successfully implemented the POP to access the services of DEAP server application by employing SaaS model of computational cloud. However, to sustain the feature of elasticity in mobile application, runtime computational offloading is successfully implemented in the SOP of DEAP framework which utilizes the IaaS services of computational cloud. The evaluation of the DEAP framework on the Android virtual device indicates the viability of DEAP framework for leveraging the application processing services of computational clouds to resources constraint SMDs. Benchmarking of the prototype application with the diverse computational intensities of the application validates performance gains of the DEAP framework for intensive applications in mobile cloud computing.

CHAPTER 6

Results and Discussion

This chapter analyzes the experimental results presented in chapter 5 for signifying the usefulness of the proposed DEAP framework. The chapter is organized into five sections. Section 6.1 analyzes the results of application processing on local mobile device. Section 6.2 investigates results of distributed application processing in the traditional runtime computational offloading. Section 6.3 analyzes the results of application processing in DEAP based distributed processing of mobile application. Section 6.4 compares results of different experimental scenarios for local and distributed processing of mobile application. Section 6.5 concludes the chapter with the significance of DEAP framework.

6.1 Analysis of Application Execution on Local Mobile Device

The prototype mobile application is tested on local mobile device to evaluate resources utilization (CPU, RAM, and Battery power) and Turnaround Time of the mobile application. Table 5.1 shows RAM allocation for sorting service in 30 different experiments. The allocated RAM for sorting a list of 11000 values is determined as 10.148(+/-) 0.0061MB with 99% confidence interval in the sample space of 30 values, which shows that the possible range of RAM allocation is between 10.1419 MB and 10.1541MB. The value of SD for RAM allocation shows the variation in the values of same sample space for each experiment. For instance, the variation of RAM allocation varies 0.013 MB for sorting a list of 11000 values in 30 different experiments which constitutes 0.1281 percent of the average RAM allocation on SMD. Similarly, the allocated RAM for sorting a list of 25000 values in the sample space of 30 values is determined as 10.21(+/-)0.0098 MB with 99% confidence interval, which shows that the possible range of RAM allocation is between 10.2095 MB and 10.2105 MB. The value of SD for RAM allocation shows 0.001 MB variation for sorting a list of 25000 values in 30 different experiments which constitutes 0.0098 percent of the average RAM allocation on SMD.

Figure 6.1 shows the allocation of RAM to the sorting component of mobile application on local mobile device in 30 different experiments. The allocation of memory for the sorting service varies according to the length of the list being sorted. For instance, in sorting the list of 15000 values on the average 10.167 MB RAM is allocated, whereas in sorting the list of 40000 values 10.265 MB RAM is allocated on the SMD. Sorting service saves the preferences file of list being sorted on local SMD. For that reason, in saving the preferences file on local device the allocation of heap size and allocated RAM increases accordingly. The average RAM allocation for the sorting service on mobile device is determine as 10.21 MB for sorting list of 11000-40000 with the RSD 0.040 percent.

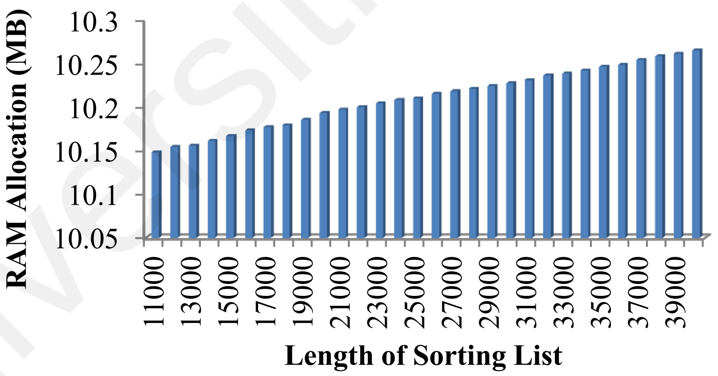

Figure 6. 1: Allocation of RAM for Sorting Service on SMD

Table 5.2 summarized RAM allocation for matrix multiplication service in 30 different experiments. For instance, the allocated RAM for 2-D array lists of 160*160 values in the sample space of 30 values is determined 10.454(+/-).0014 with 99% confidence interval, which shows that the possible range of RAM allocation is between 10.4526 MB and 10.4554 MB. The value of SD for RAM allocation shows the variation in the values of same sample space for each experiment. For instance, the variation of RAM allocation varies 0.0029MB for multiplying list of 160*160 length in 30 different experiments which constitutes 0.0277 percent of the average RAM allocation on SMD. Similarly, the allocated RAM for sorting a list of 390*390 values in the sample space of 30 values is determined as 12.4562(+/-).0511 MB with 99% confidence interval, which shows that the possible range of RAM allocation is between 12.4051 MB and 12.5073 MB. The value of SD for RAM allocation shows 0.1085 MB variation for matrix size of 290*290 in 30 different experiments which constitute 0.8711 percent of the average RAM allocation on SMD.

Figure 6.2 shows the allocation of RAM for matrix multiplication service of mobile application on local mobile device in 30 different experiments. The allocation of memory for the matrix multiplication service varies according to the length of the matrix being multiplied. For instance, in multiplying the matrix of size 250*250 values, on the average 10.8317 MB, whereas in multiplying the matrix of size 450*450 values, 13.1003MB RAM is allocated on the SMD. Matrix multiplication service saves the preferences file of the resultant matrix on SMD. Therefore, in saving the preferences file on local device the allocation the heap size and allocated RAM increases accordingly. The average RAM allocation for the matrix multiplication service on mobile device is determine as 11.5034 MB for matrix size of 160*160 – 450*450 with the RSD 7.7 percent.

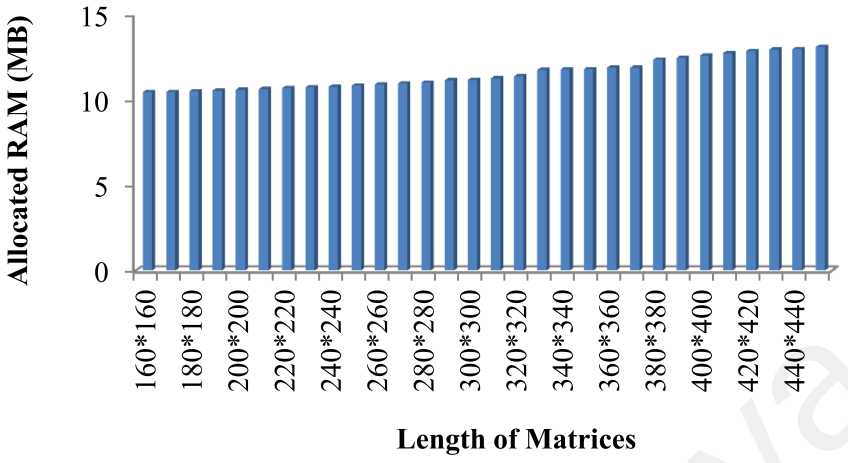

Figure 6. 2: RAM Allocation to Matrix Multiplication Service on Local SMD

Table 5.3 summarized the allocation of RAM for the execution of power compute service component of the application on local mobile device. The computational intensity of power compute service varies between 2^1000000 and 2^2000000000. The allocated RAM for power compute service is determined as 10.11(+/-).00045 MB with 99% confidence interval, which shows that the possibility of RAM allocation for power compute service with different computational intensities is between 10.109 MB and 10.110 MB. The value of SD for RAM allocation shows the variation in the values of sample space for the experimentation of power compute service with 30 different computational intensities. Hence, the variation of RAM allocation varies 0.0017 MB for power compute service in 30 different experiments which constitutes 0.016815035 percent of the average RAM allocation on SMD.

Table 5.4 summarized the TT of the sorting service component of the application. The TT of sorting service is evaluated with 30 different computational intensities of the sorting operation (sort list size of 11000-40000). It is observed that the TT of the sorting operation varies with the computational intensity of sorting operation. For instance, the TT in sorting a list of 11000 values in the sample space of 30 values is determined as 4876(+/-

)333 ms with 99% confidence interval, which shows that the possible range of TT is between 4543 ms and 5209 ms. The value of SD for TT shows the variation in the values of same sample space for each experiment. For instance, the variation of TT is 706 ms for sorting a list of 11000 values in 30 different experiments which constitutes 14 percent of the average TT on SMD. Similarly, the TT for sorting a list of 25000 values in the sample space of 30 values is determined as 16950(+/-)431 ms with 99% confidence interval, which shows that the possible range of TT is between 16519 ms and 17381 ms. The value of SD for TT shows 915 ms variation for sorting a list of 25000 values in 30 different experiments which constitutes 5 percent of the average TT value.

Figure 6.3 shows the increase in TT of the sorting operation on local mobile device in 30 different experiments. The TT for the sorting service varies according to the length of the list being sorted. The value of TT includes the processing time for performing sorting operation and the time taken in saving the resultant preferences file to the data file of the local mobile device. It is observed that by increasing the length of sorting list, the processing time of performing sorting operation and the time taken in saving preferences file increases accordingly. For instance, sorting the list of 15000 values takes on the average 7406 ms, whereas sorting the list of 40000 values takes 31207 ms on the SMD. Experimental results indicate that the TT in sorting the list of 40000 values increases 84.4 percent as compared to sorting the list of 11000 values. However, the average TT for the sorting service on mobile device is determine as 17426 ms for sorting list of 11000-40000 with the 45.2 percent RSD.

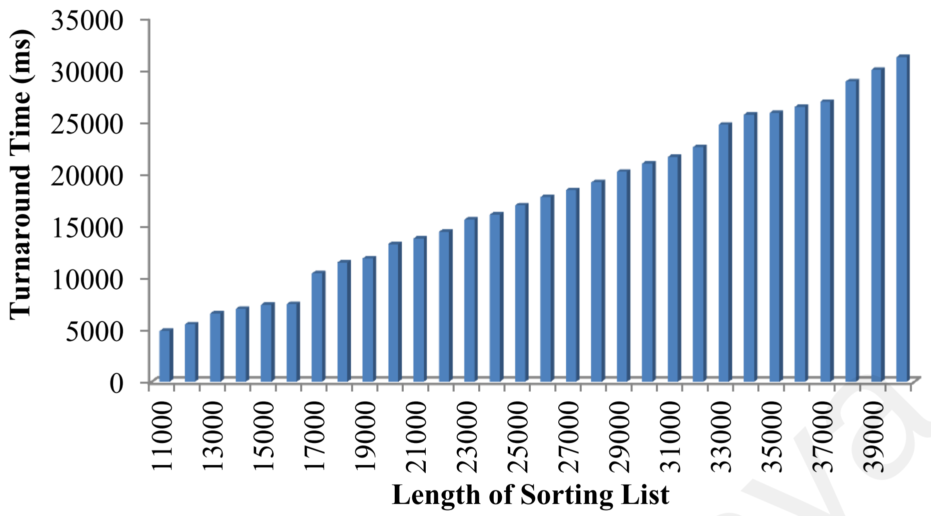

Figure 6. 3: Turnaround Time of Sorting Service on Local SMD

Table 5.5 summarized the TT of the matrix multiplication service component of the application on local mobile device. The TT of matrix multiplication service is evaluated with 30 different computational intensities of the matrix multiplication operation (matrix size 160*160-450*450). It is observed that the TT of the matrix multiplication operation varies with the computational intensity of matrix multiplication operation. For instance, the TT in multiplying 2-D arrays of 160*160 size in the sample space of 30 values is determined as 3653(+/-)90 ms with 99% confidence interval, which shows that the possible range of TT is between 3563 ms and 3563 ms. The value of SD for TT shows the variation in the values of same sample space for each experiment. For instance, the variation of TT is 191 ms for sorting a list of 11000 values in 30 different experiments, which constitutes 5 percent of the average TT on SMD. Similarly, the TT in multiplying 2-D arrays of 310*310 values in the sample space of 30 values is determined as 21185(+/-)1813 ms with 99% confidence interval, which shows that the possible range of TT is between 19372 ms and 19372 ms. The value of SD for TT shows 3849 ms variation for multiplying 2-D arrays of size 310*310, in 30 different experiments which constitutes 18 percent of the average TT value. Figure 6.4 shows the increase in TT of the matrix multiplication operation on local mobile device in 30 different experiments. The TT for the matrix multiplication varies according to the length of the 2-D arrays which are being multiplied. The value of TT includes the processing time for performing matrix multiplication operation and the time taken in saving the resultant preferences file to the data file of the local mobile device. It is observed that by increasing the size of 2-D arrays, the processing time of performing matrix multiplication operation and the time taken in saving preferences file increases accordingly.

For instance, multiplying 2-D arrays of 200*200 values takes on the average 6321 ms, whereas multiplying 2-D arrays of 450*450 values takes 99286 ms on the SMD. Experimental results indicate that the TT in multiplying the 2-D arrays of 450*450 increases 96.3 percent as compared to multiplying the 2-D arrays of 160*160 values. The average TT for the Matrix multiplication service on mobile device is determine as 31190 ms for multiplying 2-D arrays of 160*160-450*450 size with the RSD 12 percent.

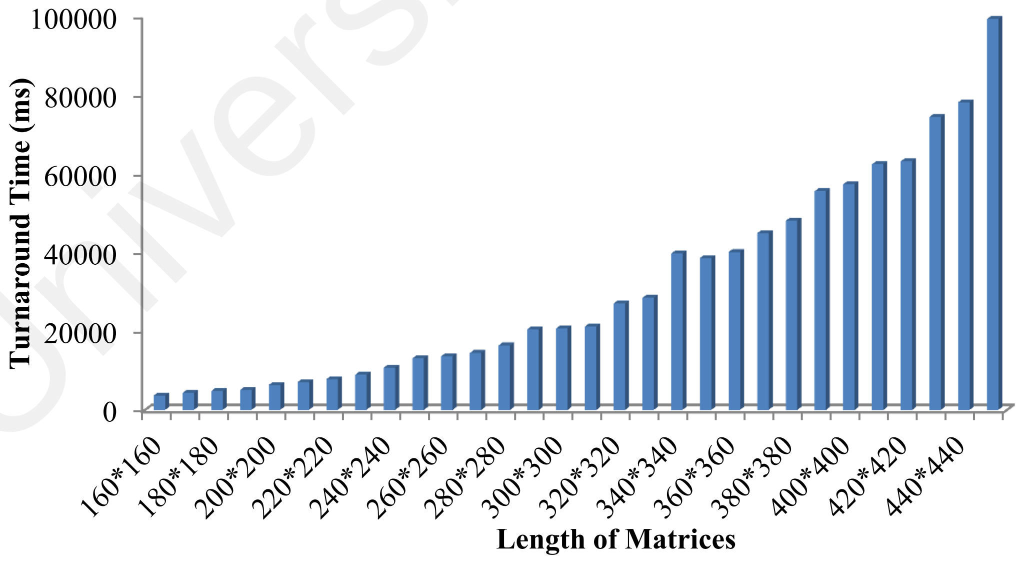

Figure 6. 4: Turnaround Time of Matrix Multiplication Operation on Local SMD

Table 5.6 summarized the TT of the power compute service component of the application on local mobile device. The TT of power compute service is evaluated with 30 different computational intensities of the power computing operation (2^1000000-2^2000000000). The TT in computing power of 2^1000000 in the sample space of 30 values is determined as 51(+/-)5 ms with 99% confidence interval, which shows that the possible range of TT is between 46 ms and 51 ms. The variation of TT is 10 ms for computing the power of 2^1000000 in 30 different experiments which constitutes 19.6 percent of the average TT on SMD. Similarly, the TT in power computing operation of 2^10000000 computational length is determined as 373(+/-)18 ms with 99% confidence interval in the sample space of 30 values, which shows that the possible range of TT is between 391 ms and 355 ms. The value of SD for TT shows 38 ms variation for computing 2^10000000, in 30 different experiments which constitutes 10.2 percent of the average TT value.

Figure 6.5 shows the increase in the TT of the power compute operation on local mobile device in 30 different experiments. The TT for the power compute varies according to the computational length of the Power compute service. The value of TT includes the processing time for performing matrix multiplication operation. It is observed that by increasing the computational length of compute service, the processing time of performing power compute operation increases accordingly. For instance, computing 2^10000000 takes 341 ms, whereas computing 2^2000000000 takes 69044 ms on the SMD. Experimental results indicate that the TT in computing 2^2000000000, increases 99.9 percent as compared to computing 2^1000000. However, the average TT for the Power compute service on mobile device is determine as 10259 ms for computational length of 2^100000-2^2000000000. It is observed that the TT of the power computing operation varies according to the computational length of Power compute service (values of base and exponent).

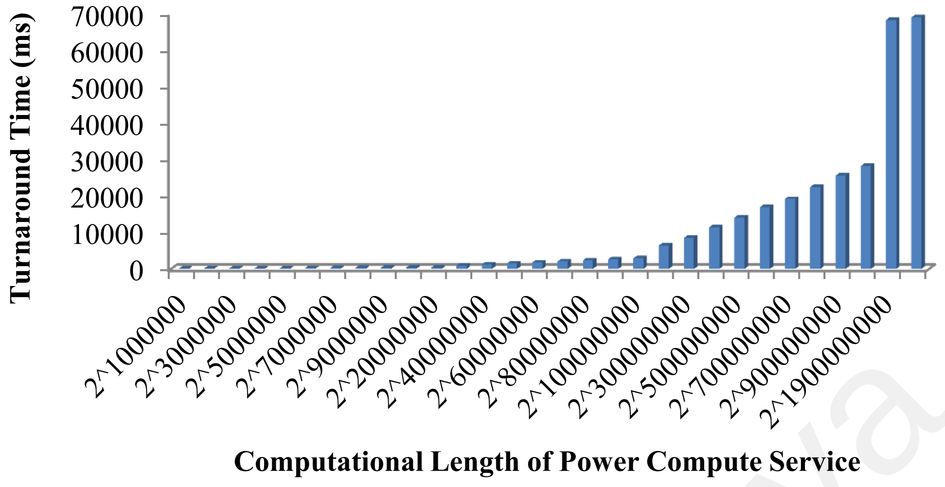

Figure 6. 5: Turnaround Time of Power Compute Service of the Application on Local SMD

Table 5.7 presented the ECC in processing sorting service component of the application on local mobile device. The ECC of sorting service is evaluated with 30 different computational intensities of the sorting operation (sort list size of 11000-40000). It is observed that the ECC of the sorting operation varies with the computational intensity of sorting operation. For instance, the ECC for sorting a list of 11000 values is determined as 16.2(+/-)1.3 J with 99% confidence interval in the sample space of 30 values, which shows that the possible range of ECC is between 14.9 J and 17.5 J.

The value of SD for ECC shows the variation in the values of same sample space for each experiment. For instance, the variation of ECC is 2.8 J for sorting a list of 11000 values in 30 different experiments which constitutes 17.3 percent of the average ECC on SMD. Similarly, the ECC for sorting a list of 25000 values in the sample space of 30 values is determined as 30.4(+/-)1.6 J with 99% confidence interval, which shows that the possible range of ECC is between 28.8 J and 32 J. The value of SD for ECC shows 3.3 J variation for sorting a list of 25000 values in 30 different experiments which constitutes 3.3 percent of the average ECC value.

Figure 6.6 shows the increase in ECC of the sorting operation on local mobile device in 30 different experiments. The ECC for the sorting service varies according to the length of the list being sorted.

The value of ECC includes the energy consumed in performing sorting operation on SMD and the energy consumed in saving the resultant preferences file to the data file of the local mobile device. It is observed that by increasing the length of sorting list, the energy consumption cost of performing sorting operation and energy consumption cost in saving preferences file increases accordingly. For instance, sorting the list of 15000 values consumes on the average 18.6 J, whereas sorting the list of 40000 values consumes 55.1 J on the SMD. Experimental results indicate that the ECC in sorting the list of 40000 values increases 70.5 percent as compared to sorting the list of 11000 values. However, the average ECC for the sorting service on mobile device is determine as 33.4 J for sorting list of 11000-40000 with the 38 percent RSD.

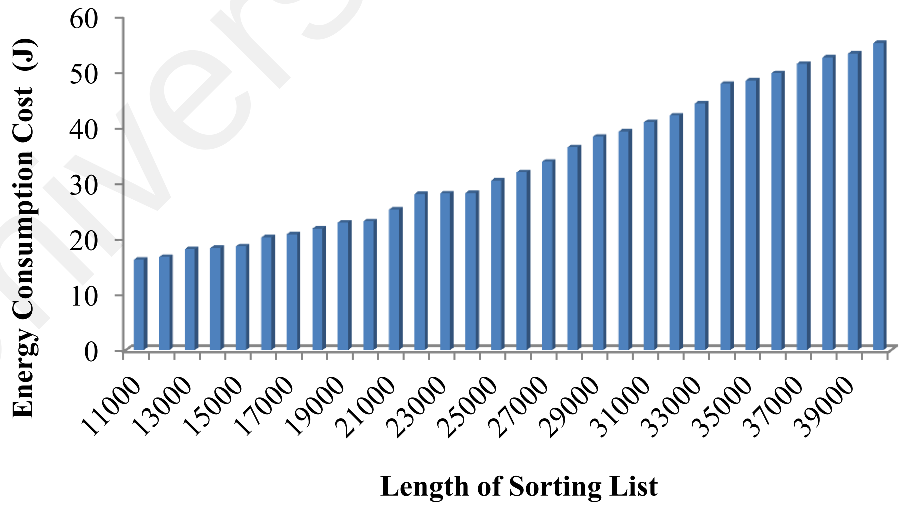

Figure 6. 6: Energy Consumption Cost of Sorting Service on Local SMD

Table 5.8 presented the ECC in processing matrix multiplication service component of the application on local mobile device. The ECC of matrix multiplication is evaluated with 30 different computational intensities of the matrix multiplication (2-D array size

160*160*450-450). It is observed that the ECC of the matrix multiplication operation varies with the computational intensity of 2-D arrays. For instance, the ECC for multiplying 2-D arrays of size 160*160 values is determined as 12.9(+/-)1.3 J with 99% confidence interval in the sample space of 30 values, which shows that the possible range of ECC is between 14.4 J and 20 J. The value of SD for ECC shows the variation in the values of same sample space for each experiment. For instance, the variation of ECC is 2.8 J for multiplying 2-D arrays of size 160*160 in 30 different experiments, which constitutes 21.7 J percent of the average ECC on SMD. Similarly, the ECC for 2-D arrays of size310*310 is determined as 39.7(+/-)3.3 J with 99% confidence interval in the sample space of 30 values, which shows that the possible range of ECC is between 36.4 J and 43 J. The value of SD for ECC shows 7 J variation in multiplying 2-D arrays of size 310*310 in 30 different experiments which constitutes 17.6 percent of the average ECC value.

Figure 6.7 shows the increase in ECC of the matrix multiplication operation on local mobile device in 30 different experiments. The ECC for the matrix multiplication varies according to the computational length of the matrix multiplication service. The value of ECC includes the energy consumed in performing matrix multiplication on SMD and the energy consumed in saving the resultant preferences file to the data file of the local mobile device. It is observed that by increasing the computational length, the energy consumption cost of performing matrix multiplication operation and energy consumption cost in saving preferences file increases accordingly. For instance, matrix multiplication of 200*200 2-D array length consumes on the average 16.3 J, whereas matrix multiplication of 200*200 matrices length consumes 71.5 J on the SMD. Experimental results indicate that the ECC in matrix multiplication of 450*450 matrices length increases 81.95 percent as compared to matrix multiplication of 160*160 matrices length. The average ECC for the matrix multiplication service on mobile device is determine as 44.56 J for the computational length 160*160-450*450, with 52.6 percent RSD.

