Scalable Resource Augmentation for Mobile Devices

by

Manjinder Paul Singh Nir

A thesis submitted to the

Faculty of Graduate and Postdoctoral Affairs

in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

in

Electrical and Computer Engineering

at the

Ottawa-Carleton Institute for Electrical and Computer Engineering

Department of Systems and Computer Engineering

Carleton University

Ottawa, Ontario, Canada.

December 12, 2014

©2014 Manjinder Paul Singh Nir

Abstract

This thesis focuses on scalability of a resource augmentation environment when a large number of mobile devices and multiple service nodes are present. To deal with congestion, a scanning method was proposed to get information on users' density in an area such that the service nodes and access points could be placed at strategic points. To lower communication overhead, a centralized broker-node architecture was proposed, which manages resource monitoring on behalf of all mobile devices. In the centralized architecture, mathematical models for the task scheduling problem in the local resources case and the mobile cloud computing case were proposed to optimally minimize the total energy consumption across all mobile devices. A generalized model for the task scheduling problem was proposed. The model optimally minimized the total energy and monetary cost when evaluated in two environments for mobile cloud computing, one using a local private cloud and the other using public clouds. The models found optimal solutions for the centralized task scheduling problems, and an improvement in the total costs was observed when offloading with optimization compared to when offloading without optimization using the centralized task scheduler.

Acknowledgements

First and above all, I praise Almighty God, for providing me this opportunity and granting me the capability to proceed successfully. I am thankful for the wisdom and perseverance that He has been bestowed upon me during this research work.

I would like to express my sincere gratitude to Prof. Ashraf Matrawy, my research supervisor, for giving me the opportunity to do research. I am very thankful to him for the continuous support of my Ph.D. study and research, for his patience and motivation, and for providing invaluable guidance throughout this research.

I would like to express my sincere gratitude to Prof. Marc St-Hilaire for helping me with the optimization work. I greatly appreciate the generous financial support I got from Carleton University.

Last but not the least, I would like to thank my parents; their love provided me inspiration and was my driving force. Finally, and most importantly, I would like to thank my wife Manpuneet Kaur, my children Kunwar and Chandan. Their support, sacrifices, and love were instrumental in finishing my research work.

To my parents

for their love and endless support

To my wife

for giving me hope and encouragement

Contents

Abstract	ii
Acknowledgements	iii
Table of Contents	v
List of Tables	xii
List of Figures	xiii
List of Acronyms	xvi
1 Introduction	1
1.1 Contributions	3
1.2 Thesis Outline	7

2	Background	9
2.1	Resource Augmentation	11
2.1.1	Enabling Technologies	12
2.1.2	Applications	14
2.1.3	Challenges	16
2.2	Existing Resource Augmentation Environments	18
2.2.1	Cyber Foraging	18
2.2.2	Cloud Computing	19
2.2.3	Mobile Cloud Computing	21
2.2.4	Challenges of using Cloud Computing for Resource Augmentation	21
2.3	Task Scheduling	23
2.3.1	Task Offloading Decision	23
2.3.2	Task Offloading Performance	25
2.3.3	Task Offloading Methods	31
2.4	Task Offloading Algorithms	33
2.4.1	Algorithms Based on Change in Resources	33
2.4.2	Algorithms Based on Execution or Response Time	35

2.4.3	Algorithms Based on Network Parameters	36
2.5	Related Work	37
2.5.1	Resource Monitoring	38
2.5.2	Task Scheduling	39
3	Problem Statement	41
3.1	Research Question	42
3.2	Research Objectives	44
4	Mapping Mobile Devices in the Local Area Network	46
4.1	Related Work	49
4.2	Scanning & Mapping the Area	51
4.3	Simulation & Results	57
4.4	Conclusion	62
5	Broker Assisted Centralized Management	64
5.1	Proposed Framework for Centralized Management of Resource Moni- toring	66
5.2	The Hybrid Simulation & Emulation Experimental Setup	73
5.3	Evaluation	75

5.3.1	Simulation Experiments	75
5.3.2	Analysis of Results	77
5.4	Conclusion	81
6	Energy Optimization: The Local Resources Case	83
6.1	Task Scheduler Model	85
6.1.1	Assumptions	86
6.1.2	Notation	87
6.1.3	Cost Function	89
6.1.4	The Model	90
6.2	Deriving Evaluation Settings	92
6.2.1	Execution Times & I/O Data Sizes	93
6.2.2	Remote Completion Time	94
6.2.3	Energy Consumption	96
6.2.4	Processing Power & Memory	98
6.3	Results	100
6.3.1	Analysis of Results	102
6.4	Conclusion	104

7	Energy Optimization: The Mobile Cloud Computing Case	106
7.1	Task Scheduler Model	107
7.1.1	Assumptions	108
7.1.2	Notation	109
7.1.3	Cost Function	111
7.