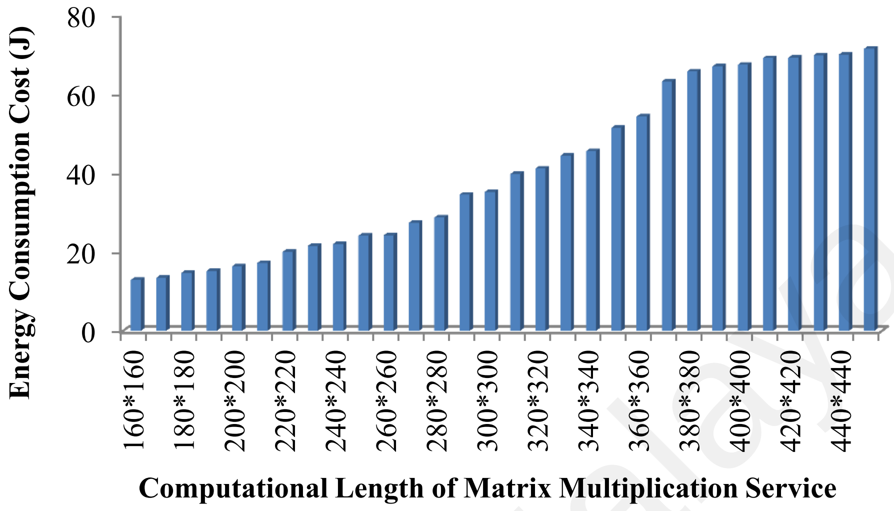

Figure 6. 7: Energy Consumption Cost of Matrix Multiplication Service on SMD

Table 5.9 presented the ECC in processing power compute service component of the application on local mobile device. The ECC of power computing is evaluated with 30 different computational intensities. It is observed that the ECC of the power compute operation varies with the computational intensities of power compute service. For instance, the ECC for computing 2^1000000 is determined as 2.2(+/-)0.3 J with 99% confidence interval in the sample space of 30 values, which shows that the possible range of ECC is between 1.9 J and 2.5 J. The value of SD for ECC shows the variation in the values of same sample space for each experiment. For instance, the variation of ECC is 0.7 J for computing 2^1000000 in 30 different experiments, which constitutes 31.8 J percent of the average ECC on SMD. Similarly, the ECC of computing 2^10000000 is determined as 4.8(+/-)0.4 J with 99% confidence interval in the sample space of 30 values, which shows that the possible range of ECC is between 4.4 J and 5.2 J. The value of SD for ECC shows 0.8 J variation in computing 2^10000000 in 30 different experiments which constitutes 16.7 percent of the average ECC value.

Figure 6.8 shows the increase in ECC of the power compute operation on local mobile device in 30 different experiments. The ECC for the power computing varies according to the computational length of the power compute service. The value of ECC includes the energy consumed in performing matrix multiplication on SMD. It is observed that by increasing the computational length, the energy consumption cost of performing power compute operation increases accordingly. For instance, computing 2^6000000 consumes on the average 3.9 J, whereas computing 2^2000000000 consumes 67 J energy on the SMD. Experimental results indicate that the ECC for computing 2^2000000000 increases 96.7 percent as compared to computing 2^1000000. The average ECC for the power compute service on mobile device is determine as 14.96 J for the computational length 2^1000000-2^2000000000 of the power compute service on SMD.

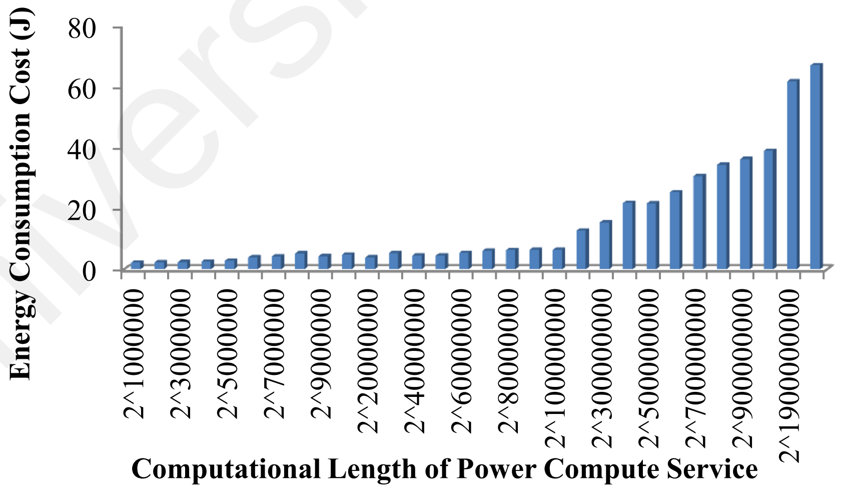

Figure 6. 8: Energy Consumption Cost of Power Compute Service on SMD

Table 5.10 summarized the average CPU utilization on local mobile device for different components of the mobile application. The CPU utilization of the mobile application depends on the computational intensities of mobile application. It is observed that the execution of the intensive components utilizes the maximum possible CPU on the SMD. Figure 6.9 shows the average CPU utilization for each of the three components of the mobile application on SMD. The CPU utilization for sorting service is evaluated with 30 different computational intensities. Sorting operation utilizes 48.67(+/-)0.96 percent (1163(+/-)22.9 MIPS) of the CPU, which shows the range of CPU utilization for the sorting service between 47.7 percent (1139.8 MIPS) and 49.63 percent (1185.7 MIPS) on the local SMD. CPU utilization for the sorting service on the SMD varies 2.62 percent (62.59 MIPS) of the average CPU utilization for sorting service on SMD.

Matrix multiplication operation utilizes 45.46(+/-)4.01 percent (1086(+/-)95.8 MIPS), which shows that the possible range of CPU utilization for matrix multiplication operation on SMD is between 41.45 percent (990.27MIPS) and 49.47 percent (1181.9 MIPS). CPU utilization for the matrix multiplication service on the SMD varies 2.62 percent (203.3 MIPS) of the average CPU utilization for matrix multiplication operation on the local SMD.

Power compute service utilizes 48.04(+/-)1.38 percent (1148 (+/-) 32.9MIPS), which shows that the possible range of CPU utilization for power computing operation on SMD is between 46.7 percent (1114.7 MIPS) and 49.42 percent (1180.7 MIPS). CPU utilization for the power compute service on the SMD varies 4.13 percent (98.6 MIPS) of the average CPU utilization for power compute operation on the local SMD.

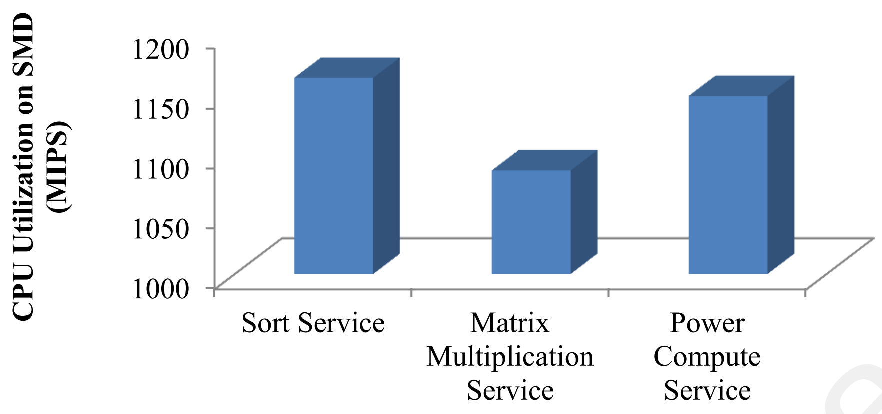

Figure 6. 9: Average CPU Utilization on SMD by the Components of Prototype Application

6.2 Analysis of Traditional Computational Offloading for MCC

Table 5.4 summarized the TT of the sorting service component of the application in traditional runtime computational offloading. Traditional computational offloading involves migration of the binary file of the mobile application and the corresponding data files at runtime.

The total TT of the component offloaded at runtime includes: 1) the time taken in saving the data states of the running instance of the component of the mobile application which is being offloaded, 2) time taken in transferring application binary code to the remote server, 3) time taken in downloading the delegated application binary code to the remote virtual machine on the cloud server node, 4) time taken in uploading the preferences (data states file) of the mobile application to remote server node, 5) time required for resuming the running state of the mobile application on the remote server node, 6) time taken in processing the application on remote machine, and 7) time taken in returning result file to the mobile device. It is observed that the TT of the components offloaded at runtime depends on two parameters. 1) The processing time of the offloaded component, this depends on the computational length of the offloaded component. 2) The data transmission time between the local and remote machine, which depends on the size of data transmission between local mobile device and remote machine. Therefore, the TT value is the total time taken in offloaded processing of the component of mobile application, which is the sum of the application processing time on the remote virtual device and timing cost of runtime component offloading (equation 3.3).

The TT in offloaded processing of the sorting service component of the mobile application is evaluated with 30 different computational intensities of the sorting operation (sort list size of 11000-40000). For instance, the TT in sorting list of 11000 values is determined as 24331(+/-) 1478 ms with 99% confidence interval in the sample space of 30 values, which shows that the possible range of total TT is between 22853 ms and 25809 ms. The value of SD for TT shows the variation in the values of same sample space for each experiment. For instance, the variation of total TT is 3138 ms for sorting a list of 11000 values in 30 different experiments which constitutes 12.9 percent of the average TT on SMD. Similarly, the TT for sorting a list of 25000 values is determined as 76615(+/-) 1636 ms with 99% confidence interval in the sample space of 30 values, which shows that the possible range of total TT is between 74979 ms and 78251 ms. The value of SD for total TT shows 3473 ms variation for sorting a list of 25000 values in 30 different experiments which constitutes 4.5 percent of the total TT value.

Figure 6.10 shows the increase in TT of the sorting operation which is offloaded at runtime in 30 different experiments. For instance, sorting the list of 15000 values takes the TT 37010 ms, whereas sorting the list of 40000 values takes 166457 ms.

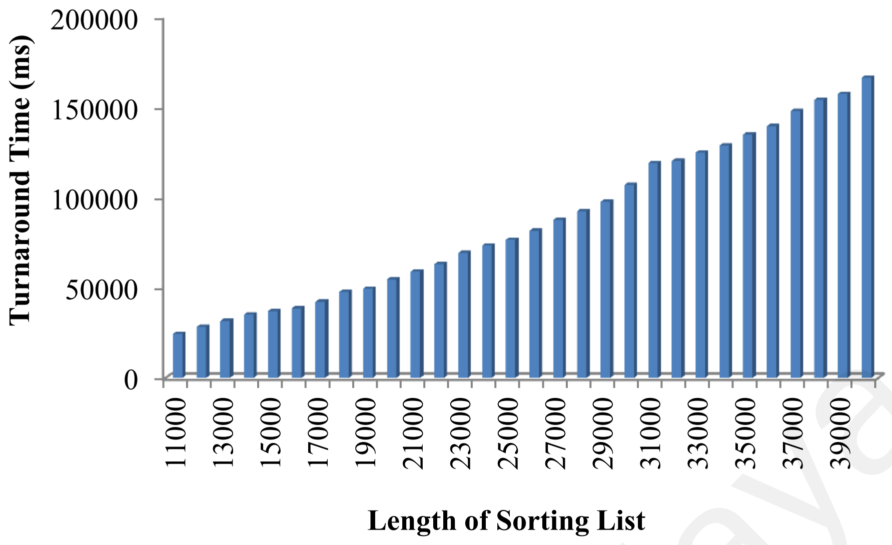

Figure 6. 10: Total Turnaround Time of Sorting Service in Traditional Computational Offloading

The total TT of offloading the matrix multiplication service component of the mobile application is evaluated with 30 different computational intensities (160*160-450*450). For instance, the total TT in offloading matrix multiplication service with computational length of matrices length 160*160 is determined as 16431(+/-)385 ms with 99% confidence interval in the sample space of 30 values, which shows that the possible range of TT is between 16046 ms and 16816 ms.

The value of SD for total TT shows the variation in the values of same sample space for each experiment. For instance, the variation of total TT is 818 ms in offloaded processing of matrix multiplication with the matrices length 160*160, in 30 different experiments, which constitutes 5 percent of the average total TT in offloaded processing. Similarly, the total TT for matrix multiplication with the matrices length of 310*310 values is determined as 68692(+/-)4058 ms with 99% confidence interval in the sample space of 30 values, which shows that the possible range of TT is between 64634 ms and 72750 ms. The value of SD for total TT shows 8615 ms variation for the multiplication of matrices length 310*310 in 30 different experiments which constitutes 12.5 percent of the average TT value.

Figure 6.11 shows the increase in TT of the matrix multiplication which is offloaded at runtime in 30 different experiments. The total TT is 18296 ms in offloaded processing of matrix multiplication operation with computational length of 1740*170 (matrices length), whereas matrix multiplication operation with the computational length of 450*450 matrices length is 37971 ms in traditional runtime offloaded processing. It indicates that the turnaround time of the application is increased in offloading highly intensive components of the mobile application at runtime.

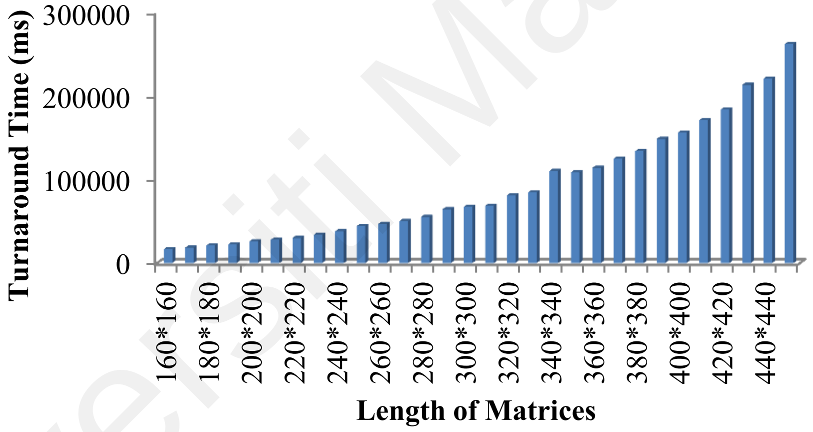

Figure 6. 11: Total Turnaround Time of Matrix Multiplication Service in Traditional Offloading

The ECC of the component offloaded at runtime includes: the energy consumed in saving the data states of the running instance of the component of the mobile application which is being offloaded, energy consumed in transferring application binary code over the wireless network medium to the remote server, energy consumed in uploading the preferences (data state file) of the mobile application to remote server node, energy consumed in processing the application on remote machine and energy consumed in returning result file over the wireless network medium to the mobile device. Hence, total ECC is the sum of energy consumed in remote application processing and energy consumption cost of runtime component offloading (equation 3.1).

Table 5.13 summarized the total energy consumed in offloaded processing of the sorting service component of the application with 30 different computational intensities. The total ECC is evaluated with 30 different computational intensities of the sorting operation (sort list size of 11000-40000). The total ECC in sorting list of 11000 values is determined as 49.7749(+/-)3.5 J with 99% confidence interval in the sample space of 30 values, which shows that the possible range of total ECC is between 46.3 J and 53.2 J. The value of SD for total ECC shows the variation in the values of same sample space for each experiment in offloaded processing. For instance, the variation of total ECC is 7.3245 J for sorting a list of 11000 values in 30 different experiments which constitutes 14.7 percent of the average ECC in offloaded processing of the sorting service component. Similarly, the total ECC for sorting a list of 25000 values is determined as 95.4014(+/-)4 J with 99% confidence interval in the sample space of 30 values, which shows that the possible range of total ECC is between 91.4 J and 99.4 J. The value of SD for total ECC shows 8.514 J variation for sorting a list of 25000 values in 30 different experiments which constitutes 8.9 percent of the total ECC value.

Figure 6.11 shows the increase in total ECC in the offloaded processing of the sorting service component of the application. For instance, in offloaded processing of sorting service with list size of 15000 values 61.9 J energy is consumed, whereas in offloaded processing of sorting service with list size of 40000 values 201.1 J energy is consumed. It shows that the total ECC is increased 75 percent for offloaded processing sorting service with sorting length of 40000 values as compared to sorting length of 11000 values.

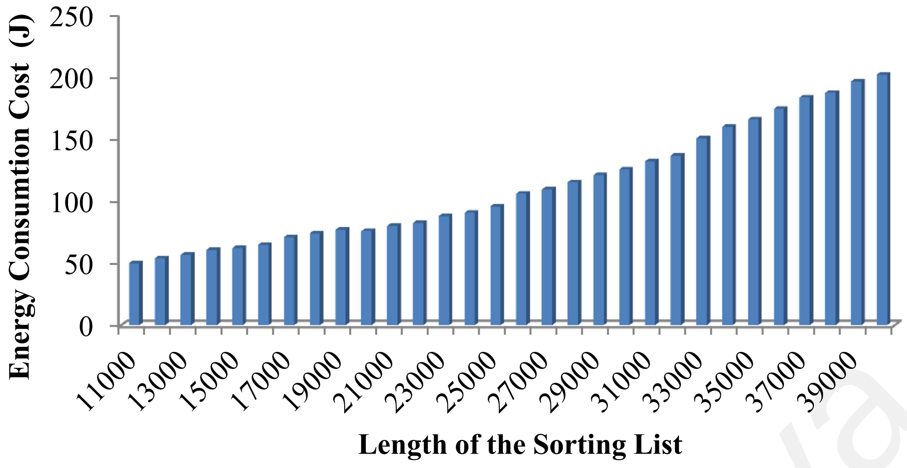

Figure 6. 12: Total Energy Consumption in Offloaded Processing of Sorting Service

Table 5.14 summarized the total energy consumed in offloaded processing of the matrix multiplication service component of the application with 30 different computational intensities. The total ECC is evaluated with 30 different computational intensities of the matrix multiplication (matrix size of 160*160-450*450). The total ECC in matrix multiplication of 160*160 values is determined as 39.9898(+/-)5.54 J with 99% confidence interval in the sample space of 30 values, which shows that the possible range of total ECC is 34.5 J and 45.5 J. The value of SD for total ECC shows the variation in the values of same sample space for each experiment in offloaded processing. For instance, the variation of total ECC is 11.8 J for multiplying 2matrices of length 160*160 in 30 different experiments which constitutes 29.4 percent of the average ECC in offloaded processing of the matrix multiplication service component.

Similarly, the total ECC in matrix multiplication of 450*450 values is determined as 131.6952(+/-)7.8 J with 99% confidence interval in the sample space of 30 values, which shows that the possible range of total ECC is 123.8 J and 139.5 J, the variation of total ECC is 16.6 J for multiplying matrixes of length 450*450 in 30 different experiments which constitutes 12.6 percent of the average ECC in offloaded processing of the matrix multiplication service component.

Figure 6.12 shows the total ECC in the offloaded processing of the matrix multiplication service component of the application. It indicates that total ECC in offloaded processing of the service increases according to the computational length of the matrix service component. For instance, 46.5 J energy is consumed in the offloaded processing of matrix multiplication service with the matrices length 210*210, whereas 119 J energy is consumed in offloaded processing of matrix multiplication service with the matrices length 420*420. It shows that the total ECC is increased 69.6 percent for offloaded processing matrix multiplication service with multiplication length of 450*450 values as compared to multiplication length of 160*160 values.

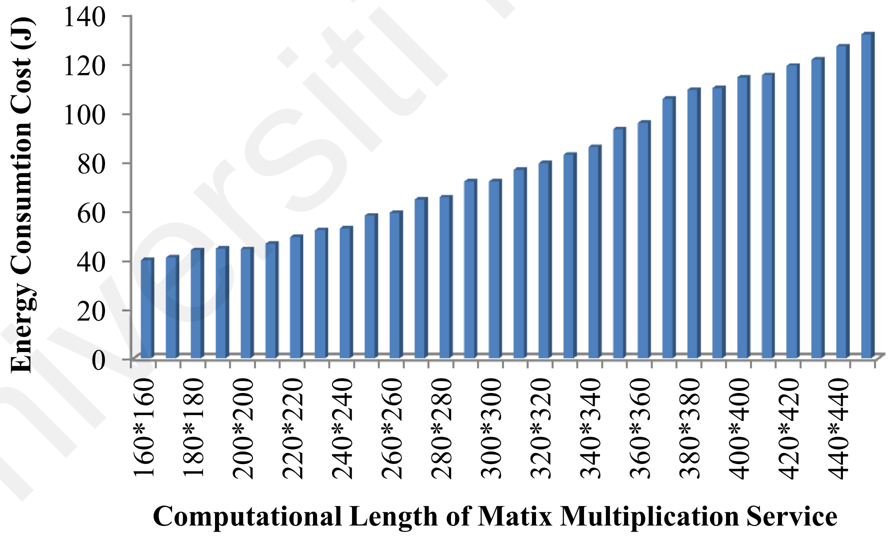

Figure 6. 13: Energy Consumption Cost of Matrix Multiplication Service in Offloaded Processing

In the traditional computational offloading techniques, the binary file of application and the data states are transmitted over the wireless network medium. Hence, the size of data transmission over the wireless network medium is determined by measuring the size of application binary file size, application preferences files size uploaded to the remote virtual machine and the application preferences file size returned to the mobile device after the completion of execution on the remote server node.

Figure 6.14 shows the increase in data transmission over the wireless network medium in offloading sort service with respect to the varying size of sorting list in 30 different experiments. It is observed that the size of application binary file remains constant in all instances of offloading sorting service application at runtime. However, the size of data file varies accordingly the size of list being sorted. For instance, linear list of 11000 values is offloaded in data file of 354 KB, whereas the list of size 40000 values is offloaded in data file size of 1300.48 KB (as shown in Table 5.15). Hence, the data file size is increased in offloading sort service component of the application with larger list size. It is observed that the average goodput of network is 841.7(+/-) 18.49 Kbps with 99.9% confidence in the sample space of 30 values for offloading sorting service at runtime.

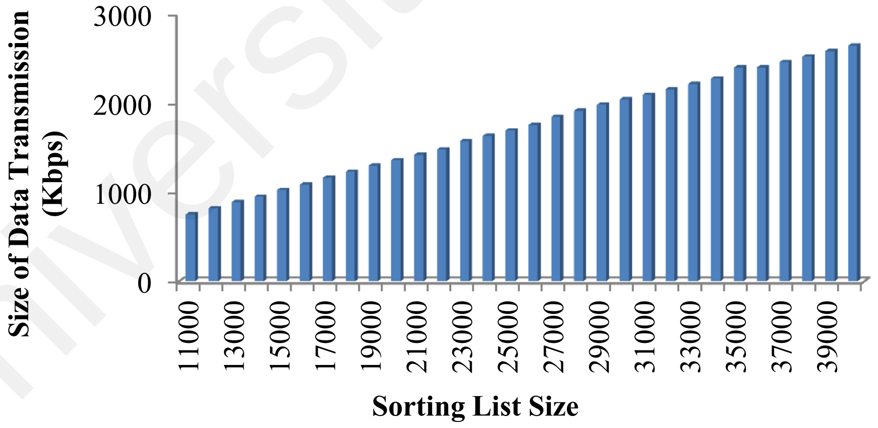

Figure 6. 14: Size of Data Transmission in Offloading Sorting Service at Runtime

Figure 6.14 shows the increase in data transmission over the wireless network medium in offloading matrix multiplication service with respect to the varying size of matrix multiplication matrices in 30 different experiments. It is observed that the size of application binary file remains constant in all instances of offloading matrix multiplication service at runtime. However, the size of data file varies according to the length of matrices being multiplied. For instance, matrices length 160*160 are offloaded in data file of 2846.72 KB, whereas the matrices length 450*450 are offloaded in data file size of 23347.2 KB. Hence, the data file size is increased in offloading matrix multiplication service component of the application with larger matrix size (as shown in Table 5.16). It is observed that the average goodput of network is 10295.9(+/-) 557.2 Kbps with 99.9% confidence in the sample space of 30 values for offloading matrix multiplication service at runtime.

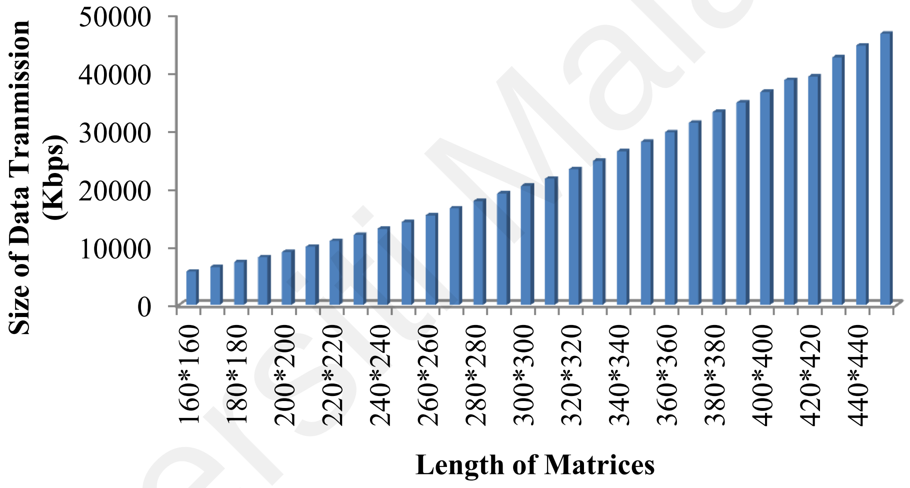

Figure 6. 15: Size of Data Transmission in Offloading Matrix Multiplication Service at Runtime

6.3 Analysis of DEAP Based Computational Offloading for MCC

This section discusses results of the application execution in the Primary Operating Procedure (POP) and Secondary Operating Procedure (SOP) procedures of DEAP framework. Table 5.17 summarized the TT of performing sorting operation in the POP of DEAP Client application. The TT of the sorting operation includes the execution time of performing the sorting operation on the DEAP server application and the time taken in saving the resultant preferences (data file) on the local mobile device.

The TT of sorting service is evaluated with 30 different computational intensities of the sorting operation (sort list size of 11000-40000). It is observed that the TT of the sorting operation varies with the computational intensity of sorting operation. For instance, the TT in sorting a list of 11000 values is determined as 2559(+/-)210 ms with 99% confidence interval in the sample space of 30 values, which shows that the possible range of TT is between 2349 ms and 2769 ms. The value of SD for TT shows the variation in the values of same sample space for each experiment. For instance, the variation of TT is 446 ms for sorting a list of 11000 values in 30 different experiments which constitutes 17.4 percent of the average TT. Similarly, the TT for sorting list of 25000 values in the sample space of 30 values is determined as 6770(+/-)157 ms with 99% confidence interval, which shows that the possible range of TT is between 6613 ms and 6927 ms. The value of SD for TT shows 334 ms variation for sorting a list of 25000 values in 30 different experiments which constitutes 4.9 percent of the average TT value.

Figure 6.16 shows the increase in TT of the sorting operation in the POP of DEAP client application in 30 different experiments. The TT for the sorting service varies according to the length of the list being sorted. The value of TT includes the processing time for performing sorting operation and the time taken in saving the resultant preferences file on the local mobile device. It is observed that by increasing the length of sorting list, the processing time of performing sorting operation on DEAP server and the time taken in saving preferences file increases accordingly. For instance, sorting the list of 15000 values takes on the average 3494 ms, whereas sorting the list of 40000 values takes 13416 ms in the POP of DEAP client application. Experimental results indicate that the TT in sorting the list of 40000 values increases 80.92 percent as compared to sorting the list of 11000 values in the POP of DEAP client application. However, the average TT for the sorting service in the POP of DEAP client is determined as 7224 ms for sorting list of 11000-40000 with the 45.8 percent RSD.

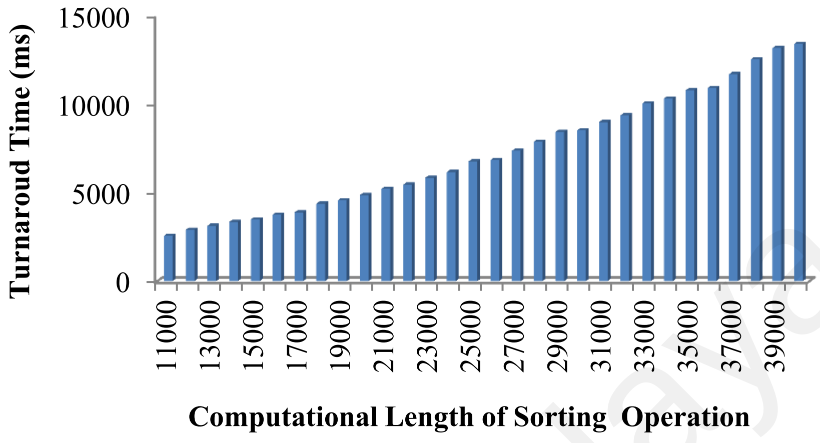

Figure 6. 16: Total Turnaround Time of Sorting Service in the POP of DEAP Client Application

Table 5.18 summarized the total TT of performing matrix multiplication operation in the POP of DEAP client application. The TT of the matrix multiplication includes the turnaround time of performing the sorting operation on the DEAP server application and the time taken in saving the resultant preferences (data file) on the local mobile device. The TT of matrix multiplication service is evaluated with 30 different computational intensities of the multiplication operation (matrices length 160*160-450*450).

It is found that the TT of the matrix multiplication service varies with the computational intensity of multiplying matrices length. For instance, the TT in multiplying matrices of length 160*160 is determined as 4241(+/-)98 ms with 99% confidence interval in the sample space of 30 values, which shows that the possible range of TT is between 4143 ms and 4339 ms. The value of SD for TT shows the variation in the values of same sample space for each experiment. For instance, the variation of TT is 207 ms for multiplying matrix of size 160*160 in 30 different experiments which constitutes 4.9 percent of the average TT. Similarly, the TT in multiplying matrices of length 450*450 is determined as 97887 ms with 99% confidence interval in the sample space of 30 values, which shows that the possible range of TT is between 92705 ms and 103069 ms. The variation of TT in the sample space of 30 values is 11002 ms for multiplying matrices of length 450*450 which constitutes 11.2 percent of the average TT.

Figure 6.17 shows the increase in TT of the matrix multiplication operation in the POP of DEAP Client application in 30 different experiments. The TT for the matrix multiplication service varies according to the computational length of matrix multiplication operation. The value of TT includes the processing time for performing matrix multiplication operation and the time taken in saving the resultant preferences file on the local mobile device.

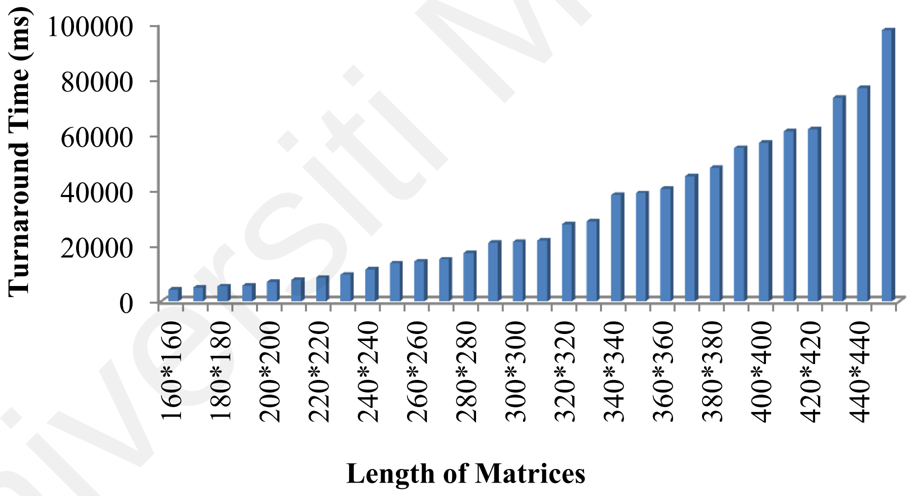

Figure 6. 17: Turnaround Time of Matrix Multiplication Service in POP of DEAP Client Application

It is observed that by increasing the length of the matrix, the processing time of performing multiplication operation on DEAP Server and the time taken in saving preferences file increases accordingly. For instance, the total TT for multiplying matrices of size 200*200 8560 ms, whereas the total TT for multiplying matrices of size 450*450 is 97887 ms in the POP of DEAP Client application. Experimental results indicate that the TT in multiplying matrices of length 450*450 increases 95.6 percent as compared to multiplying matrices of length 160*160 in the POP of DEAP client application. However, the average TT for the matrix multiplication operation in the POP of DEAP client is determined as 31445 ms for matrices of length 160*160-450*450 with the 80.8 percent RSD.

The total energy consumption cost (ECC) of the sorting operation in the POP of DEAP client application includes the energy consumed in accessing the sorting operation service of the DEAP server application, energy consumed in receiving the resultant sorted list from the remote server and energy consumed in saving the resultant preferences (data file) on the local mobile device.

Table 5.19 presented the total ECC of accessing sorting operation in POP of the DEAP client application. The ECC of sorting service is evaluated with 30 different computational intensities of the sorting operation (sort list size of 11000-40000). It is observed that the ECC of accessing the sorting service varies with the computational intensity of sorting list. For instance, the ECC in sorting a list of 11000 values is determined as 7.4(+/-).6 J with 99% confidence interval in the sample space of 30 values, which shows that the possible range of ECC is between 6.8 J and 8 J. The value of SD for ECC shows the variation in the values of same sample space for each experiment. For instance, the variation of ECC in 30 different experiments is 1.3 J for sorting a list of 11000 values which constitutes 17.6 percent of the average ECC. Similarly, the ECC in sorting a list of 40000 values is determined as 23(+/-)2.6 J with 99% confidence interval in the sample space of 30 values, which shows that the possible range of ECC is between 20.4 J and 25.6 J. The variation of ECC in 30 different experiments is 5.6 J for sorting the list of 40000 values which constitutes 24.3 percent of the average ECC.

Figure 6.18 shows the increase in ECC of the sorting operation with respect to the length of sorting list in the POP of DEAP client application in 30 different experiments. The ECC for the sorting service varies according to the length of the list being sorted. It is observed that by increasing the length of sorting list, the energy consumed in accessing sorting operation on DEAP server, energy consumed in returning resultant list and the energy consumed in saving preferences file locally on SMD increases accordingly. For instance in the POP of DEAP client application, ECC in accessing the sorting service for sorting the list of 15000 values 9.3 J, whereas for accessing the sorting service in sorting the list of 40000 23 J energy is consumed. Experimental results indicate that the ECC in sorting the list of 40000 values increases 67.8 percent as compared to sorting the list of 11000 values in the POP of DEAP client application. However, the average ECC for the sorting service in the POP of DEAP client is determined as 13.9 J for sorting list of 11000-40000 with the 27.9 percent RSD.

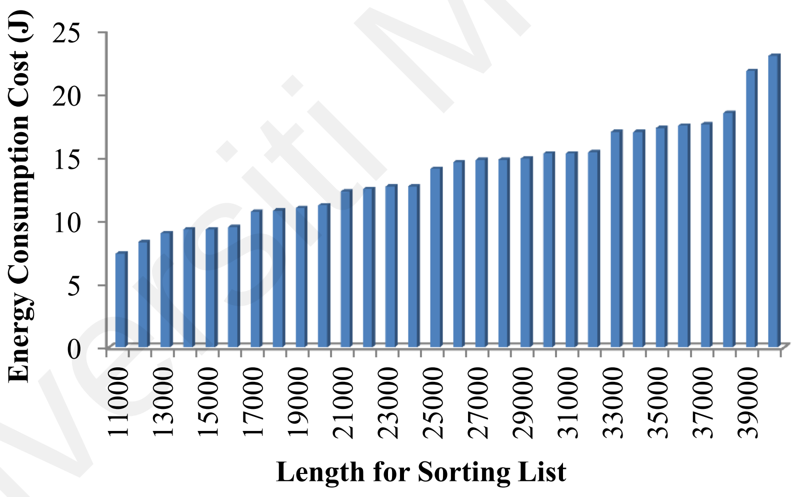

Figure 6. 18: Total Energy Consumption Cost of Sorting Service in POP of DEAP Client Application

The ECC of the matrix multiplication operation in the POP of DEAP client application includes the energy consumed in accessing the matrix multiplication service of the DEAP server application, energy consumed in receiving the resultant data from the remote server and energy consumed in saving the resultant preferences (data file) on the local mobile device. Table 5.20 presented the total ECC of accessing matrix multiplication service in POP of the DEAP Client application. The ECC of matrix multiplication service is evaluated with 30 different computational intensities of the matrix multiplication operation (matrices of size 160*160-450*450).

It is examined that the ECC of accessing the matrix multiplication service varies with the computational intensity of matrices. For instance, the ECC in multiplying matrices of length 160*160 is determined as 10.8(+/-)1.8 J with 99% confidence interval in the sample space of 30 values, which shows that the possible range of ECC is between 9 J and 12.6 J. The value of SD for ECC shows the variation in the values of same sample space for each experiment. For instance, the variation of ECC in 30 different experiments is 3.8 J for accessing matrix multiplication service with the matrices of length 160*160, which constitutes 35.2 percent of the average ECC. Similarly, the ECC in multiplication matrices of length 450*450 is determined as 65.3(+/-)5.1 J with 99% confidence interval in the sample space of 30 values, which shows that the possible range of ECC is between 60.2 J and 70.4 J. The variation of ECC in 30 different experiments is 10.8 J for accessing matrix multiplication service with the matrices of length 450*450, which constitutes 16.5 percent of the average ECC.

Figure 6.19 shows the increase in ECC in 30 different experiments of the matrix multiplication operation with respect to the size of matrices being multiplied in the POP of DEAP client application. It is observed that by increasing the size of matrices, the energy consumption cost increases accordingly in accessing matrix multiplication operation, returning resultant matrix and saving preferences file locally on SMD. For instance, in the POP of DEAP client application, the total ECC in accessing the matrix multiplication service for multiplying matrices of length 310*310 is 25.6 J, whereas for accessing matrix multiplication service for multiplying matrices of length 450*450, 65.3 J energy is consumed. Experimental results indicate that in the POP of DEAP client application, the ECC in multiplying matrices of length 450*450 increases 83.4 percent as compared to multiplying matrices of length 160*160. However, the average ECC for the matrix multiplication service in the POP of DEAP client is determined as 30.3 J for multiplying matrices of size 160*160-450*450 with the 56.7 percent RSD.

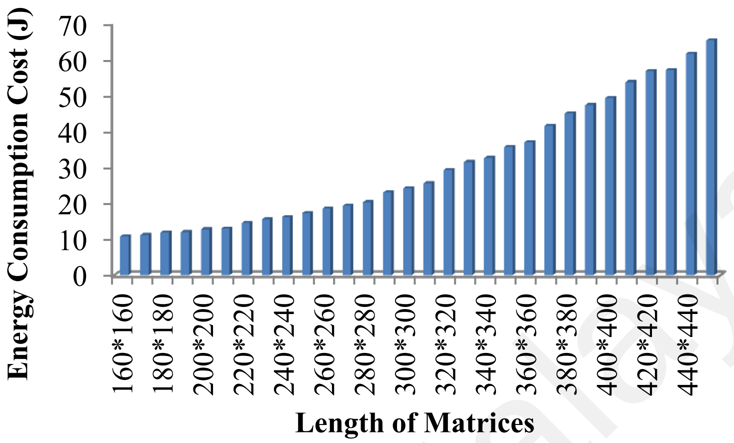

Figure 6. 19: Energy Consumption Cost of Matrix Multiplication Service in POP of DEAP Client Application

Power compute service of the application is offloaded at runtime in the SOP of DEAP client application. The TT of power compute service in the SOP includes time taken in transferring the binary file of the application, time taken in downloading the delegated application on the virtual device instance on the remote server node, time taken in the reconfiguration of delegated application service, and time taken in executing the service application and returning results to the local mobile device.

Table 5.21 summarized the time taken in runtime component offloading for 30 different experiments. Figure 6.20 shows time taken in different stages of offloading power compute service in the SOP of DEAP client application. The offloading time of power compute service is determined as 52(+/-)4 ms with 99% confidence interval in the sample space of 30 values, which shows that the possible range of offloading time for the power compute service is between 48 ms and 52 ms. The value of SD for offloading time of the service varies 9 ms in the sample space of 30 experiments, which constitutes 17.3 percent of the service offloading time in the SOP of DEAP client application.

The service download time to remote virtual device instance and reconfiguration time of the power compute service on remote machine is evaluated in 30 different experiments. It is examined that the download time to remote virtual device of the power compute service is 212(+/-)18 ms, which shows that the possible range of download time to remote virtual device on the server node is between 194 ms and 230 ms. The value of SD for service download time varies 9 ms in the sample space of 30 experiments, which constitutes 18.4 percent of the service download time to remote virtual device in the SOP of DEAP Client application.

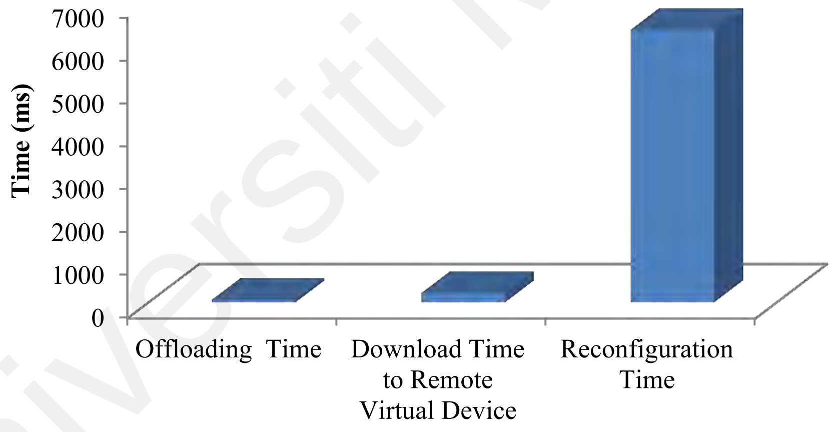

Figure 6. 20: Time Taken in Offloading Power Compute in the SOP of DEAP Client Application

Similarly, the reconfiguration time of the power compute service on the remote server node is determined 6349(+/-)312 ms, which shows that the possible range of download time to remote virtual device on the server node is between 6037 ms and 6661 ms. The variation in the reconfiguration time value of power compute service is 663 ms in the sample space of 30 experiments, which constitutes 10.4 percent of the reconfiguration time on remote virtual device in the SOP of DEAP client application.

Table 5.22 summarized the TT of the execution of power compute service component of the application in the SOP of DEAP client application. The TT of the power compute application includes the total time taken in runtime component offloading and execution time of the application on the remote server node.

The total TT in offloaded processing of the power compute service component of the mobile application is evaluated with 30 different computational intensities of the power computing (2^1000000-2^2000000000). For instance, the TT in computing 2^1000000 is determined as 7175(+/-)340 ms with 99% confidence interval in the sample space of 30 values, which shows that the possible range of TT is between 6835 ms and 7515 ms. The value of SD for TT shows the variation in the values of same sample space in 30 different experiments. For instance, the variation of TT is 721 ms for computing 2^1000000, which constitutes 10.1 percent of the average TT of compute service in the SOP of DEAP client application. Similarly, the TT in computing 2^2000000000 is determined as 265724(+/-) 2485 ms with 99% confidence interval in the sample space of 30 values, which shows that the possible range of TT is between 263239 ms and 268209 ms. The SD in TT is 5275 ms for computing 2^2000000000, which constitutes 2 percent of the average TT of compute service in SOP of DEAP client application.

Figure 6.21 shows the increase in TT of the power compute service with respect to the computational length of the computing operation in 30 different experiments. It is examined that by varying the computational length of the power compute service, the average total time taken in runtime offloading remains constant. However, the execution time of the power computing on remote machine increases by increasing the computational intensity of the power compute service. For instance, the average time taken in runtime offloading and reconfiguration is 6613 ms for all the instances of offloading power compute service in the SOP of DEAP Client application. Whereas, the TT of computing 2^2000000 is determined as 7587 and the TT of computing 2^2000000000 is determined as 265724 ms. Results indicate that TT increases 97.2 percent in computing 2^2000000000 as compared to computing 2^1000000.

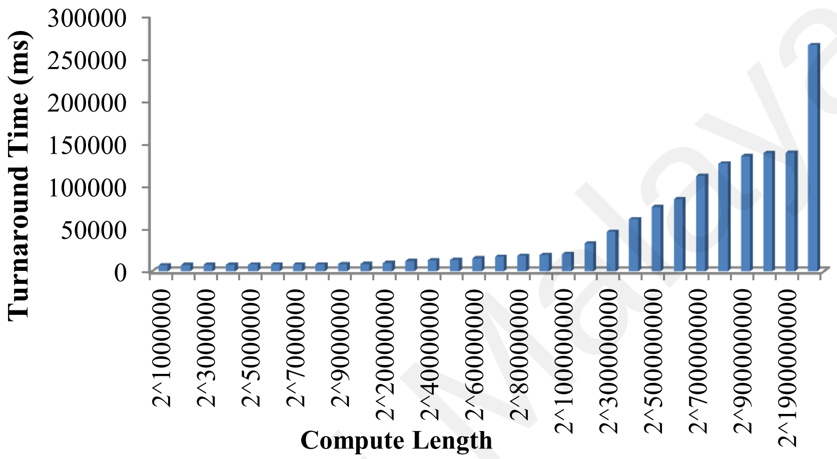

Figure 6. 21: Turnaround Time of Power Compute Service in the SOP of DEAP Client Application

Table 5.23 summarized the total energy consumed in offloading power compute service in the SOP of DEAP client application. The total ECC is evaluated with 30 different computational intensities of the power compute operation (2^1000000-2^2000000000). The total ECC of the power compute application includes the energy consumed in runtime component offloading and energy consumed in remote application processing. The ECC for computing 2^1000000 on remote server node is determined as 5.4(+/-).7 J with 99% confidence interval in the sample space of 30 values, which shows that the possible range of total ECC is between 4.7 J and 6.1 J. The value of SD for total ECC shows the variation in the values of same sample space for each experiment in offloaded processing. For instance, the variation of total ECC is 1.4 J for remote processing of computing 2^1000000 which constitutes 25.9 percent of the average ECC in offloaded processing of the power compute service in the SOP of DEAP client application.

The ECC for computing 2^2000000000 on remote server node is determined as 351(+/-)4.5 J with 99% confidence interval in the sample space of 30 values, which shows that the possible range of total ECC is between 343.6 J and 58.4 J. The variation of total ECC is 15.7 J for remote processing of computing 2^2000000000, which constitutes 4.5 percent of the average ECC in offloaded processing of the power compute service in the SOP of DEAP client application. Figure 6.22 shows the total ECC of power compute service in the SOP of DEAP client application.

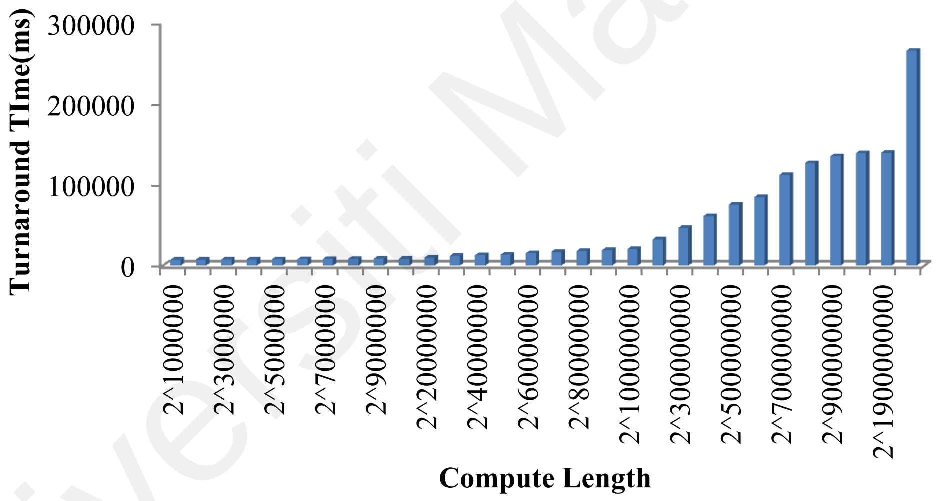

Figure 6. 22: Energy Consumption Cost of Power Compute Service in SOP of DEAP Client Application

It is observed that the value of energy consumption in component offloading increases steadily (the average value is 4.7 J); however the value of ECC in performing compute operation increases quickly by increasing the computational length. For instance, 10 J energy is consumed in computing 2^9000000, whereas 351 J energy is consumed in computing 2^2000000000 in SOP of the DEAP Client application. It shows that the total

ECC is increased 98.5 percent for computing 2^2000000000 as compared to 2^1000000 in offloaded processing of Power compute service in the SOP of DEAP client application.

Table 5.24 summarized the increase in RAM allocation on local mobile device for accessing the sorting service in the POP of DEAP Client application. The increase in the allocated RAM for sorting a list of 11000 values is determined as 1.165(+/-)0.028 MB with 99% confidence interval in the sample space of 30 values, which shows that the possible range of increase in RAM allocation is between 1.137 MB and 1.193MB. The increase in allocation of RAM for DEAP client application varies 0.06 MB for accessing sorting service with the sorting the list 11000 values in 30 different experiments. It shows that the increase in allocation of RAM on local mobile device varies 5.2 percent in accessing sorting service of DEAP Server application.

The increase in allocated RAM for sorting a list of 25000 values in the sample space of 30 values is determined as 2.382(+/-)0.02 MB with 99% confidence interval, which shows that the possible range of increase in RAM allocation is between 2.362 MB and 2.402 MB. Figure 6.23 shows the increase in allocation of RAM to DEAP client application in accessing sorting service of DEAP server application. The change in the allocation of RAM on mobile device for DEAP client application is evaluated in 30 different experiments. The allocation of RAM to DEAP client application in accessing the sorting service on DEAP server application varies according to the length of the resultant sorted list returned to local mobile device.

It is examined that in returning the sorted list of 15000 values the average RAM allocation increases 1.6 MB, whereas in returning the list of 40000 values the average RAM allocation increases 5.3 MB for DEAP client application on local mobile device. Analysis of the results shows that in the process of saving the resultant preferences file on local device, the heap size and allocated RAM allocation for DEAP client application increases as per the length of resultant values returned from DEAP Server application.

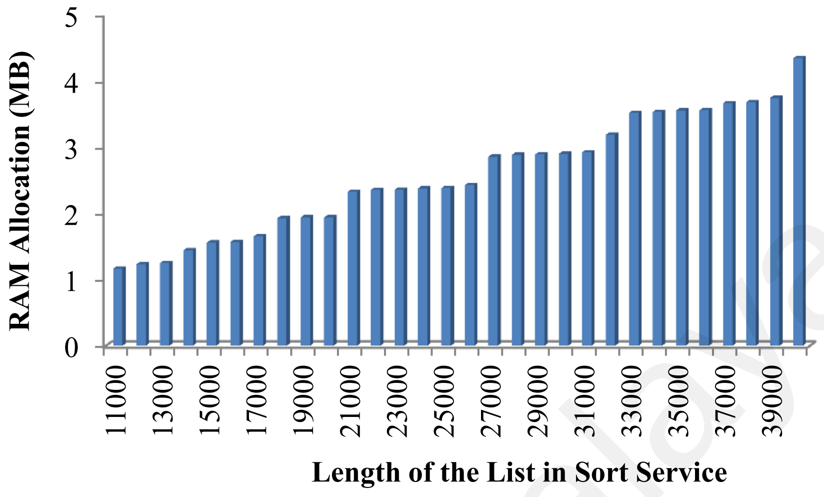

Figure 6. 23: Increase in the RAM Allocation to DEAP Client Application for Accessing Sorting Service in POP

Table 5.25 summarized the increase in RAM allocation on local mobile device for accessing the matrix multiplication service in the POP of DEAP client application. The increase in allocated RAM for DEAP client application on local mobile device in accessing matrix multiplications service for multiplying matrices of size 160*160 is determined as 1.695(+/-).08 MB with 99% confidence interval in the sample space of 30 values, which shows that the possible range of increase in RAM allocation is between 1.615 MB and 1.775 MB. The increase in allocation of RAM for DEAP client application varies 0.16 MB for accessing matrix multiplication service with the matrices size 160*160, which shows that the increase in allocation of RAM on local mobile device varies 9.4 percent in accessing matrix multiplication service of DEAP server application.

The increase in allocated RAM for DEAP client application on local mobile device in accessing matrix multiplications service for multiplying matrices of length 450*450 is determined as 13.056(+/-)0.31 MB with 99% confidence interval in the sample space of 30 values, which shows that the possible range of increase in RAM allocation is between 12.746 MB and 13.366 MB. The increase in allocation of RAM for DEAP client application varies 0.653 MB for accessing matrix multiplication service with the matrices of length 450*450, which shows that the increase in allocation of RAM on local mobile device varies 5 percent in accessing matrix multiplication service of DEAP server application.

Figure 6.24 shows the increase in allocation of RAM to DEAP client application in accessing matrix multiplication of DEAP server application. The change in the allocation of RAM on mobile device for DEAP client application is evaluated in 30 different experiments. The allocation of RAM to DEAP client application in accessing the sorting service on DEAP Server application varies according to the length of the resultant matrix length returned to local mobile device .

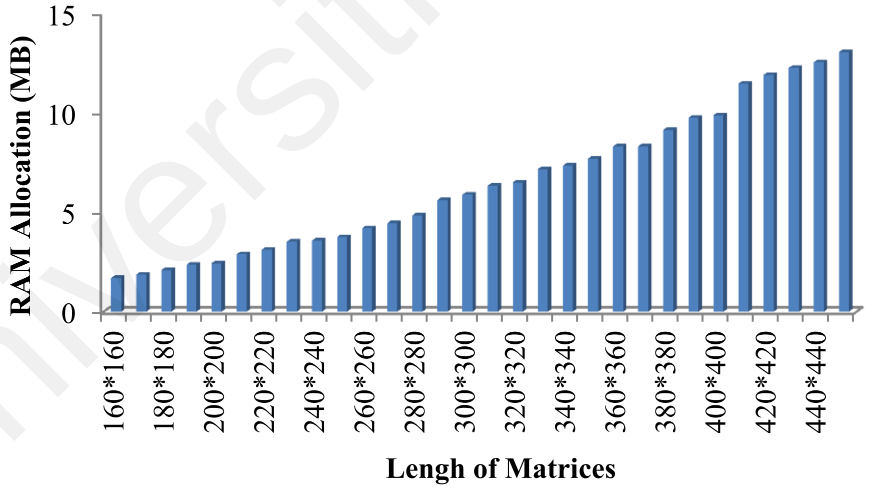

Figure 6. 24: Increase in the RAM Allocation to DEAP Client Application for Accessing Matrix Multiplication Service in POP

It is examined that in returning matrices of length 200*200 values the average RAM allocation increases 2.438 MB, whereas in returning matrices of length 450*450 values the average RAM allocation increases 5.3 MB for DEAP client application on local mobile device. Analysis of the results shows that in the process of saving the resultant preferences file on local device, the heap size and allocated RAM allocation for DEAP client application increases as per the length of resultant values returned from DEAP server application.

Table 5.26 summarized the average CPU utilization on mobile device for DEAP client application in offloaded processing of different components of the mobile application. DEAP employs POP and SOP for computational offloading to cloud server node; hence, the CPU utilization of DEAP client application on local mobile device depends majorly on the size of resultant data returned from the remote server node.

Analysis of the results for CPU utilization in the POP and SOP of DEAP framework indicates that minimal percentage (2-3%) of the CPU is utilized in offloaded processing of the components of the mobile application. However, the CPU utilization increases with increase in data size received as a result of remote processing. For instance, the CPU utilization for accessing all the three service components with varying computational intensities (sorting service (11000-40000), matrix multiplication service (160*160-450*450), and power compute service (2^1000000-2^2000000000)) remains constant. However, the utilization of CPU on the local device increases while processing the resultant data received from the remote server node.

Figure 6.25 shows the average CPU utilization in the POP and SOP of DEAP client application. The CPU utilization for accessing the sorting service is evaluated with 30 different computational intensities (list size 11000-40000). DEAP client utilizes 25.5 percent (609(+/-)134 MIPS) of the CPU, which shows the range of CPU utilization for accessing sorting service in the POP of DEAP client application between 19.9 percent (474.4 MIPS) and 31.1 percent (744 MIPS) on the local SMD. Relative standard deviation in CPU utilization for accessing the sorting service in the POP DEAP Client application is 54.91 percent of the average CPU utilization on SMD.

The CPU utilization for accessing the matrix multiplication service is evaluated with 30 different computational intensities (matrices size 160*160-450*450). It is examined that DEAP client utilizes 35.4 percent (845.7(+/-)200MIPS) of the CPU, which shows the range of CPU utilization for accessing matrix multiplication service in the POP of DEAP client application is between 27 percent (645.4 MIPS) and 43.7 percent (1046 MIPS) on the local SMD. Relative standard deviation in CPU utilization for accessing the sorting service in the POP of DEAP client application is 50.3 percent of the average CPU utilization on SMD.

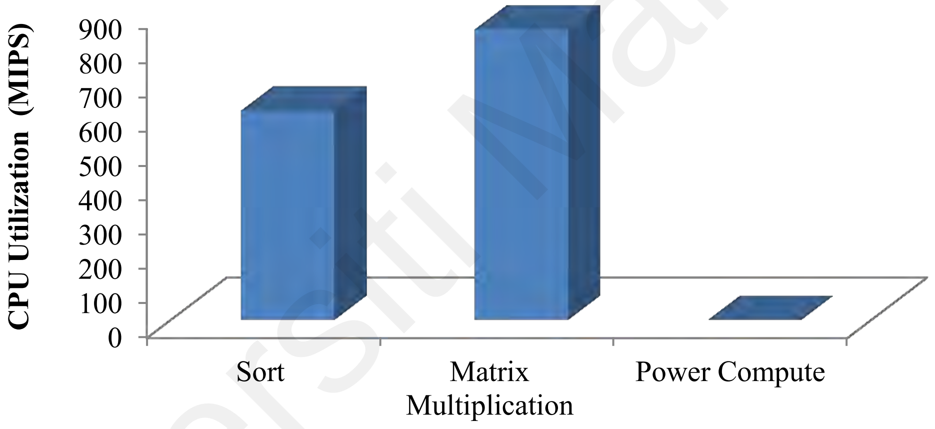

Figure 6. 25: CPU Utilization for DEAP Client Application on Local Mobile Device in POP and SOP

The power compute component of the application is offloaded at runtime in SOP of DEAP client application. Hence, the average CPU utilization by DEAP client application in offloading power compute service is determined as 3(+/-)0.38 percent (71.6(+/-)9 MIPS), which shows that the range of CPU utilization for computing Power compute operation in the SOP of DEAP Client application is 2.62 percent (62.5 MIPS) and 3.38 percent (80.75 MIPS) on the local mobile device. Relative standard deviation in CPU utilization for performing power compute operation in the SOP of DEAP Client application is 26.7 percent of the average CPU utilization on SMD.

Figure 6.26 shows the increase in data transmission over the wireless network medium in accessing sort service on the DEAP Server application with respect to the varying size of sorting list. Analysis of the results for the data transmission over the wireless network medium for sorting service indicates that in the POP of DEAP framework, the data transmission over the wireless network medium involves the resultant values of the sorted list. Hence, the large list of values in the sorting operation results in returning a larger size of data. However, the data transmission cost of application binary offloading is eliminated in the POP of DEAP Client application.

It is observed that sorting the list of 11000 values on DEAP server returns 123 KB data to DEAP client application, whereas sorting the list of 40000 values on DEAP server returns 692 KB of data to DEAP client application.

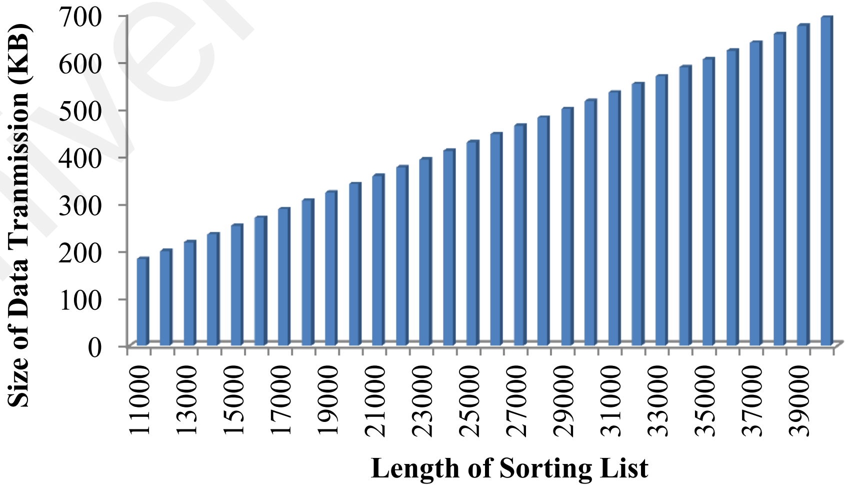

Figure 6. 26: Size of Data Transmission in Accessing Sorting Service of DEAP Server Application

Figure 6.27 shows the increase in data transmission over the wireless network medium in accessing matrix multiplication service on the DEAP server application with respect to the varying size of matrices.

Analysis of the results for the size of data transmission over the wireless network medium for matrix multiplication service indicates that in the POP of DEAP framework, the data transmission over the wireless network medium involves the resultant values of the matrix multiplication operation. Hence, larger size of matrices in matrix multiplication results in returning a larger size of data. For instance, multiplying matrices of size 160*160 returns 463 KB of data to the DEAP client application, whereas multiplying matrices of size 450*450 returns 3308 KB of data to the DEAP client application.

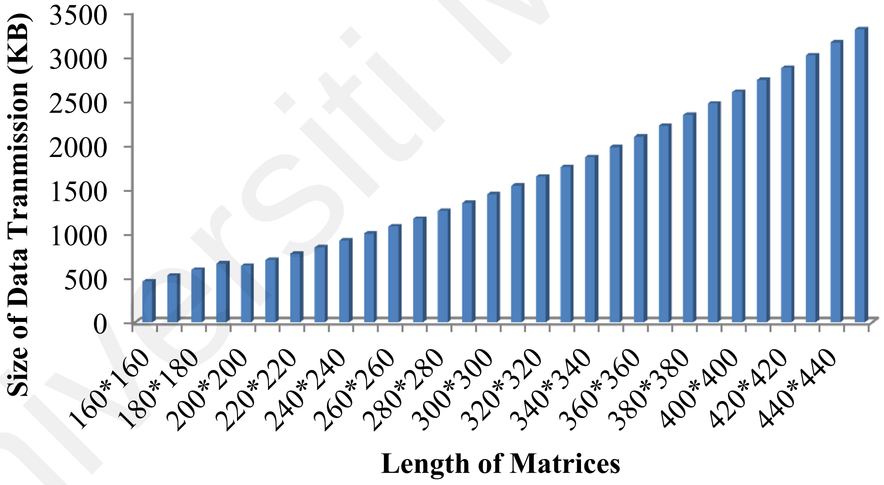

Figure 6. 27: Size of Data Transmission in Accessing Matrix Multiplication Service of DEAP Server Application

The power compute service component is offloaded at runtime by using the SOP of DEAP client application. Therefore, the binary file of the service component is transmitted over the wireless network medium at runtime. The total data transmission size of the power compute service offloading is 42.7 KB with the network goodput ratio 65.96 Kbps.

6.4 Comparison of Experimental Results

This section analyzes the comparison of experimental results in different scenarios. The usefulness of the proposed solution is verified by comparing experimental results from the following perspectives. 1) Application execution on local mobile device and traditional application offloading. 2) Application execution by including runtime application profiling and excluding runtime application profiling technique. 3) Application execution on local mobile device and the operating procedures of DEAP proposed framework. 4) Traditional computational offloading and DEAP based computational offloading.

Initially, the prototype application is tested on the Android Virtual Device (AVD) in the emulation environment to test the viability of the proposed framework. Components of the mobile application are executed on the local AVD and remote DEAP server by employing the emulator. The POP of DEAP client application is implemented over the AVD in the emulation environment. Table 5.28 summarized the statistics of turnaround time and energy consumption cost of executing sorting service on local AVD and DEAP server application by using emulator.

Figure 6.28 shows the comparison of TT for sorting service execution on the local Android Virtual Device (AVD) and in the POP of DEAP client application by using emulator. It is examined that the TT of the sorting services reduces significantly in the POP of DEAP client application. Experimental results indicate that the TT of sorting operation in the POP of DEAP reduces 75.8 percent for sort the list of 11000 values, 79.4 percent for sorting list 15000, 84.4 percent for sorting list 25000, 88.9 percent for sorting list of 340000 values and 89.4 percent for sorting list 40000.

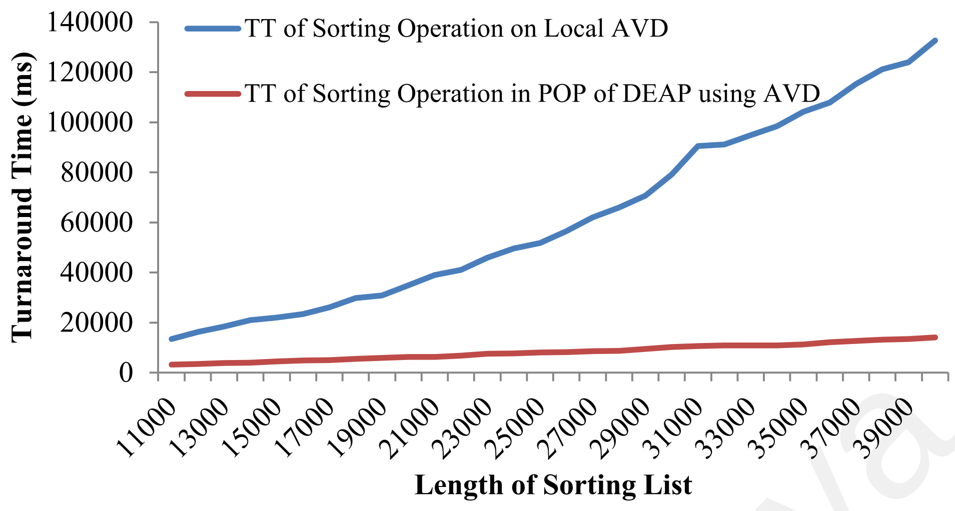

Figure 6. 28: Comparison of Turnaround Time of Sorting Operation on Local AVD and POP of DEAP

Figure 6.29 shows the comparison of the ECC of sorting service execution on the local Android Virtual Device (AVD) and in the POP of DEAP client application by using emulator (as shown in table 5.28). The ECC of the sorting service reduces significantly in the POP of DEAP client application. Experimental results indicate that the ECC of sorting operation in the POP of DEAP reduces 33.8 percent for sort the list of 11000 values, 46 percent for sorting list 15000, 68.8 percent for sorting list 25000, 79.8 percent for sorting list of 340000 values and 79.2 percent for sorting list 40000.

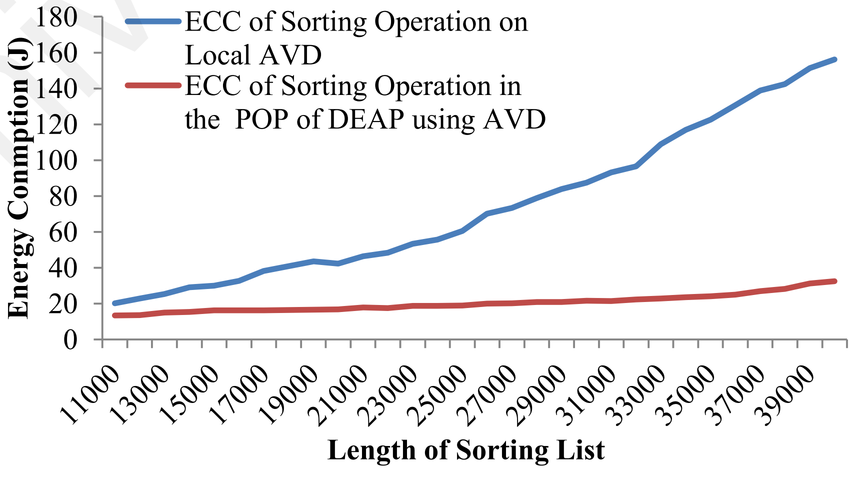

Figure 6. 29: Comparison of Energy Consumption Cost of Sorting Operation on Local AVD and POP of DEAP

The reduction in the TT and ECC of the sorting service is for the reason of executing the intensive sorting operation on the remote server node which has higher computing potentials as compared to the local virtual device. Analysis of the empirical results for the sample space of 30 computational intensities with 1800 experiments conducted (both on local AVD and DEAP Server) for sorting operation indicates that in the POP of DEAP client application, battery consumption is reduced 66.2(+/-)6.5 percent with 99% confidence for the sample space of 30 values and utilization of the computing resources (CPU, RAM) of the local device is reduced for minimum period of time 85(+/-)1.8 percent with 99% confidence for the sample space of 30 values.

Table 5.29 summarized the comparison of execution time and energy consumption in the execution of matrix multiplication operation on local virtual device instance and DEAP based service execution by using AVD emulator. Figure 6.30 shows the comparison the TT of matrix multiplication service execution on the local AVD and remote DEAP server application by using emulator. Experimental results indicate that the TT of matrix multiplication is greater (1.3 percent) in the POP of DEAP Client application as compared to the execution of the matrix service on local AVD. It indicates the unfeasibility of accessing the services of DEAP server for low intensive tasks of mobile application. However, the turnaround time of the matrix multiplication service with high computational length reduces significantly in the POP of DEAP client application.

It is observed that reduction in the turnaround time of the matrix multiplication services increases gradually with the increase in computational intensity of the matrix multiplication operation. Experimental results indicate that the turnaround time of matrix multiplication operation in the POP of DEAP reduces 10.9 percent for multiplying matrices of 170*170 size, 31.4 percent for multiplying matrices of 250*250 size, 62.6 percent for multiplying matrices of 350*350 size and 76.9 percent for multiplying matrices of 450*450 size which shows increase in reduction of turnaround time of the matrix multiplication service with the increase of matrices size. The reduction in the turnaround time of the matrix multiplication service is for the reason of executing the intensive multiplication operation on the remote server node which has higher computing potentials as compared to the local virtual device.

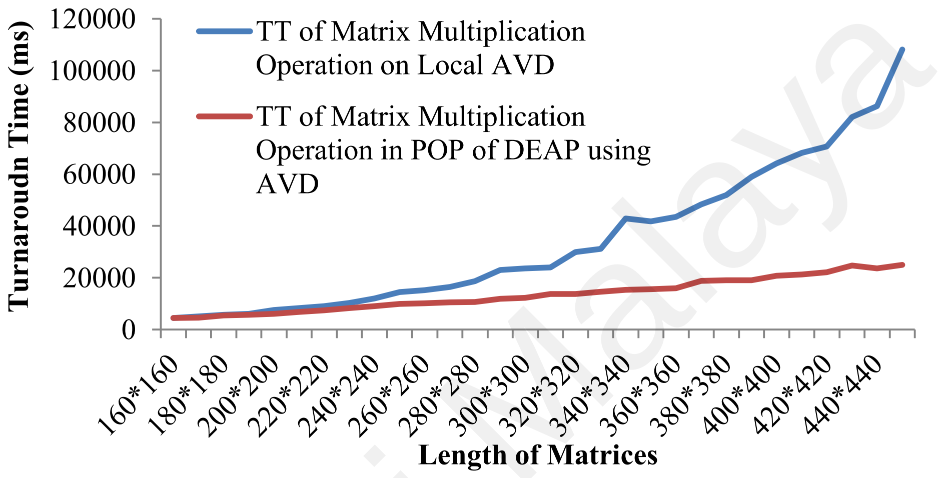

Figure 6. 30: Comparison of Turnaround Time of Matrix Multiplication Operation on Local AVD and POP of DEAP

Figure 6.31 shows the comparison the ECC of matrix multiplication service execution on the local AVD and remote DEAP server application by using emulator. Experimental results indicate that for smaller computational intensity of matrix multiplication operation, the ECC of matrix multiplication is greater in the POP of DEAP client application as compared to the execution of the matrix service on local AVD. For instance in the POP of DEAP, ECC is greater 5.4 percent for matrices of length 160*160, 12.5 percent for matrices of length 170*170, 10.9 percent for matrices of length 180*180, 10 percent for matrices of length 190*190, 5.1 percent for matrices of length 200*200, and 2.7 percent for matrices of length 210*210. It indicates the unfeasibility of accessing the services of DEAP server for low intensive tasks of mobile application.

However, the ECC of the matrix multiplication service with high computational intensity reduces significantly in the POP of DEAP client application. It is observed that reduction in the ECC of the matrix multiplication services increases gradually with the increase in computational intensity of the matrix multiplication operation. Experimental results indicate that the ECC of matrix multiplication operation in the POP of DEAP reduces 10.3 percent for multiplying matrices of 220*220 size, 28.5 percent for multiplying matrices of 270*270 size, 53.1 percent for multiplying matrices of 350*350 size and 60.8 percent for multiplying matrices of 400*400 size which shows increase in reduction of ECC of the matrix multiplication service with the increase of matrices size.

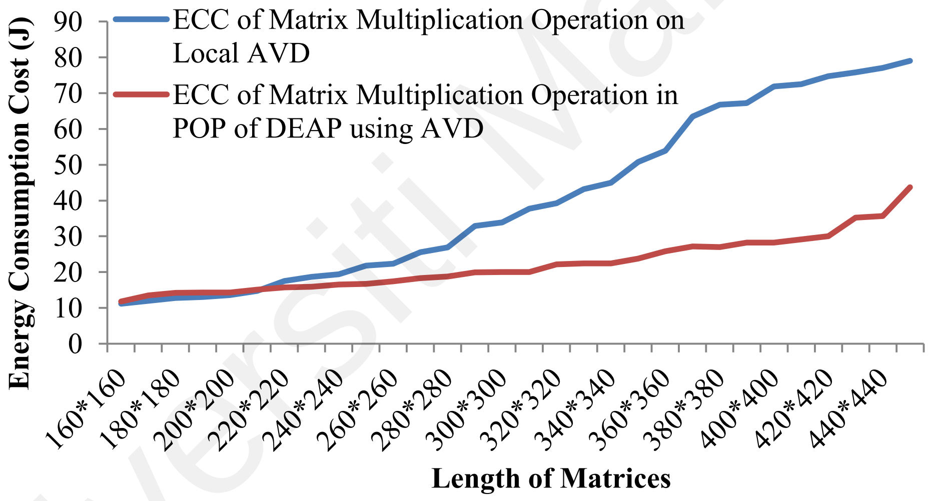

Figure 6. 31: Comparison of Energy Consumption Cost of Matrix Multiplication Service on Local AVD and POP of DEAP

The reduction in the ECC of the matrix multiplication service is for the reason of executing the intensive multiplication operation on the remote server node which has higher computing potentials as compared to the local virtual device. Analysis of the empirical results for the sample space of 30 computational intensities with 1800 experiments conducted (both on local AVD and DEAP server) for matrix multiplication indicates that in the POP of DEAP client application; battery consumption is reduced (interval estimate

32.2(+/-)11.8 percent with 99% confidence for the sample space of 30 values) and utilization of computing resources (RAM and CPU) of the local device is reduced for minimum period of time (interval estimate 43.79(+/-)11.2 percent with 99% confidence for the sample space of 30 values).

Table 5.30 summarized the comparison of TT and energy consumption in the execution of power compute service on local AVD and remote DEAP server application by using Android emulator. Figure 6.32 shows the comparison of the TT of power compute service execution on the local AVD and in the POP of DEAP client application by using emulator. The TT of the power compute service reduces significantly in the POP of DEAP client application.

It is observed that reduction in the TT of the power compute multiplication services increases gradually with the increase in computational intensity of the power compute operation. The TT of power compute operation in the POP of DEAP reduces 69.2 percent for computing 2^1000000, 85.4 percent for computing 2^7000000, 93.8 percent for computing 2^40000000 and 96.3 percent for computing 2^2000000000, which shows increase in reduction of turnaround time of the power computing operation with the increase in computational length. The overall reduction in the TT of power computing in DEAP client application is 91 percent, with the 7 percent RSD. The reduction in TT of the power compute service is for the reason of leveraging the processing services of high computing potential remote server node.

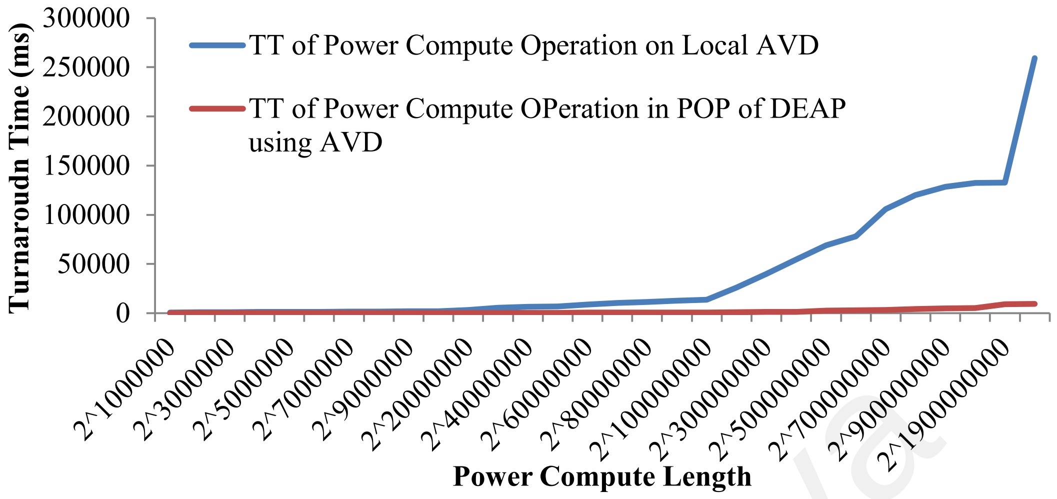

Figure 6. 32: Comparison of Turnaround Time of Power Compute Operation on Local AVD and POP of DEAP

Figure 6.32 shows the comparison the ECC of power compute service execution on the local AVD and in the POP of DEAP Client application by using emulator. The ECC of the power compute service reduces significantly in the POP of DEAP client application. It is observed that reduction in the ECC of the power compute multiplication services increases gradually with the increase in computational intensity of the power compute operation. Experimental results indicate that the ECC of power compute operation in the POP of DEAP reduces 6.5 percent for computing 2^1000000, 28.1 percent for computing 2^7000000, 60.9 percent for computing 2^40000000 and 95.7 percent for computing 2^2000000000, which shows increase in reduction of ECC of the power computing operation with the increase in computational length. The overall reduction in the ECC of power computing in DEAP client application is 52.24 percent, with the 62.9 percent RSD.

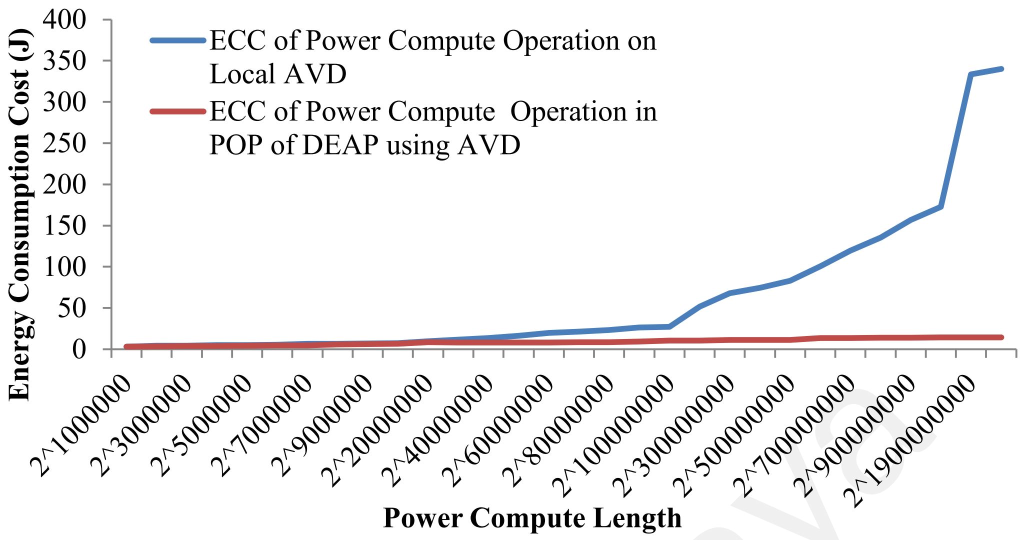

Figure 6. 33: Comparison of Energy Consumption Cost of Power Compute Operation on Local AVD and POP of DEAP

The empirical results for the sample space of 30 computational intensities with 1800 experiments conducted (both on local AVD and DEAP Server) for power compute indicates that in the POP of DEAP Client application; battery consumption is reduced (interval estimate 52.2(+/-)15 percent with 99% confidence for the sample space of 30 values) and the utilization of computing resources of the local device is reduced for minimum period of time (interval estimate 91(+/-)3 percent with 99% confidence for the sample space of 30 values). Hence, the experimental results in the emulation environment signify the usefulness of DEAP framework for computational offloading in mobile cloud computing.

This section compares the Turnaround Time (TT) and Energy Consumption Cost (ECC) of the intensive operations of mobile application in different scenarios of the real mobile cloud computing environment. The TT and EEC of the components of the mobile application are compared from the perspective of local execution of the intensive component of the application, and execution of the intensive component on the remote sever node. Local execution of the services is presented from the perspective of service execution without including profiling mechanism and with including profiling mechanism on the local mobile device. The contemporary application offloading frameworks implement runtime application profiling for the evaluation of resources utilization on SMD and making the decision of component offloading at runtime (Messer et al., 2002; Giurgiu et al., 2009; Cuervo et al., 2010; Chun et al., 2011; Zhang et al. 2011). Hence, the TT and ECC for each service component which is evaluated on the local mobile device is presented in two different scenarios; execution of service component without runtime profiling and execution of service component by activating the runtime profiling process.

The value of TT and ECC for different service components of the prototype application is compared with 30 different computational intensities. The objective of this comparison is to analyze the additional resources utilization and additional time taken in runtime profiling of mobile application.

Execution of the service component on the remote server node is presented from the perspective traditional runtime component offloading and DEAP based computational offloading in MCC. The traditional computational offloading frameworks implement runtime component migration techniques for outsourcing computational load of the mobile application (Liu et al. 2010; Iyer et al. 2011; Zao et al., 2011; Cuervo et al., 2010; Chun et al., 2011; Zhang et al. 2011 and Hung et al., 2012). The traditional computational offloading technique reduces the computational load on the mobile devices which results in minimization of computing resources utilization on SMD. However, it is examined that the size of data transmission, TT and ECC of the offloaded component increases considerably in runtime computational offloading.

The TT and ECC of sorting service and matrix multiplication component is compared by offloading without profiling process and including profiling mechanism on local mobile device. Finally, the TT and ECC of sorting service and matrix service are compared with the value of TT in DEAP client application. Table 5.31 summarized the comparison of the TT of sorting operating of the application in different scenarios of the real distributed mobile cloud computing environment.

Figure 6.34 shows the comparison the TT of sorting service execution in different scenarios. It is examined that the TT on the local SMD is smaller for performing sorting operation without runtime profiling. Experimental results indicate that by including the runtime profiling mechanism the value of TT increases 76.5 percent for sorting list of 11000 values, 86.6 percent for sorting list of 22000 values, 87.6 percent for sorting list of 30000 and 88.7 percent for sorting list of 40000 values. The overall increase in the TT of sorting service by including runtime profiling in the sorting operation is determined as 85.4(+/-)1.7 with 99% confidence in the sample space of 30 values.

The comparison of TT for sorting operation in local execution and traditional offloading technique shows that TT of the sorting service increases considerably in runtime component offloading. It is observed in offloading sorting service without employing runtime profiling on the local mobile device, the TT of the sorting service in remote processing compared to local execution of on mobile device increases by: 80 percent for sorting list of 11000 values, 75 percent for sorting list 17000 values, 80 percent for sorting list of 30000 values and 81 percent for sorting list of 40000 values. Similarly, in offloading sorting service by employing runtime profiling on the local mobile device, the TT of the sorting service in remote processing compared to local execution on mobile device increases by: 88 percent for sorting list of 11000 values, 91 percent for sorting list 25000 values, 92 percent for sorting list of 35000 values and 93 percent for sorting list of 40000 values. The comparison of sorting service execution on local mobile device and the DEAP based execution signifies the decrease in TT of the sorting operation in the POP of DEAP client application. It is examined that by accessing the services of DEAP server application in the POP of DEAP client application on mobile device, the TT of sorting services reduces 48 percent for sorting list of 11000 values, 60 percent for sorting list of 25000 values and 57 percent for sorting list of 40000 values. The overall reduction in TT value for sorting service in POP of DEAP client application is 57.8(+/-) 2 percent with 99% confidence in the sample space of 30 values.

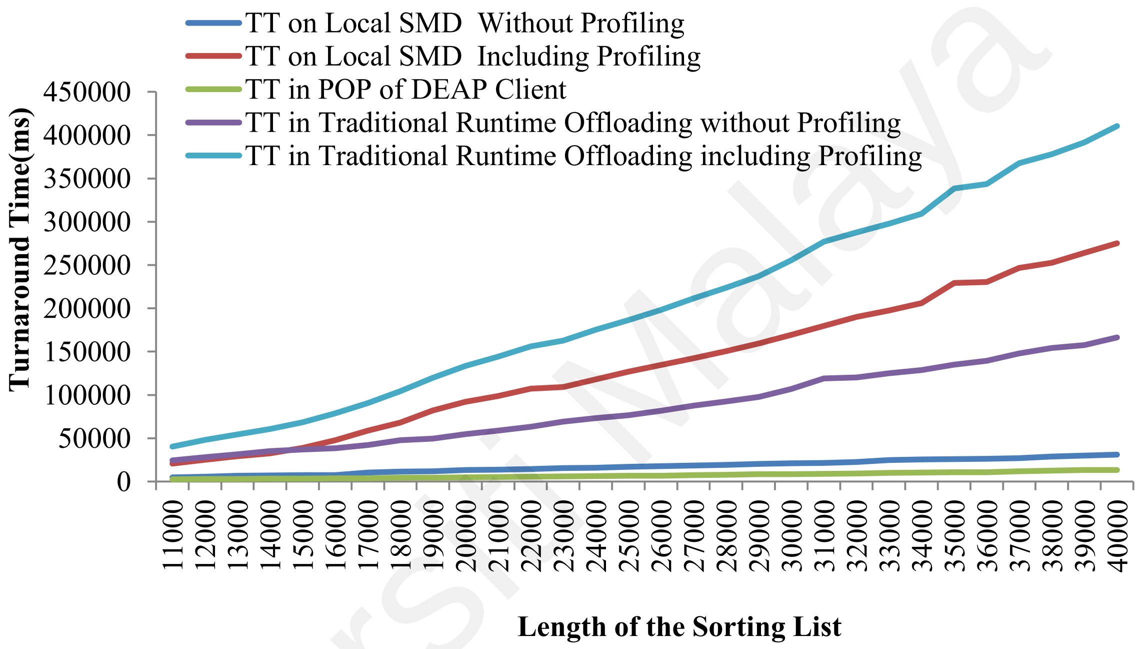

Figure 6. 34: Comparison of the Turnaround Time (TT) of the Sorting Service Execution in Local and Remote Execution

The comparison of TT for the sorting operation in the POP of DEAP and traditional offloading signifies the lightweight nature of DEAP framework for computational offloading in MCC. Figure 6.34 shows the increasing trend of TT in offloading sorting service. The increase in TT of sorting service in runtime component offloading (without including profiling) as compared to the POP of DEAP is examined as follows: 89 percent for sorting list of 11000 values, 91 percent for sorting list of 20000 values, 92 percent for sorting list of 31000 values and 92 percent for sorting list of 40000 values. Similarly, the increase in TT of sorting service in runtime component offloading by including profiling on

SMD as compared to the POP of DEAP as follows: 93 percent for sorting list of 11000 values, 96.4 percent for sorting list of 20000 values, 96.8 percent for sorting list of 31000 values and 96.7 percent for sorting list of 40000 values.

Table 5.32 summarized the comparison of matrix multiplication operation in local and remote execution scenarios. It is examined that the TT on the local SMD is smaller for performing matrix multiplication operation without runtime profiling as compared to matrix multiplication operation included with runtime profiling process. Figure 6.35 compares the TT of matrix multiplication operation in local and remote execution. Experimental results indicate that by including the runtime profiling mechanism the value of TT increases by 91.6 percent for multiplying matrices of length 160*160, 91.1 percent for matrices of length 250*250, 93 percent for multiplying matrices of length 350*350 and 92.7 percent for multiplying matrices of length 450*450.

The overall increase in the TT of matrix multiplication operation by including runtime profiling in the matrix multiplication operation is determined as 91.4(+/-)0.3 with 99% confidence in the sample space of 30 values. The comparison of TT for matrix multiplication operation in local execution and traditional offloading technique shows that TT of the matrix multiplication increases considerably in runtime component offloading. It is observed in offloading matrix multiplication service without employing runtime profiling on the local mobile device the TT of the matrix multiplication service in remote processing compared to local execution on mobile device increases by 78 percent for multiplying matrices of length 160*160, 70 percent for multiplying matrices of length 250*250, 66 percent for multiplying matrices of length 300*300 and 65 percent for multiplying matrices of length 450*450.

Similarly, in offloading matrix multiplication service with employing runtime profiling on the local mobile device, the TT of the matrix multiplication service in remote processing compared to local execution on mobile device increases by 81 percent for multiplying matrices of length 160*160, 76 percent for multiplying matrices of length 250*250, 75 percent for multiplying matrices of length 300*300 and 72 percent for multiplying matrices of length 450*450.

The comparison of matrix multiplication service execution on local mobile device and the DEAP based execution signifies the decrease in TT of matrix multiplication operation in the POP of DEAP client application. It is observed that by accessing the services of DEAP server application in the POP of DEAP client application on mobile device, the TT of matrix multiplication operation reduces by: 10 percent for matrices of length 160*160, 9 percent for multiplying matrices of length 350*350 and 8 percent for multiplying matrices of length 450*450. The overall reduction in TT for matrix multiplication service in POP of DEAP client application is found (10.3+/-) 0.5 percent with 99% confidence in the sample space of 30 values.

The comparison of TT for the matrix multiplication operation in the POP of DEAP and traditional offloading signifies the usefulness of DEAP framework for computational offloading. Figure 6.35 shows the increasing trend of the TT in offloading matrix multiplication service. The increase in TT of matrix multiplication operation in runtime component offloading (without including profiling) as compared to the POP of DEAP is examined as follows: 74 percent for multiplying matrices of length 160*160, 72 percent for multiplying matrices of length 230*230, 64 percent for multiplying matrices of length 350*350 and 63 percent for multiplying matrices of length 450*450. Similarly, the increase in TT of matrix multiplication operation in runtime component offloading by including profiling as compared to the POP of DEAP is examined as follows: 79 percent for multiplying matrices of length 160*160, 77.8 percent for multiplying matrices of length

230*230, 73.9 percent for multiplying matrices of length 350*350 and 73.3 percent for multiplying matrices of length 450*450.

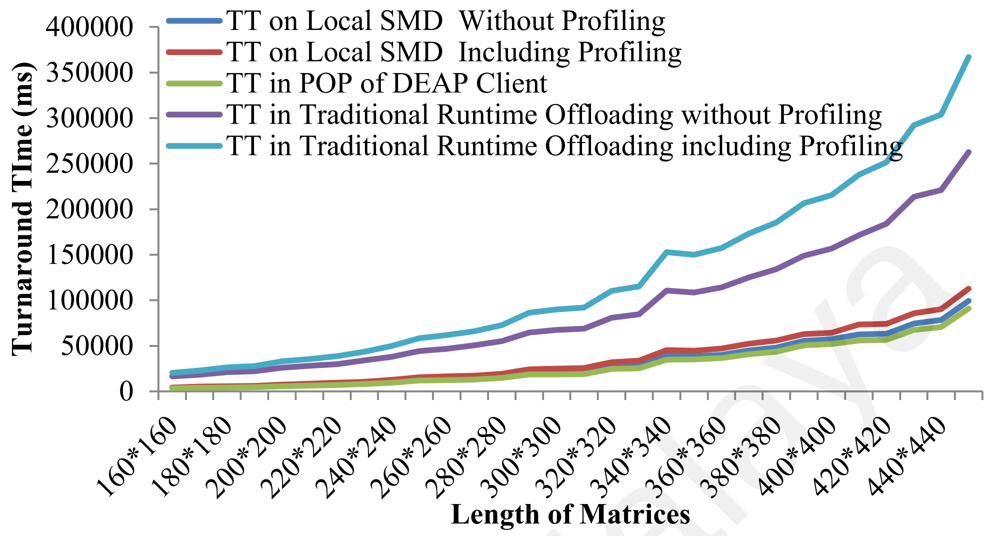

Figure 6. 35: Comparison of the Turnaround Time of the Matrix Multiplication Service Execution in Local and Remote Execution

Figure 6.36 shows the comparison the TT of power compute service execution in different scenarios. It is examined that the TT on the local SMD is smaller for performing power compute operation without runtime profiling as compared to power compute operation included with runtime profiling process. Experimental results indicate that by including the runtime profiling mechanism the value of TT increases 68.1 percent for computing 2^1000000, 84.8 for computing 2^20000000, 80.5 for computing 2^400000000 and 83.8 for computing 2^2000000000. The overall increase in the TT of power compute service by including runtime profiling is determined as 76.8(+/-)2.4 with 99% confidence in the sample space of 30 values.

The comparison of TT for power compute operation in local execution and traditional offloading technique shows that TT of the power computing increases considerably in runtime component offloading. It is examined that in offloading power compute service in the SOP of DEAP client application, the TT of power computing increases as follows: 99.3 percent for computing 2^1000000, 96.2 percent for computing 2^20000000, 81.4 percent for computing 2^400000000 and 74 for computing 2^2000000000. Similarly, in offloading power compute service with employing runtime profiling on the local mobile device the TT of the power compute operation in remote processing as compared to local execution on mobile device increases by 99.3 percent for computing 2^1000000, 96.9 percent for computing 2^20000000, 89.5 percent for computing 2^400000000 and 88.9 percent for computing 2^2000000000.

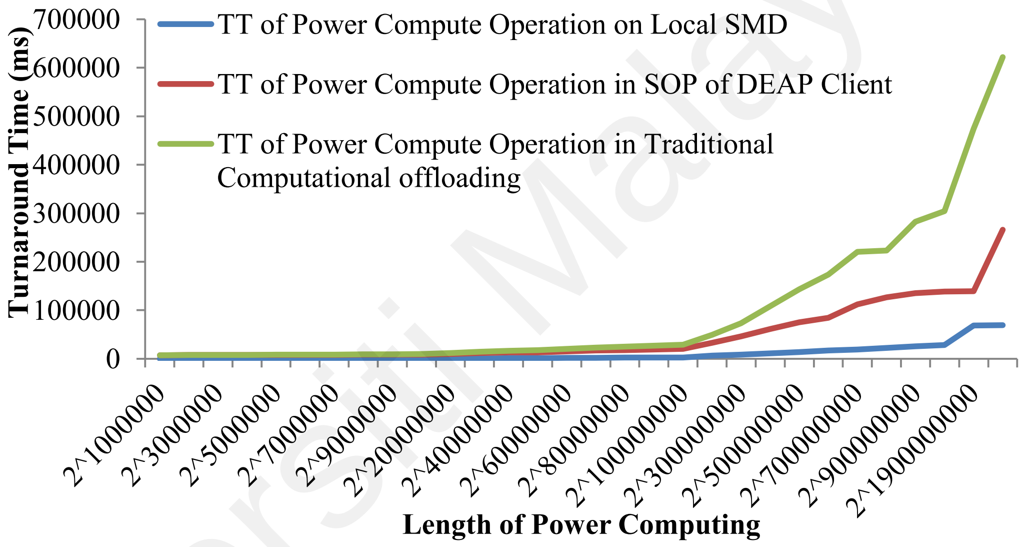

Figure 6. 36: Comparison of the Turnaround Time (of Power Compute Operation in in Local and Remote Execution

The comparison of power compute service execution on local mobile device and the DEAP based execution indicates that TT of power compute operation is increased in the SOP of DEAP client application. The comparison of TT for power compute operation in local execution and SOP of offloading technique shows that TT of the power computing increases considerably in runtime component offloading. However, the increase in TT is higher in traditional runtime computational offloading with employing profiling mechanism as compared to the SOP of DEAP which offloads power compute service without employing profiling. The decrease in TT in SOP of DEAP client application as compared to the traditional runtime computational offloading which includes profiling mechanism is examined as 2.6 percent for computing 2^2000000, 9.1 percent for computing 2^9000000, 28 percent for computing 2^80000000 and 57.3 percent for computing 2^2000000000. The overall decrease in the TT of SOP of DEAP as compared to traditional computational offloading is 27(+/-)9.2 with 99% confidence for the sample space of 30 values.

This section discusses the comparison of energy consumption cost of different components of the prototype application in different scenarios. The ECC of the components of the mobile application is compared from the perspective of local execution of the intensive component of the application, and execution of the intensive component on the remote sever node. Table 5.34 summarized the comparison of energy consumption cost for sorting operation in different scenarios.

Figure 6.37 shows the comparison the ECC of sorting service execution in different scenarios. It is examined that the ECC on the local SMD is smaller for performing sorting operation without runtime profiling. Experimental results indicate that by including the runtime profiling mechanism the value of ECC increases 2.3 percent for sorting list of 11000 values, 57.2 percent for sorting list of 22000 values, 68.8 percent for sorting list of 30000 and 72.9 percent for sorting list of 40000 values. The overall increase in the ECC of sorting service by including runtime profiling in the sorting operation is determined as 97(+/-)1 with 99% confidence in the sample space of 30 values.

The comparison of ECC for sorting operation in local execution and traditional offloading technique shows that ECC of the sorting service increases considerably in runtime component offloading. The increase in ECC of the sorting service in remote processing compared to local execution of on mobile device is examined as follows: 28.7 percent for sorting list of 11000 values, 61.1 percent for sorting list 25000 values, 64.9 percent for sorting list of 30000 values and 65.9 percent for sorting list of 40000 values.

Similarly, in offloading sorting service by employing runtime profiling on the local mobile device, the increase in ECC of the sorting service in remote processing compared to local execution on mobile device is as follows: 58 percent for sorting list of 11000 values, 77 percent for sorting list 25000 values, 81 percent for sorting list of 35000 values and 82 percent for sorting list of 40000 values.

The comparison of sorting service execution on local mobile device and the DEAP based execution signifies the decrease in ECC of the sorting operation in the POP of DEAP client application. It is examined that by accessing the services of DEAP server application in the POP of DEAP client application on mobile device, the ECC of sorting services reduces 85.1 percent for sorting list of 11000 values, 85.2 percent for sorting list of 25000 values and 88.6 percent for sorting list of 40000 values. The overall reduction in ECC value for sorting service in POP of DEAP client application is 86(+/-)0.9 percent with 99% confidence in the sample space of 30 values.

The comparison of ECC for the sorting operation in the POP of DEAP and traditional offloading signifies the lightness of DEAP framework for computational offloading. Figure 6.37 shows the increasing trend of ECC in offloading sorting service at runtime. The ECC of sorting service in runtime component offloading (without including profiling) as compared to the POP of DEAP increases 85.1 percent for sorting list of 11000 values, 85.1 percent for sorting list of 20000 values, 88.4 percent for sorting list of 31000 values and 88.4 percent for sorting list of 40000 values. Whereas, ECC of sorting service in runtime component offloading by including profiling as compared to the POP of DEAP increases 85.2 percent for sorting list of 11000 values, 90.3 percent for sorting list of 20000 values, 93.6 percent for sorting list of 31000 values and 94 percent for sorting list of 40000 values.

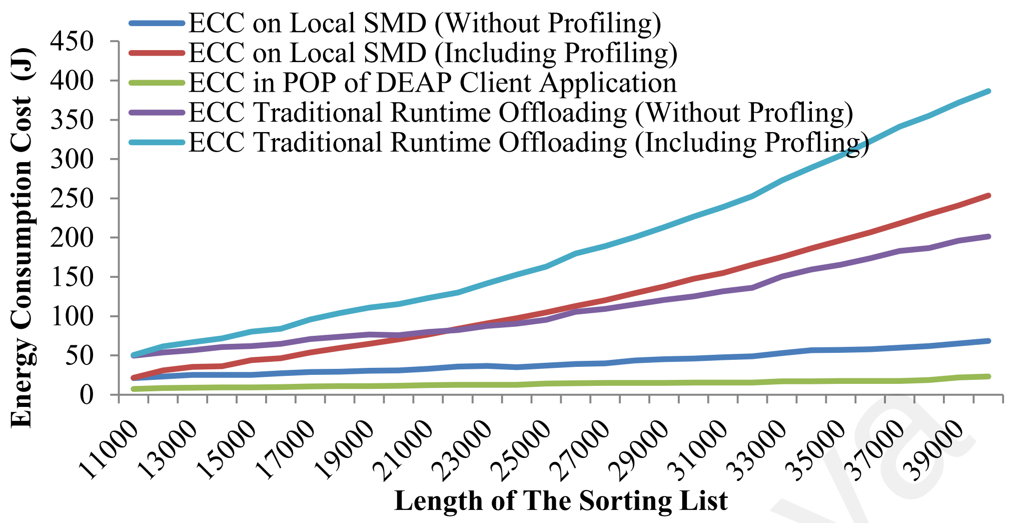

Figure 6. 37: Comparison of Energy Consumption Cost for Sorting Service in Local and Remote Execution

Table 5.35 summarized the comparison of the ECC of matrix multiplication operating of the application in local and remote execution. It is examined that the ECC on the local SMD is smaller for performing matrix multiplication without runtime profiling as compared to matrix multiplication included with runtime profiling process.

Figure 6.38 shows the comparison the ECC of matrix multiplication service execution in different scenarios. Experimental results indicate that by including the runtime profiling mechanism the value of ECC increases by 19.4 percent for multiplying matrices of length 160*160, 19.1 percent for multiplying matrices of length 250*250, 18.9 percent for multiplying matrices of length 350*350 and 22.1 percent for multiplying matrices of length 450*450. The overall increase in the ECC of matrix multiplication service by including runtime profiling in the matrix multiplication operation is determined as 20(+/-)1 with 99% confidence in the sample space of 30 values.

The comparison of ECC for matrix multiplication operation in local execution and traditional offloading technique shows that ECC of the matrix multiplication increases considerably in runtime component offloading. It is examined that in offloading matrix multiplication service without employing runtime profiling on the local mobile device the ECC of the matrix multiplication service in remote processing compared to local execution on mobile device increases by 68 percent for multiplying matrices of length 160*160, 59 percent for multiplying matrices of length 250*250, 51 percent for multiplying matrices of length 300*300 and 46 percent for multiplying matrices of length 450*450.

Similarly, in offloading matrix multiplication service with employing runtime profiling on the local mobile device, the ECC of the matrix multiplication service in remote processing compared to local execution on mobile device increases 70 percent for multiplying matrices of length 160*160, 62 percent for multiplying matrices of length 250*250, 57 percent for multiplying matrices of length 300*300 and 53 percent for multiplying matrices of length 450*450.

The comparison of matrix multiplication service execution on local mobile device and DEAP based execution signifies the decrease in ECC of matrix multiplication operation in the POP of DEAP client application. It is examined that in the POP of DEAP client application on mobile device, the ECC of matrix multiplication operation reduces 19.4 percent for multiplying matrices of length 160*160, 42 percent for multiplying matrices of length 270*270, 44 percent for multiplying matrices of length 350*350 and 9.5 percent for multiplying matrices of length 450*450. The overall reduction in ECC value for matrix multiplication service in POP of DEAP client application is 34.9(+/-) 5.4 percent with 99% confidence in the sample space of 30 values.

The comparison of ECC for the matrix multiplication operation in the POP of DEAP and traditional offloading signifies the lightness of DEAP framework for computational offloading. Figure 6.38 shows the increasing trend of ECC in offloading matrix multiplication service. The increase in ECC of matrix multiplication operation in runtime component offloading (without including profiling) as compared to the POP of

DEAP is examined as follows: 73 percent for multiplying matrices of length 160*160, 70.3 percent for multiplying matrices of length 230*230, 61.7 percent for multiplying matrices of length 350*350 and 50.4 percent for multiplying matrices of length 450*450. Whereas, the increase in ECC of matrix multiplication in runtime component offloading by including profiling as compared to the POP of DEAP is 74.9 percent for multiplying matrices of length 160*160, 72.9 percent for multiplying matrices of length 230*230, 66 percent for multiplying matrices of length 350*350 and 57 percent for multiplying matrices of length 450*450.

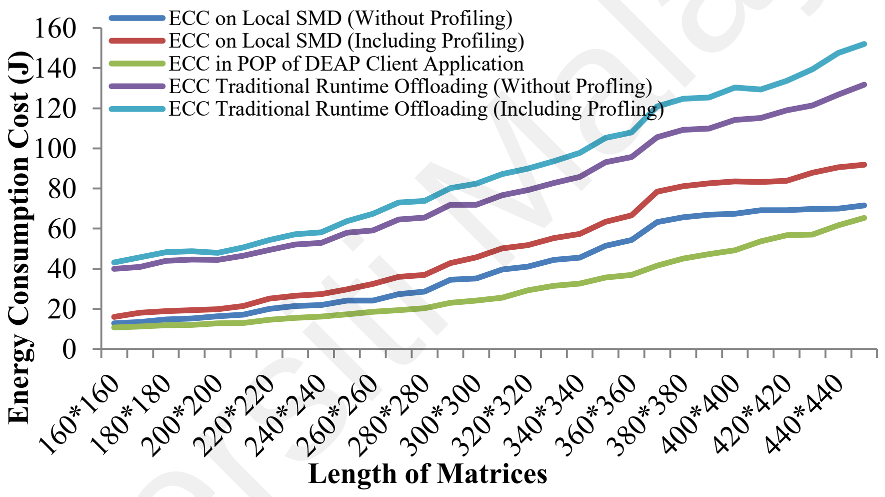

Figure 6. 38: Comparison of Energy Consumption Cost for Matrix Multiplication Service in Local and Remote Execution

Table 5.36 summarized the comparison of the ECC of the power compute operation of the application in local and remote execution scenarios. It is examined that the ECC on the local SMD is smaller for performing power compute operation without runtime profiling as compared to power compute operation included with runtime profiling process. Figure 6.39 shows the comparison the ECC of power compute service execution in different scenarios. Experimental results indicate that by including the runtime profiling mechanism the value of ECC increases by 11.5 percent for computing 2^2000000, 32.8 for computing 2^20000000, 57.6 percent for computing 2^400000000 and 62.1 percent for computing 2^2000000000. The overall increase in the ECC of power compute service by including runtime profiling is determined as 40(+/-)12 with 99% confidence in the sample space of 30 values.

The increase in the ECC of offloading power compute service in the SOP of DEAP client application is examined as follows: 59.3 percent for computing 2^1000000, 70.5 percent for computing 2^20000000, 72.6 percent for computing 2^400000000 and 80.9 for computing 2^2000000000. Similarly in offloading power compute service with traditional computational offloading by using profiling on the local mobile device the ECC of the power compute operation increases in remote processing compared to local execution on mobile device increases 59.3 percent for computing 2^1000000, 74.2 percent for computing 2^20000000, 80 percent for computing 2^400000000 and 85.5 percent for computing 2^2000000000.

The comparison of power compute service execution on local mobile device and the DEAP based execution indicates that ECC of power compute operation is increased in the SOP of DEAP Client application. The comparison of ECC for power compute operation in local execution and SOP of offloading technique shows that ECC of the power computing increases considerably in runtime component offloading. However, the increase in ECC is higher in runtime computational offloading as compared to the SOP of DEAP which offloads power compute service without profiling. The decrease in ECC in SOP of DEAP client application as compared to the traditional runtime computational offloading which includes profiling mechanism is examined as 4.4 percent for computing 2^2000000, 14.5 percent for computing 2^40000000, 31 percent for 2^500000000 and 23.8 percent for computing 2^2000000000. The overall decrease in the ECC of SOP of DEAP as compared to traditional computational offloading is 17(+/-)5.2 with 99% confidence for the sample space of 30 values.

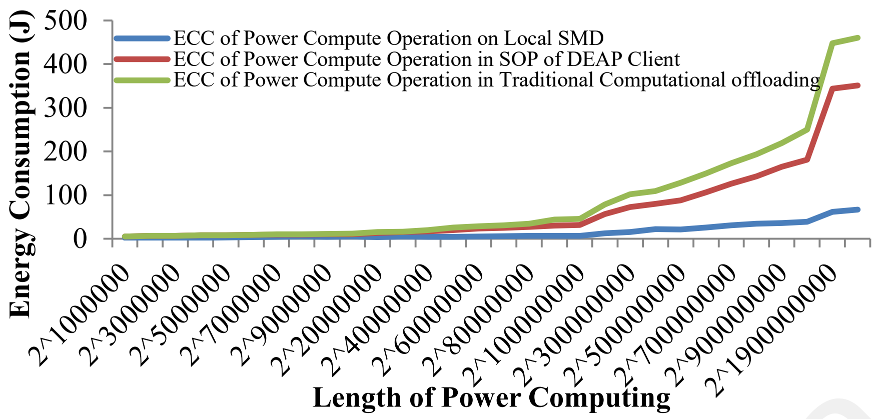

Figure 6. 39: Comparison of Energy Consumption Cost for Power Compute Service in Local and Remote Execution

Table 5.37 summarized the comparison of RAM utilization for sorting operation in executing sort service on local mobile device and remote execution in the POP of DEAP client application. It is examined that RAM allocation on the mobile device increase for the sorting service with the increase in the length of sorting list. For instance sorting service is allocated 10.148 MB RAM in sorting list of 11000 values, 10.21 MB in sorting list 25000 values and 10.265 MB in sorting list 40000 values. It indicates that the allocation of RAM on the mobile device varies with length of sorting list.

The allocation of RAM to DEAP client application varies in accessing sorting service of the DEAP server application. It is examined that the allocation of RAM to the DEAP client application increases by increasing the length of sorting list while accessing sorting operation of the DEAP server application. The size of resultant list returned to the local mobile device increases by increasing the length of sorting list, hence the size of RAM allocated to DEAP client application increases accordingly.

The comparison of RAM allocation in local sorting service execution and DEAP based sorting operation signifies the usefulness of DEAP framework. It is examined that the interval estimate for allocated RAM for sorting service component of the application is 10.20987(+/-) 0.016367 MB with 99% confidence for the sample space of 30 values, whereas in accessing sorting service of the DEAP Server application, additional RAM allocated for DEAP client application is 2.6023(+/-)0.457 MB with 99% confidence for the sample space of 30 values. It indicates that RAM allocation on the local mobile device is reduced 74.5 percent in accessing sorting operation in the POP of DEAP client application. Figure 6.40 shows the comparison of RAM allocation for sorting service in local application execution and remote execution in the POP of DEAP Client application.

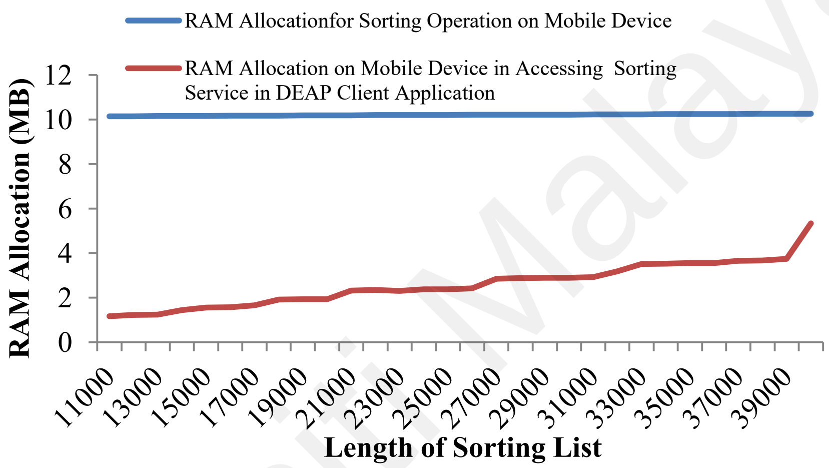

Figure 6. 40: RAM Allocation for Sorting Operation on SMD Local Service Execution and in POP of DEAP

Table 5.38 summarized the comparison of RAM utilization for matrix multiplication operation in executing matrix multiplication service on local mobile device and remote execution in the POP of DEAP client application.

Figure 6.41 shows the comparison of RAM allocation in local execution of matrix multiplication service and accessing matrix multiplication service in the POP of DEAP client application. It is examined that RAM allocation on the mobile device increase for the matrix multiplication service with the increase in the length of multiplying matrices. For instance, matrix multiplication service is allocated 2.78 MB RAM in multiplying matrices of length 160*160, 6.94 MB in multiplying matrices of length 25*250, 13.7 MB in multiplying matrices of length 350*350 and 22.8 MB in multiplying matrices of length

450*450. It indicates that the allocation of RAM on the mobile device varies with length of matrices being multiplied. Similarly, the allocation of RAM to DEAP client application varies in accessing matrix multiplication service of the DEAP server application.

It is examined that allocation of RAM to the DEAP client application increases by increasing the length of multiplying matrices size while accessing matrix multiplication service of the DEAP server application. The size of resultant matrix returned to the local mobile device increases by increasing the length of multiplying matrix, hence the size of RAM allocated to DEAP client application increases accordingly.

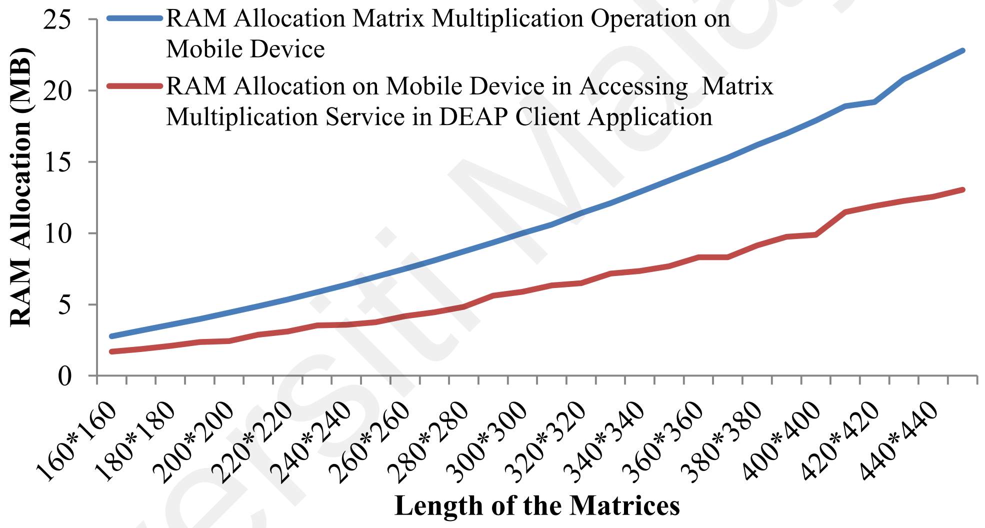

Figure 6. 41: Comparison of RAM Allocation in Local Execution of Matrix Multiplication Service and in the POP of DEAP

The comparison of RAM allocation in local matrix multiplication service execution and DEAP based matrix multiplication operation signifies the usefulness of DEAP framework. It is examined that the sample mean for allocated RAM for matrix multiplication service component of the application is 11.205(+/-)2.866 with 99% confidence for the sample space of 30 values, whereas in accessing matrix multiplication service of the DEAP server application, additional RAM allocated for DEAP client application is 6.467(+/-)1.6723 MB with 99% confidence for the sample space of 30 values. Analysis of the results indicates that RAM allocation on the local mobile device is reduced 42.2 percent in accessing matrix multiplication operation in the POP of DEAP client application.

The power compute service of the application is offloaded at runtime by employing SOP of DEAP client application. Hence, the entire logic of the power computing is executed on the remote server node. It is examined that in the SOP the allocation of RAM to the local mobile device remains constant and the increase in the RAM allocation is observed as zero percent. It is determined that sample mean for RAM allocation in local execution of power computing is 10.11(+/-).00045 MB with 99% confidence with the sample space of 30 values. However, the allocation of RAM is reduced to zero percent in SOP of DEAP Client application. Hence, the RAM allocation for power compute operation is saved up to 100% in the SOP of DEAP client application.

In the POP of DEAP client application the entire logic of the intensive components of the mobile application is offloaded to the preconfigured server. Hence, the application running on the local mobile devices is not required to allocate RAM for the execution of offloaded components of the application. It is observed that the allocation of RAM for the DEAP client application on local mobile device remains constant for accessing the sorting operation, matrix multiplication operation and power compute operation on the DEAP server application. It indicates that the increase in the RAM allocation for DEAP client application reduces to zero during the execution of services on the preconfigured server application on the remote server node. However, the allocation of RAM increases gradually whenever the size of data returned from the remote server node increases. For instance, the allocation of RAM for DEAP client application increases 1.14 percent for the returned sorted list of 40000 values as compared 11000 values. Similarly, the allocation of RAM for

DEAP client application increases 87.8 percent for the returned resultant 450*450 size matrix as compared 160*160 size matrix.

Table 5.39 summarized the comparison of CPU utilization in the execution of mobile application on local mobile device and remote execution by employing the POP and SOP of DEAP framework. Figure 6.51 compares CPU utilization on the mobile device in local execution of components the mobile application and accessing the services of cloud server node. The execution of application on local mobile devices resulted in high CPU utilization for a longer period of time as compared to accessing the services of cloud server node.

It is examined that the average CPU utilization for executing sorting service on local mobile device is 48.67 percent of the total CPU utilization on local mobile device for 17427(+/-) 3707 ms duration. However, in accessing the sorting service of DEAP server application on the cloud server node, the CPU utilization for DEAP client application on local mobile device is observed as 25.5 percent of the total CPU utilization for 7224(+/-) 1560 ms duration. Analysis of the comparison for CPU utilization between local sort service execution and accessing sorting service of cloud server node indicates the average CPU utilization for sorting operation reduces 47.6 percent on the local mobile device. Further, the period of CPU utilization on the local mobile device is reduced up to 58.5 percent.

The average CPU utilization for executing matrix multiplication service on local mobile device is 45.46 percent of the total CPU utilization on local mobile device for 31190(+/-) 12270 ms duration. However, in accessing the matrix multiplication service of DEAP server application on the cloud server node, the CPU utilization for DEAP client application on local mobile device is observed as 35.4 percent of the total CPU utilization for 28085(+/-) 11132 ms duration. Analysis of the comparison for CPU utilization between local matrix multiplication service execution and accessing matrix multiplication service of cloud server node indicates the average CPU utilization for matrix multiplication operation reduces 22.1 percent on the local mobile device. Further, the period of CPU utilization on the local mobile device is reduced up to 10 percent for matrix multiplication operation in DEAP based computational offloading.

The average CPU utilization for executing power compute service on local mobile device is 48 percent of the total CPU utilization on local mobile device. However, it is examined that the CPU utilization for operating system CPU utilization increases up to 3 percent on the Android virtual device, whereas for the physical mobile device the increase in CPU utilization is zero percent.

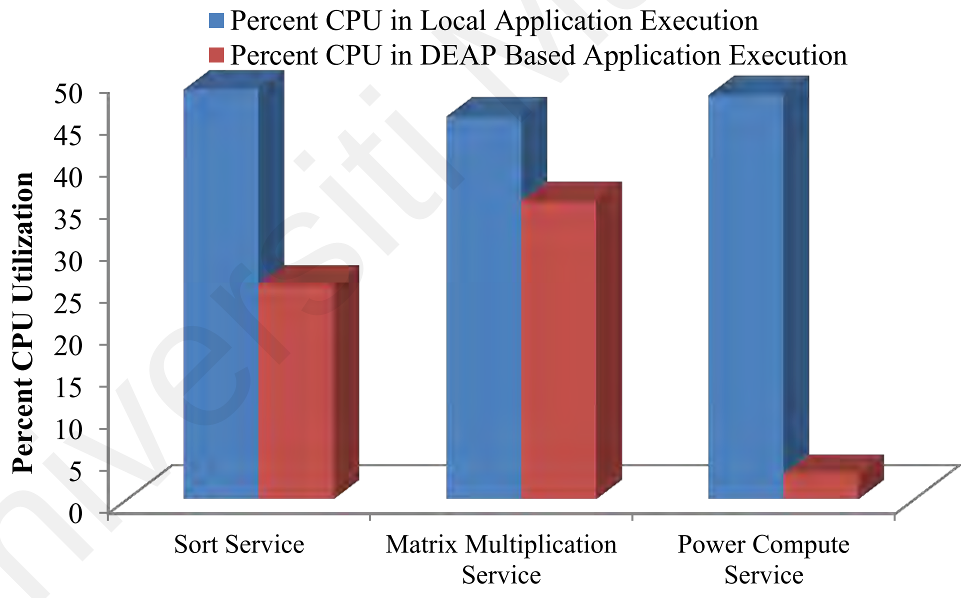

Figure 6. 42: Comparison of CPU Utilization in Local Application Execution and DEAP Based Execution

The comparison for CPU utilization between local power compute service execution and accessing power computing service of cloud server node indicates the average CPU utilization for power computing operation reduces 93.8 percent on the local mobile device.

Further, the period of CPU utilization on the local mobile device is reduced up to 90 percent for power computing operation in DEAP based computational offloading.

In the POP and SOP of DEAP client application the entire logic of the intensive components of the mobile application is offloaded to cloud sever node. Hence, the application running on the local mobile devices is not required to schedule CPU for the execution of offloaded components of the application. It is observed that the utilization of CPU for the DEAP client application on local mobile device remains constant for accessing the sorting operation, matrix multiplication operation and power compute operation on the DEAP server application. It indicates that the increase in the CPU utilization for DEAP client application reduces to zero during the execution of services on the cloud server node.

However, the results returned for the remote server node to mobile device are extracted from returned SOAP message which requires additional processing on the local mobile device. Hence, the percentage and the time period of CPU utilization for DEAP client application increases gradually whenever the size of data returned from the remote server node increases. For instance, CPU utilization for the returned sorted list of 40000 values increases 22.5 percent and for the 72.6 percent longer period of time as compared to sorted list of 110000 values. Similarly, CPU utilization for the returned resultant of 450*450 size matrix increases 32.4 percent and for the 96.38 longer period of time as compared to the resultant matrix of 160*160 size.

Table 5.40 summarized the comparison of the size of data transmitted over the wireless network medium for sorting service in offloading computational load in the proposed DEAP framework and traditional application offloading technique. It is examined that larger amount of data is transmitted in traditional component offloading as compared to the DEAP based computational offloading. Figure 6.43 shows the comparison of data transmission in traditional runtime application offloading and proposed DEAP based computational offloading for sort service component of the application.

The size of data transmission over the wireless network medium varies for different length of sorting list. It is examined that the data transmission size for offloading sort service with the list of 11000 values is 752.4 KB, whereas the size of data transmission in accessing sorting service of DEAP server application is 183 KB. Similarly, the size of data transmission is 2645.4 KB for list of 40000 values in traditional computational offloading, whereas the size of data transmission in DEAP is 692 KB in accessing sorting service of DEAP server application. It shows that in DEAP based computational offloading the size of data transmission over the wireless medium is reduced 76 percent for sorting list of 1100 values and 74 percent for sorting list of 40000 values. The average reduction of data transmission over the wireless network medium is 74.7 percent in DEAP based computational offloading for the sorting list of 11000-40000 values.

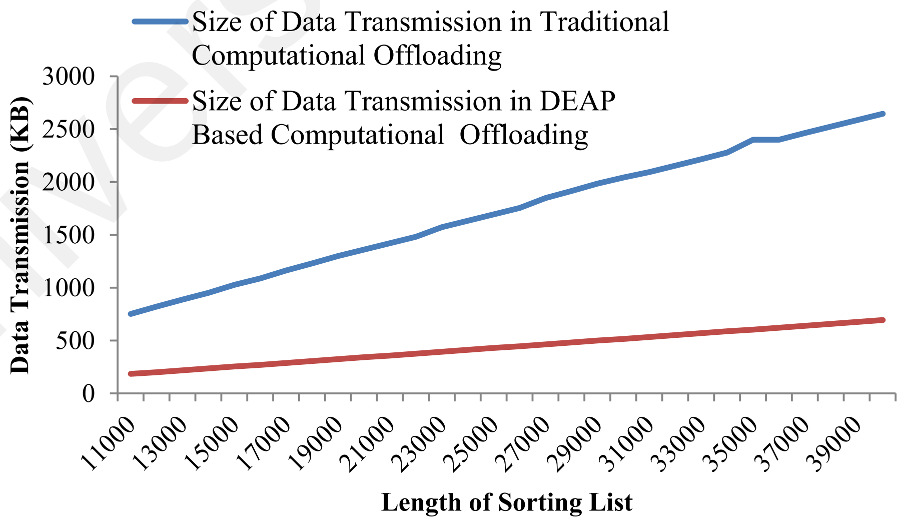

Figure 6. 43: Comparison of the Size of Data Transmission in Traditional Offloading and DEAP Based Offloading for Sorting Operation

Table 5.41 summarized the comparison of the size of data transmitted over the wireless network medium for matrix multiplication service in offloading computational load in the proposed DEAP framework and traditional application offloading technique. It is examined that larger amount of data is transmitted in traditional component offloading as compared to the DEAP based computational offloading.

Figure 6.44 shows the comparison of data transmission in traditional runtime application offloading and proposed DEAP based computational offloading for matrix service component of the application. The size of data transmission over the wireless network medium varies for different size of matrices. It is examined that the data transmission size for offloading matrix multiplication service with the size of matrix 160*160 values is 5739.44 KB, whereas the size of data transmission in accessing matrix multiplication service of DEAP Server application is 463 KB. Similarly, the size of data transmission is 46740.4 KB for the size of matrix 450*450 values in traditional computational offloading, whereas the size of data transmission in DEAP is 3308 KB in accessing matrix multiplication service of DEAP server application. It shows that in DEAP based computational offloading the size data transmission over the wireless medium is reduced 91.9 percent for matrix size 160*160 and 92.2 percent for matrix size of 450*450 values. The average reduction of data transmission over the wireless network medium is 92 percent in DEAP based computational offloading for the matrices of size 160*160-450*450.

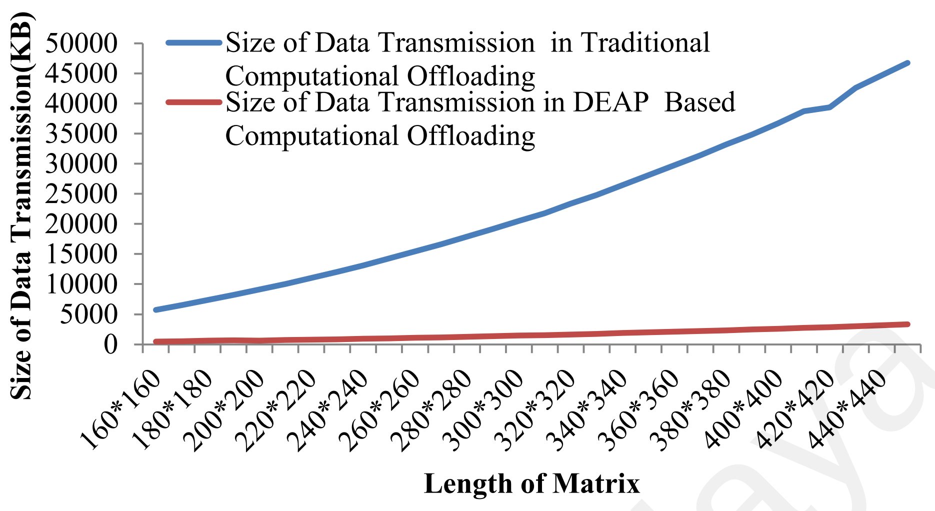

Figure 6. 44: Comparison of the Size of Data Transmission in Traditional Offloading and DEAP Based Offloading for Matrix Multiplication Operation

6.5 Conclusion

DEAP framework employs lightweight procedures for leveraging application processing services of computational clouds. DEAP reduces computing resources (RAM and CPU) utilization on mobile device by leveraging the service of cloud server node. It is observed that in the POP of DEAP, the allocation of RAM and CPU utilization on mobile device for the DEAP client application remains constant for accessing the sorting operation, matrix multiplication operation and power compute operation on the DEAP server application. It shows that the increase in the RAM allocation and CPU utilization for DEAP client application reduces to zero during the execution of services on the preconfigured server application on the remote server node. However, the allocation of RAM on the local mobile device increases for processing the resultant data returned from the clouds server node. It is examined that RAM allocation is reduced 74.5 percent in sorting the list of 11000-40000 values, 42.28 percent in multiplying matrices of length 160*160-450*450 in the POP of DEAP framework and 100% in offloading power compute service to cloud server node in the SOP of DEAP framework. Similarly, CPU utilization is reduced 47.6 percent for sorting list 11000-40000 values, 22.2 percent in 93.8 percent in offloading power compute service in the SOP of DEAP framework.

DEAP framework reduces the cost of migration of application binary file and data file of the running instances of the mobile application. As a result, the size of data transmission over the wireless network medium, turnaround time of the intensive operations and energy consumption cost on mobile device is reduced considerably. Analysis of the results signifies the lightweight nature of DEAP framework by reducing the size of data transmission, turnaround time and energy consumption cost in cloud based processing of the intensive component of mobile application. The data communication cost over the wireless network medium is reduced considerably in DEAP application. The distributed nature of DEAP results in the reduction of the size of data transmission over the wireless network medium. It is observed that the POP of DEAP framework eliminates the additional cost application binary code migration and active data state migration to the cloud server node at runtime.

Hence, the energy consumption cost and turnaround time of the component of the mobile application are reduced. For instance, the size of data transmission for sorting service is reduced 74.8 percent, turnaround time of sorting operation is reduced 91.3 percent and energy consumption cost is reduced 86.7 percent compared to the traditional computational offloading technique. Similarly, the size of data transmission for matrix multiplication operation is reduced 92.8 percent, turnaround time is reduced 67.8 percent and energy consumption cost is reduced 64.2 percent compared to the traditional computational offloading technique. It is concluded that DEAP framework provides a lightweight application layer solution for the distributed processing of intensive mobile applications in mobile cloud computing.

CHAPTER 7

Conclusion

This chapter reevaluates the research objectives, summarizes contribution of the research, discusses the scope and limitation, and future research directions of this research.

The chapter is organized into four sections. Section 7.1 discusses the reappraisal of the objectives of this research work. Section 7.2 summarizes contribution of the research work. Section 7.3 discusses the scope of the research work. Section 7.4 proposes future directions of the research works

7.1 Reappraisal of the Objectives Achievement

This research investigates the problem of additional computing resources utilization in traditional computational offloading techniques and proposes a lightweight DEAP framework for the processing of intensive mobile applications in MCC. Section 1.4 highlighted four objectives of this research, which are achieved as follows:

A thematic taxonomy of traditional computational offloading frameworks is produced to achieve the objective of literature review. We studied state-of-the-art from web databases and online digital libraries (IEEE, ACM and ISI Web of Knowledge). We selected 250 papers in the broader domain of cloud computing, mobile computing and mobile cloud computing and reviewed the latest literature for current DAPFs by selecting 25 frameworks for five years (2007-2012). Computational offloading frameworks are reviewed for four different distributed application processing models which are categorized into a thematic taxonomy (Section 2.2.1). Qualitative analysis is used to highlight the implications and critical aspects of current computational offloading frameworks (Section 2.2.2). Current

DAPFs are synthesized on the basis of taxonomy (Section 2.4) and the issues and challenges for computational offloading in MCC are highlighted.

The research problem is established by quantitative analysis of runtime computational offloading and Virtual Machine (VM) deployment for application processing. A prototype application is benchmarked in the real mobile cloud computing environment for evaluating the additional energy consumption cost, timing cost and size of data transmission in the traditional computational offloading techniques (section 3.2). The impact of virtual machine deployment for application processing is analyzed in the simulation environment (section 3.3).

A lightweight Distributed and Elastic Application Processing (DEAP) framework is proposed to achieve the objective of lightweight solution for the processing of intensive mobile applications in MCC. DEAP framework addresses the problem of additional computing resources utilization in distributed processing of intensive mobile applications. It proposes lightweight operation procedures for the processing of intensive mobile applications in MCC (Section 4.2). DEAP framework reduces the cost of migration of application binary file and data file of the running instances of the mobile application. As a result, the size of data transmission over the wireless network medium, turnaround time of the intensive operations and energy consumption cost on mobile device is reduced considerably.

Synthetic workload is tested in the emulation environment on the Android virtual device instance to achieve the objective of evaluating DEAP. Virtual device instance is enabled to operate in the distributed mobile cloud computing environment. The performance of DEAP framework is validated by benchmarking prototype application in the real mobile cloud computing environment.

Results of different experimental scenarios are compared to validate the lightweight nature of DEAP framework. DEAP framework reduces resources utilization on SMD and the cost of distributed processing of the prototype application in MCC as follows: RAM allocation on mobile device 71.4 percent, CPU utilization on mobile device 55 percent, the size of data transmission over the wireless network medium 84 percent, turnaround time of the application 80.6 percent and energy consumption cost 69.9 percent. Hence, DEAP framework provides a lightweight application layer solution for intensive mobile application processing in MCC.

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 implications and critical aspects of current computational offloading frameworks and compare current DAPFs on the basis of significant parameters. Literature review contributed to identify issues and challenges for computational offloading in MCC.
- ***A Computational Intensive Prototype Application***: We developed a prototype application to evaluate the additional computing resources utilization in traditional computational offloading. The prototype application is composed of computational intensive components which are offloaded at runtime to analyze additional resource utilization in traditional computational offloading technique. We develop monitoring application for implementing the algorithm of active service level component migration on Android devices. The required managerial components are developed for the runtime distributed deployment and management of intensive components of mobile application. The prototype application is evaluated for

establishing the research problem. The measurement parameters for the analysis of traditional computational offloading include energy consumption cost, timing cost and size of network data traffic.

- ***An Extended Simulation Model***: We simulated the IaaS service provision model of computational clouds by using CloudSim. Cloud computing environment is simulated by modeling datacenters, hosts and datacenter broker. We model two datacenters and each datacenter is composed of fifty computing hosts. Each datacenter contains a datacenter broker which is responsible for arbitrating negotiations between SaaS and cloud providers; such negotiations are driven by Quality of Service (QoS) requirements. We create different number of VMs in various scenarios. Cloudlet is modeled for cloud based application services (such as content delivery, social networking, and business workflow). CloudSim organizes the complexity of an application in terms of its computational requirements. Cloud application service (cloudlet) is simulated with pre-assigned instruction length and data size overhead which is undertaken during its lifecycle. VM is simulated with the required computing specifications. Cloudlet is allocated to a certain VM inside the host of datacenter. VM deployment for application execution is evaluated in different scenarios.
- ***A Simulation tool (SmartSim):*** We developed SmartSim for the partitioning of elastic mobile applications. SmartSim models the application processing attributes of SMDs in an easy to set up environment. The SmartSim toolkit is developed on the basis of CloudSim by using J2SE. It supports both the system and behavior modeling of SMD components such as application processor, memory, resources provision, computing resources utilization evaluation, dynamic processing

management policies and computational intensive mobile application modeling for SMD. Currently, SmartSim implements iterative algorithm for the partitioning of elastic mobile application on mobile device. SmartSim is a Java based platform independent and generic simulation toolkit which can be easily configured for the evaluation of the application partitioning algorithms for MCC.

- ***A Lightweight DEAP Framework**:* We propose a novel Distributed and Elastic Application Processing (DEAP) framework for the processing of intensive mobile applications in MCC. DEAP framework focuses on reducing the cost of distributed application deployment and management for the processing of intensive mobile applications in MCC. DEAP bridges the gap of distributed design in current DAPFs for the processing of intensive mobile applications in MCC. The Primary Operating Procedure (POP) of DEAP framework employs the SaaS service provision model of computational clouds for the deployment of distributed processing of intensive mobile applications in MCC. The distributed aspect of DEAP employs lightweight procedure for the processing of intensive mobile applications in MCC. DEAP framework sustains robust features of the elasticity of traditional frameworks by employing runtime component offloading as Secondary Operation operating Procedure (SOP). The SOP of DEAP framework is significant for computational offloading of traditional mobile applications which lack in the distributed design for cloud based distributed processing.
- ***DEAP Framework Based Prototype Application***: We develop a prototype application to implement the algorithm of DEAP framework. The prototype application is evaluated with varying computational intensities to validate the usefulness of DEAP framework. We implement the operating procedures of DEAP

by developing our own Application Programming Interface (API) for enabling active service migration to cloud server nodes at runtime. Socket programming is employed for synchronization and data communication between the local mobile device and cloud server node. The API provides middleware services for masking the complexities of computational offloading at runtime. The API implements the services of uploading the computational load of the mobile device at runtime.

DEAP framework reduces the cost of migration of application binary file and data file of the running instances of the application. As a result, the size of data transmission over the wireless network medium, turnaround time of the intensive operations and energy consumption cost on mobile device is reduced considerably. DEAP framework reduces additional resources utilization for the processing of intensive mobile application in MCC. Hence, DEAP framework provides a lightweight application layer solution for intensive mobile application processing in MCC.

We were awarded the following distinctions while doing this research.

<u>Distinctions</u>

- **Winner of the 3 Minutes Thesis (3MT) Competition** at Faculty of Computer Science and Information Technology, University of Malaya, Malaysia in July 2013.
- **Best Research Paper Award** in the 1st Post Graduate Symposium for Excellence organized by Faculty of Computer Science and Information Technology University of Malaya, Malaysia in September, 2011.
- **Runner up in the** 2nd Post Graduate Symposium for Excellence organized by Faculty of Computer Science and Information Technology University of Malaya, Malaysia September, 2012.

We produced the following research papers from this research.

Accepted Articles

- Muhammad Shiraz, Ejaz Ahmed, Abdullah Gani, Qi Han Investigation on Runtime Partitioning of Elastic Mobile Applications for Mobile Cloud Computing Journal of Supercomputing, DOI:10.1007/s11227-013-0988-6, August 2013 (ISI Indexed Q2, Impact Factor 0.917)
- Muhammad Shiraz, Abdullah Gani, Rashid Hafeez Khokhar, and Rajkumar Buyya, A Review on Distributed Application Processing Frameworks in Smart Mobile Devices for Mobile Cloud Computing, IEEE Communications Surveys & Tutorials, Volume 15, Issue 3, November 2012November 2012. (ISI Indexed Q1, Impact Factor 6.3, the top Most Journal of the Domain in ISI WoS Ranking for 2011)
- Muhammad Shiraz, Saeid Abolfazli, Zohreh Sanaei, Abdullah Gani A study on virtual machine deployment for application outsourcing in mobile cloud computing, Publication in Journal of Supercomputing, Volume 63, No. 3, pp. 946-964 March 2013 (ISI Indexed Q3, Impact Factor 0.578)
- Muhammad Shiraz, Abdullah Gani, Rashid Hafeez Khokar An Extendable Simulation Framework for Modeling Application Processing Potentials of Smart Mobile Devices for Mobile Cloud Computing, Proceedings of Frontiers of Information Technology 2012, Pakistan, 19-21 December 2012. (ACM, IEEE Indexed Publication)
- Muhammad Shiraz, Abdullah Gani, Rashid Hafeez Khokar Towards Lightweight Distributed Applications For Mobile Cloud Computing, 2012 IEEE International Conference on Computer Science and Automation Engineering (CSAE 2012),

Zhangjiajie, China, May 25-27, 2012 (Scopus, , Ei-Compendex and IEEE Indexed Publication Publication)

- Muhammad Shiraz, Abdullah Gani "Mobile Cloud Computing: Critical Analysis of Application Deployment in Virtual Machines" 2012 Proceedings of ICICN, Singapore, 2012, 26-28 February 2012 (Scopus and ISI Indexed)

<u>Articles under Review:</u>

- Muhammad Shiraz,Ejaz Ahmed, Mehdi Sookhak, Abdullah Gani, Rajkumar Buyya A Lightweight Distributed Framework for Computational Offloading in Mobile Cloud Computing, Pervasive and Mobile Computing, Under Review Since July 2013 (ISI Indexed Q1, Impact Factor 1.629)
- Muhammad Shiraz, Abdullah Gani, Ejaz Ahmed, Rajkumar Buyya An Energy Efficient Computational Offloading Framework for Mobile Cloud Computing, Journal of Grid Computing, Under Review Since May 2013(ISI Indexed Q1, Impact Factor 1.607)
- Muhammad Shiraz, Abdullah Gani, Ejaz Ahmed Computational Offloading for Mobile Cloud Computing, Issues and Challenges, Journal of Grid Computing, Under Review, Since April 2013(ISI Indexed Q1, Impact Factor 1.607)
- Muhammad Shiraz, Abdullah Gani A Distributed and Elastic Application Processing Model for Mobile Cloud Computing, Frontiers of Computer Science, Under Review Since April 2013, (ISI Indexed Q3, Impact Factor 0.298)
- Muhammad Shiraz, Abdullah Gani, Ejaz Ahmed, Qi Han,A Lightweight Active Service Migration Framework for Intensive Mobile Applications in Mobile Cloud Computing, Journal of Supercomputing, Under Review Since May 2013(ISI Indexed Q2, Impact Factor 0.807).

This research has contributed to the following collaborative research articles.

Articles in Collaboration with Group Members

- Liu Jie Yao, Muhammad Shiraz, Ejaz Ahmed, Abdullah Gani, Partitioning of Elastic Mobile Applications for Mobile Cloud Computing: Review, Issues and Challenges, IEEE Communication Surveys and Tutorials, Under Review, March, 2013 (ISI Indexed Q1, Impact Factor 6.3)
- Ejaz Ahmed, Muhammad Shiraz, Abdullah Gani Seamless Application Execution in Mobile Cloud Computing, Review, Challenges and Issues, IEEE Communication Surveys and Tutorials, Under Review, March, 2013 (ISI Indexed Q1, Impact Factor 6.3)
- Saeid Abolfazli, Zohreh Sanaei, Muhammad Shiraz, Abdullah Gani, MOMCC: Market-Oriented Architecture for Mobile Cloud Computing Based on Service Oriented Architecture, MobiCC 2012 : 2012 Proceedings of IEEE Workshop on Mobile Cloud Computing, Beijing, China (ISI-Indexed Publication)
- Zohreh Sanaei, Saeid Abolfazli, Abdullah Gani, Muhammad Shiraz, SAMI: Service-Based Arbitrated Multi-Tier Infrastructure for Mobile Cloud Computing, MobiCC 2012 Proceedings of IEEE Workshop on Mobile Cloud Computing, Beijing, China (ISI-Indexed Publication).
- Laleh Boroumand, Muhammad Shiraz, Abdullah Gani A Review on Port-Knocking Authentication Methods for Mobile Cloud Computing, Computing, under review since June 2013 (ISI Indexed Q2, Impact Factor 0.807)
- Laleh Boroumand, Muhammad Shiraz, Abdullah Gani, Rashid Hafeez Khokar Global healthcare village based on mobile cloud computing, Journal of Supercomputing, under review since June 2013 (ISI Indexed Q2, Impact Factor 0.917)

- Ejaz Ahmed, Muhammad Shiraz, Abdullah Gani, Spectrum-aware Distributed Channel Assignment in Cognitive Radio Wireless Mesh Networks, Malaysian Journal of Computer Science, First Revision Submitted,August 2013(ISI Indexed Q4)
- Ejaz Ahmed, Muhammad Shiraz, Abdullah Gani, A Survey on Design Perspectives of Seamless Distributed Application Execution Frameworks For Mobile Cloud Computing, Journal of Computer Networks and Applications, under review, May 2013, (ISI Indexed Q1, Impact Factor 1.065)
- Abdullah Gani, Muhammad Shiraz,Golam Muhammad Naeem A Review on Interworking and Mobility Techniques for Seamless Connectivity in Mobile Cloud Computing, Journal of Computer Networks and Applications, under review since June 2013 (ISI Indexed Q1, Impact Factor 1.46)
- Mehdi Sookhak, Md Whaiduzzaman, Muhammad Shiraz, Abdullah Gani, A Survey on Remote Data Checking Auditing in Cloud Computing, ACM Computing Surveys, under review, since July 2013 (ISI Indexed Q1, Impact Factor 3.543)
- Mehdi Sookhak, Md Whaiduzzaman, Muhammad Shiraz, Abdullah Gani, Anomaly Detection on Wormhole in Geographic Routing Protocols of Wireless Sensor Networks, Journal of Computer Networks and Applications, under review Since August 2013 (ISI Indexed Q1, Impact Factor 1.46)

7.3 Research Scope and Limitations

The scope of this research is limited to analyzing the problem of heaviness in traditional computational offloading and proposing lightweight solution for the processing of computational intensive mobile applications in MCC. This research lacks in the consideration of the supplementary issues which are associated in leveraging the application processing services of computational clouds. The supplementary issues includes seamless application execution in mobile cloud computing, richness of local services and offline usability, privacy of the application processing on the cloud server nodes, security in the wireless communication network and cloud datacenters, consistency of parallel execution between local mobile device and remote server node and homogenous services provision for the heterogeneous mobile devices operating platforms.

7.4 Future Work

This research is focused on the lightness of distributed deployment for the processing of intensive mobile applications in MCC. It emphasizes on the minimization of computing resources utilization on smart mobile device in cloud based processing of intensive mobile applications. However, it lacks of considering supplementary issues associated with computational offloading for MCC. Hence, the future research work includes extending the scope this research to address the issues of seamless application execution, smart applications for mobile cloud computing and heterogeneous service provision for mobile devices operating platform. The following section discusses the future directions of this research.

- The issue of seamless application execution in MCC is aimed to be addressed in our future research. The seamless features of mobile application include concealing the complexity of distributed application processing from mobile users. It is aimed to investigate optimal middleware solutions for achieving the goals of seamless application execution in computational offloading to cloud server nodes.
- We aim to address the issue of rich user experiences in distributed processing of intensive mobile applications in our future research. Smart applications for mobile cloud computing are attributed with the features of offline usability and rich user

experience which can operate in the situations of inaccessibility of cloud server nodes. It is expected to analyze the incorporation of distributed application architecture for the design and development of smart mobile applications for mobile cloud computing.

- The issue of heterogeneity in the operating system platform and hardware architecture of the smart mobile devices is aimed to be addressed in our future research. Homogenous solution is expected to provide a uniform service provisioning model for heterogeneous devices, operating platforms and network technologies with minimum possible resources utilization on the mobile device. We aim to investigate the deployment the SaaS service provisioning model on the cloud server nodes for addressing the issue of heterogeneity in mobile devices architectures and operating platforms.

References

Abebe, E., & Ryan, C. (2012). Adaptive application offloading using distributed abstract class graphs in mobile environments. Journal of Systems and Software, 85(12), 2755-2769.

Abolfazli, S., Sanaei. Z., & Gani. A., (2011). “Mobile Cloud Computing: A Review on Smartphone Augmentation Approaches”. Proceedings of The 1st International Conference on Computing, Information Systems and Communications, Singapore, 11-13 May, 2012.

Albanesius, C. (n.d.). Smartphone shipments surpass PC shipments for first time. what’s next? Accessed on 15th December 2011.

Ali, M. (2009) “Green Cloud on the Horizon.” Proceedings of the 1st International Conference on Cloud Computing (CloudCom), Lecture Notes in Computer Science, Springer, 5931, 451-459, 2009.

Amazon S3. http://status.aws.amazon.com/s3-20080720.html Accessed on 20th July, 2011

Android Developers developer.android.com/ Accessed on 1st January, 2012.

Android Debug Bridge developer.android.com/tools/help/adb.html Accessed on 1st January, 2012

Armbrust, M., Fox, A., Grifth, A., Joseph, D. A., Katz, H. R., Konwinski, A., Lee, G., Patterson, A. D., Rabkin, A., Stoica, A., & Zaharia, M. (2009). Above the Clouds: A Berkeley View of Cloud Computing. Electrical Engineering and Computer Sciences University of California at Berkeley.

Bahl, P., Han., Y. R., Li., E. L., Satyanarayanan., M. (2012). Advancing the State of Mobile Cloud Computing. MCS’12, Low Wood Bay, Lake District, UK, June 25, 2012.

Balan, K. R., Satyanarayanan, M., Park, Y. S., & Okoshi, T. (2003) Tactics-Based Remote Execution for Mobile Computing. MobiSys‘03: Proceedings of the 1st International conference on Mobile systems, applications and services, ACM Press, San Francisco, CA, USA 5-8 May (pp. 273-286), 2003.

Baldauf, M., Dustdar, S., & Rosenberg. F. (2007). A Survey On Context-Aware Systems. International Journal of Ad Hoc and Ubiquitous Computing, 2(4), 263–277.

Barga, R., Auban, B. J., Gannon, D., Gannon, C. (2009). Cloud Computing Architecture and Application Programming, News Microsoft Corporation SIGACT, ACM press, 40(2), June, 2009.

Barga, R., Gannon, D., Reed, D. (2011). The Client and the Cloud Democratizing Research Computing. IEEE Internet Computing IEEE Computer Society, 15(1), 72-75.

Begum, Y., & Mohamed. M. (2010). “A DHT-based process migration policy for mobile clusters,” in 7th International Conference on Information Technology, Las Vegas, 12-14 April (934–938), 2010.

Buyya, R., Yeo, C. S., Venugopal, S., Broberg, J., & Brandic, I. (2009). Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. Future Generation computer systems, 25(6), 599-616.

Caceres, R., Cox, L., Lim, H., Shakimov, A., Varshavsky, A. (2009) Virtual Individual Servers as Privacy-Preserving Proxies for Mobile Devices, MobiHeld’09 Barcelona, Spain, 17 -18 August (37-42), 2009, ACM.

Calheiros, N. R., Ranjan, R., Beloglazov, Rose, D. F. A. C., & Buyya, R. (2011) CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms. Software:Practice and Experience, 41(1), 23–50, January 2011.

Canepa, H. G., & Lee, D. (2010) A Virtual Cloud Computing Provider for Mobile Devices, ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond MCS’10, San Francisco, California, USA ACM Press, 15 June, 2010.

Chandramouli, R., Kharrazi, M., & Memon, N. (2004). Image steganography and steganalysis: Concepts and practice. In Digital Watermarking (pp. 35-49). Springer Berlin Heidelberg.

Chetan, S., Kumar, G., Dinesh, K., Mathew, K., & Abhimanyu, M. A. (n.d.) Cloud Computing for Mobile World. http://www.chetan.ueuo.com/projects/CCMW.pdf , Accessed on 28th June 2011

Choi, K. M., Robles, J. R., Hong, H. C., & Kim, H. T. (2008). Wireless Network Security: Vulnerabilities, Threats and Countermeasures. International Journal of Multimedia and Ubiquitous Engineering, 3(3), July, 2008.

Chow, R., Jakobsson, M., & Masuoka, R. (2010). Authentication in the Clouds: A Framework and its Application to Mobile Users. CCSW’10, Chicago, Illinois, USA, October, 2010, ACM Press

Chun, G. B., & Maniatis, P., (2009). Augmented Smartphone Applications Through Clone Cloud Execution, Intel Research Berkeley.

Chun, G. B., & Maniatis, P. (2010). Dynamically Partitioning Applications between Weak Devices and Clouds. Intel Labs Berkeley San Francisco, CA, USA, 15 June, 2010, ACM.

Chun, G. B., Ihm, S., Maniatis, P., & Naik, M., Patti, A. (2011). CloneCloud: Elastic Execution between Mobile Device and Cloud. EuroSys'11 Salzburg Austria, 10–13 April, 2011. ACM

Christensen, H. J. (2009). Using RESTful Web-Services and Cloud Computing to Create Next Generation Mobile Applications. OOPSLA 2009, Orlando, Florida, USA, 25-29 October, 2009. ACM.

Chu, S. F., Chen, C. K., & Cheng, M. C. (2011). Towards green cloud computing. Proceedings of the 5th International Conference on Ubiquitous Information Management and Communication, Seoul, Korea, 21-23 February, 2011.

Chung, G., Lai, & Ko. S. R. (2010) DISHES: A Distributed Shell System Designed for Ubiquitous Computing Environment. International Journal of Computer Networks & Communications (IJCNC), 2(1), January 2010.

Chunlin, L. & Layuan. L. (2010). Energy constrained resource allocation optimization for mobile grids. Journal of Parallel and Distributed Computing, 70(3), 245–258, 2010.

Cloud computing, http://en.wikipedia.org/wiki/ Cloud computing, Accessed on 16th June 2011.

Confidence Intervals and Sample Size http://highered.mcgraw-hill.com/sites/dl/free/0072549076/79745/ch07.pdf, Accessed on 25th December 2012.

Cuervo, E., Balasubramanian, A., Cho, K.D. Wolman, A., Saroiu, S., Chandra, R., & Bahlx. P. (2010). MAUI: Making Smartphones Last Longer with Code Offload MobiSys'10, San Francisco, California, USA. 15–18 June, 2010.

Cutsem, V. T., Dedecker, J., & Mostinckx, S. (2006). Ambient references: Addressing Objects in Mobile Networks. The Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications, Portland, Oregon, USA 22–26 October, 2006.

Dean, J., & Ghemawat. S. (2004). MapReduce: Simplified Data Processing on Large Clusters, Google, Inc.

Dey, A. K., & Abowd, G.D. (2000). The Context Toolkit: Aiding The Development Of Context-Aware Applications. Workshop on Software Engineering for Wearable and Pervasive Computing, Limerick, Ireland.

Dong, Y., Peng, J., Wang, D., Zhu, H., Wang, F., Chan, C. S., Mesnier, P. M. (2011). RFS – A Network File System for Mobile Devices and the Cloud. ACM SIGOPS Operating Systems Review archive, 45(1), January 2011.

Dou, A., Kalogeraki, V., Gunopulos, D., Mielikainen, T., & Tuulos, H. V. (2010). Misco: A MapReduce Framework for Mobile Systems, PETRA'10 Samos, Greece, 23 – 25 June, 2010ACM .

Dropbox http://www.dropbox.com. Accessed on 15th July 2011.

Elliot, K., Neustaedter, C., & Greenberg, S. (2005). Time, ownership and awareness: The value of contextual locations in the home. In UbiComp 2005: Ubiquitous Computing (pp. 251-268). Springer Berlin Heidelberg.

Fan, X., Cao, J., & Mao. H. (2011). A Survey of Mobile Cloud Computing ZTE Communications, 9 (1), 8-12, March 2011.

Fernando, N. Loke, W. S., Rahayu, W. (2012). Mobile cloud computing: A survey. Future Generation Computer Systems, 29(1), 84–106, January 2013.

Fortino, G., Mastroianni, C., Pathan, M., Vakali, A. (2009). Next Generation Content Networks: Trends and Challenges, UPGRADE-CN'09, Munich, Germany. ACM Press, 9 June, 2009.

Gao, Y., Fu, L., Zhang, Z., Luo, S., & Lu, P. (2011). A Case for Cloud Based Mobile Search ZTE Communications, 9(1), 37-40, March 2011.

Gartner (2011). Gartner Says Android to Command Nearly Half of Worldwide Smartphone, Press Release Egham, UK, April 7, 2011. http://www.gartner.com/it/page.jsp, Accessed on December 1st, 2011.

Google Docs http://docs.google.com Accessed on 15th July 2011.

Giurgiu, I., Riva, O., Juric, D., Krivulev, I., & Alonso. G. (2009). Calling the Cloud: Enabling Mobile Phones As Interfaces To Cloud Applications. Middleware'09 Proceedings of the ACM/IFIP/USENIX 10th International conference on Middleware pp. 83-102.

Goyal, S., Carter, J. (2004). A Lightweight Secure Cyber Foraging Infrastructure for Resource-Constrained Devices. WMCSA 2004 6th IEEE Workshop, 2-3 Dec, 2004.

Hong, Y. J., Suh, H. E., & Kim, J. S. (2009). Context-Aware Systems: A Literature Review and Classification. Expert Systems with Applications, 36(4), 8509–8522, May 2009.

Hoang, T., Dinh, Chonho. L., Dusit, N., & Ping, W. (2011). A Survey of Mobile Cloud Computing: Architecture, Applications, and Approaches. Accepted for publication in Wireless Communications and Mobile Computing , Wiley Publishers.

Huang, D., Zhang, X., Kang, M., & Luo, J. (2010). MobiCloud: Building Secure Cloud Framework for Mobile Computing And Communication, IEEE Computing Society.

Hung, H. S., Kuo, W.T., Shih, S.C., Sheih, P. J., Lee, P. C., Chang, W. C., & Wei. W. J. (2011). A Cloud Based Virtualized Execution Environment for Mobile Applications ZTE Communications 9(1), 19-25, March 2011.

Hung, H. S., Shih, S. C., Shieh, P. J., Lee, P. C., & Huang, H.Y. (2012). Executing mobile applications on the cloud: Framework and Issues. Computers and Mathematics with Applications, 63(2), 573–587, January 2012.

Iyer, R., Srinivasan, S., Tickoo, O., Fang, Z., Illikkal, R., Zhang, S., Chadha, V., Stillwell, M. P., & Lee, E. S. (2011). CogniServe: Heterogeneous Server Architecture For Large-Scale Recognition, IEEE MICRO, 31(3), 20-31, May-Jun 2011.

Jiang, J., Wu, Y., Huang, X., Yang, G., & Zheng, W. (2010). Online Video Playing on Smartphones: A Context-Aware Approach based on Cloud Computing. Journal of Internet Technology, 11(6), 821-827, November 2010.

Kangarlou, A., Gamage, S., Kompella, R. R., & Xu, D. (2010). vSnoop: Improving TCP Throughput in Virtualized Improving TCP Throughput in Virtualized Environments via Acknowledgement Offload. SC10 New Orleans, Louisiana, USA, IEEE Computer Society.

Kelenyi, I., & Nurminen, K. J. (2009) Bursty Content Sharing Mechanism for Energy, PM2HW2N '09 Proceedings of the 4th ACM workshop on Performance monitoring and measurement of heterogeneous wireless and wired networks.

Kelenyi, I., & Nurminen, K. J. (2010). CloudTorrent: Energy-Efficient BitTorrent Content Sharing for Mobile Devices via Cloud Services, IEEE Communications Society.

Klein, I., Mannweiler, C., Schneider, J., & Schotten. D. H. (2010). Access Schemes for Mobile Cloud Computing. 11th International Conference on Mobile Data Management, IEEE Computer Society.

Kloch, C., Petersen, B. E., & Madsen. B. O. (2011). Cloud Based Infrastructure, the New Business Possibilities and Barriers, April 2011. Springer.

kSOAP2 http:// http://ksoap2.sourceforge.net/, Accessed on 15th July, 2012.

Kovachev, D., Renzel, D., Klamma, R., & Cao. Y. (2010). Mobile Community Cloud Computing: Emerges and Evolves. 11th International Conference on Mobile Data Management, IEEE Computing Society.

Kovachev, D. & Klamma. R. (2012). Framework for Computation Offloading in Mobile Cloud Computing. International Journal of Interactive Multimedia and Artificial Intelligence, 1(7), 6-15.

Kumar, K., & Lu, H. Y. (2010). Cloud Computing For Mobile Users: Can Offloading Computation Save Energy. Computer, 43(4), 51-56, April 2010, IEEE Computer Society.

Lagar-Cavilla, A. H., Whitney, J., Scannell, A., Patchin, P., Rumble, M. S., Lara, E. D., Brudno. M., & Satyanarayanan. M. (2009). Snowflock: Rapid virtual machine cloning for cloud computing. In Eurosys.

Lagar-Cavilla, A. H., Whitney, J., Scannell, A., Patchin, P., Rumble, M. S., Lara, E. D., Brudno, M., & Satyanarayanan. M. (2011). "SnowFlock: *Virtual Machine Cloning as a First-Class Cloud Primitive* " ACM Transactions on Computer Systems, 29(1), February 2011.

Lai, C. C., & Ko, S. R. (2010). DISHES: A Distributed Shell System Designed for Ubiquitous Computing Environment. International Journal of Computer Networks & Communications, 2(1), January 2010.

Larus, R. J. (2011). Programming the Cloud, PPoPP'11 San Antonio, Texas, USA., February 12–16, 2011. ACM Press.

LaMarca, A. (2005). Self-Mapping in 802.11 Location Systems. In M. Beigl, S. Intille, J. Rekimoto & H. Tokuda (Eds.), Proceedings of UbiComp 2005: Ubiquitous Computing (Vol. 3660, pp. 87-104): Springer Berlin Heidelberg

Liang, H., Huang, D., Cai, X. L., Shen, S.X., & Peng. D. (2011). Resource Allocation for Security Services in Mobile Cloud Computing, IEEE Computing Society.

Liu, Q., Jian, X., Hu, J., Zhao, H., & Zhang, S. (2009). An Optimized Solution for Mobile Environment Using Mobile Cloud Computing, IEEE Computing Society.

Liu, J., Kumar, K., & Lu, H. Y. (2010). Tradeoff between Energy Savings and Privacy Protection in Computation Offloading, ISLPED'10, Austin, Texas, USA, 18–20 August, 2010, ACM Press.

Luo, X. (2009). From Augmented Reality to Augmented Computing: A Look at Cloud-Mobile Convergence. International Symposium on Ubiquitous Virtual Reality, IEEE Computing Society.

Iyer, R., Srinivasan, S., Tickoo, O., Fang, Z., Illikkal. R. Zhang. S., Chadha. V., Jr. S. M. P., & Lee. E. S. (2011). Cogniserve: Heterogeneous Server Architecture for Large-Scale Recognition. IEEE MICRO, 31(3), 20-31 June 2011.

MobileMe. http://en.wikipedia.org/wiki/MobileMe Mao, Accessed on 15th June 2011.

Mao, H., Xiao, N., Shi, W., & Lu, Y. (2010). Wukong: Toward a Cloud-Oriented File Service for Mobile Devices. 2010 IEEE International Conference on In Services Computing (SCC), (pp. 498-505) , July 2010.

Marinelli, E. E. (2009). Hyrax: Cloud Computing on Mobile Devices using MapReduce, ANSI Std Z3, 9-18 September 2009.

Mei, L., Chan, K. W., & Tse, H. T. (2008). A Tale of Clouds: Paradigm Comparisons and Some Thoughts on Research Issues. APSCC '08, 9-12 December, 2008, IEEE Computing Society.

Messer, A., Greenberg, I., Bernadat, P., Milojicic, D., Chen, D., Giuli, J.T., & Gu, X. (2002) Towards a Distributed Platform for Resource-Constrained Devices. Hewlett-Packard Company.

Mun, M., Hao, S., Mishra, N., Shilton, K, Burke, J., Estrin, D., Hansen., M., Govindan, R. (2010). Personal Data Vaults: A Locus of Control for Personal Data Streams, CoNEXT 2010 Philadelphia, USA, 30 November – 3 December, 2010 ACM Press.

Nagin, K., Hadas, D., Dubitzky, Z., Glikson, A., Loy, I., Rochwerger, B., & Schour, L.(2011) Inter-Cloud Mobility of Virtual Machines IBM Haifa Research Lab, ACM SYSTOR '11 Haifa, Israel, 30 May – 01 June, 2011.

Nguyen, B. C., Yoon, S. M., & Lee, H. K. (2006). Multi Bit Plane Image Steganography. International Workshop on Digital Watermarking, In Digital Watermarking, Springer Berlin Heidelberg, (pp.61-70).

OpenMobster http://code.google.com/p/openmobster/, Accessed on 24th June, 2011.

Oracle VMTemplates http://www.oracle.com/technology/products/vm/templates/index.html, Accessed on 30th July, 2011.

Oh, J., Lee. S., & Lee. E. (2006). An Adaptive Mobile System Using Mobile Grid Computing In Wireless Network, Proceedings of the 6th International Conference on Computational Science and its Applications (ICCSA 2006), Glasgow, UK, 8-11 May (49–57), 2006.

Owens. D. (2010). Securing Elasticity in the Cloud, Communications of the ACM Press 53.

Holman, R. (2010). Mobile Cloud Computing: *$9.5 billion by 2014* URL http://www.juniperresearch.com/analyst-xpress-blog/2010/01/26/mobile-cloud-application-revenues-to-hit-95-billion-by-2014-driven-by-converged-mobile-services/, Accessed on 18th August 2011.

Prosper Mobile Insights, Smartphone/tablet user survey (2011). URL http://prospermobileinsights.com/Default.aspx?pg=19, Accessed on 20th July, 2011.

Protecting Portable Devices: Physical Security, http://www.us-cert.gov/cas/tips/ST04-017.html, Accessed on 15th September, 2012.

Sanaei, Z., Abolfazli, S., Gani, A., & Khokhar. H. R. (2012). Tripod of Requirements in Horizontal Heterogeneous Mobile Cloud Computing. Proceedings of the 1st International Conference on Computing, Information Systems and Communications. Singapore, 11-13 May, 2012.

Satyanarayanan, M. (2001). Pervasive computing: Vision and challenges. IEEE Personal Communications, 8(4), 10–17.

Satyanarayanan, M., Bahl, P., & Caceres, R. (2009). The Case for VM-Based Cloudlets in Mobile Computing, December, 2009, IEEE Computing Society.

Satyanarayanan, M. (2010). Mobile Computing: The Next Decade, ASP-DAC '08: Proceedings of the 2008 Asia and South Pacific Design Automation Conference, June 2010, IEEE Computer Society Press.

Sokol, K., Andrius, A., Pan, H., Richard, M., & Xinwen, Z. ThinkAir: Dynamic resource allocation and parallel execution in cloud for mobile code offloading, INFOCOM, 2012 Proceedings IEEE, Orlando, 25-30 March (945- 953), 2012.

Shanklin. M. (n.d.) Mobile Cloud Computing http://www.google.com/mcc.html, Accessed on 16th June, 2011.

Sharifi, M., Kafaie, S., Kashefi, O. (2011). A Survey and Taxonomy of Cyber Foraging of Mobile Devices, IEEE Communications Surveys Tutorials, 14(4), 1232-1243, November 2011.

Shiraz, M., Gani, A., & Rashid, H. K. (2011). Towards Lightweight Distributed Applications in Mobile Cloud Computing, Proceedings of 2012 IEEE International Conference on Computer Science and Automation Engineering, 25-27 May, 2012, Zhangjiajie, China.

Shiraz, M., Gani, A., Khokhar, H. R., & Buyya, R., (2012). A Review on Distributed Application Processing Frameworks in Smart Mobile Devices for Mobile Cloud Computing, IEEE Communications Surveys & Tutorials, 15(3), 1294 – 1313, November 2012.

Shiraz, M., Abolfazli, S., Sanaei, Z., & Gani, A. (2013a). A Study on Virtual Machine Deployment for Application Outsourcing in Mobile Cloud Computing, The Journal of Supercomputing, 63(3), 946-964, March 2013.

Shiraz, M., Ahmed, E., Gani, A., & Han, Q. (2013b). Investigation on Runtime Partitioning of Elastic Mobile Applications for Mobile Cloud Computing, Accepted for publication in Journal of Supercomputing, DOI:10.1007/s11227-013-0988-6, August 2013.

Smartphone URL:http://en.wikipedia.org/wiki/Smartphone, Accessed on 20th July 2011.

Subashini, N. S., & Kavitha. V. (2010). A Survey on Security Issues in Service Delivery Models of Cloud Computing, Journal of Network and Computer Applications, 34(1),1–11, June 2010.

Sud, S., Want, R., Pering. T., Lyons, K., Rosario, B., Gong, X. M. (2012). Dynamic Migration of Computation Through Virtualization of the Mobile Platform, Mobile Networks and Applications, 17(2), 206–215, February 2011.

Sun Micro Systems (2009) Introduction to Cloud Computing Architecture White Paper 1st Edition, June 2009.

Tso, P. F., Cui, L., Zhang, L., Jia, W. (2011). Building a Platform to Bridge Low End Mobile Phones and Cloud Computing Services ZTE Communications, 9(1), 26-30 March 2011.

Vartiainen, E., Mattila,V. V. K. (2010). User Experience of Mobile Photo Sharing in the Cloud, MUM '10 Limassol, Cyprus, ACM Press, December 1-3, 2010.

De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Paraboschi, S., Pelosi, G., & Samarati, P. (2008). Preserving Confidentiality of Security Policies in Data Outsourcing. Proceedings of the 7th ACM workshop on Privacy in the electronic society October (pp. 75-84) 2008.

Wang, K., Rao, J., Xu, & Z. C. (2011), Rethink the Virtual Machine Template VEE'11, Newport Beach, California, USA., March 9–11, 2011, ACM.

Weiss, A. (2007) Computing in The Clouds, netWorker , 11 ACM.

What are Smartphones http://cellphones.about.com/od/smartphonebasics/a/-what_is_smart.html. Accessed on 20th July 2011.

What is Smartphones URL:
http://cellphones.about.com/od/smartphonebasics/a/what_is_smart.html, Accessed on 20th July, 2011.

What-Makes-Smart-Phones-Smart, URL: http://www.samsung.com/.../what-makes-smart-phones-smart, Accessed on 20th July, 2011

Wu & Tsai (2003). A Steganographic Method for Images by Pixel-value Differencing. In Pattern Recognition Letters, pp.1613-626.

Zao, B., Xu, Z., Chi, C., Zhu, S. & Cao, G. (2011). Mirroring Smartphones for Good: A Feasiblity Study. ZTE Communications 9(1), 13-18, March 2011.

Zaplata, S., & Lamersdorf, W. (2010). Towards Mobile Process as a Service SAC'10 Sierre, Switzerland, 22-26 March, 2010 ACM.

Zheng, W., Xu, P., Huang, X. (2010). Design A Cloud Storage Platform For Pervasive Computing Environments, Cluster Computing-The Journal of Networks Software Tools and Applications, 13(2), 141-151, June 2010.

Zhang, X., Schiffman, J., Gibbs, S., Kunjithapatham, A., & Jeong, S. (2009). Securing Elastic Applications on Mobile Devices for Cloud Computing. CCSW'09, Chicago, Illinois, USA. November 13, 2009, ACM.

Zhang, X., Kunjithapatham, A., Jeong., S., & Gibbs. S. (2011). Towards an Elastic Application Model for Augmenting the Computing Capabilities of Mobile Devices with Cloud Computing, Mobile Networks & Applications, 16(3), 270-285, June 2011.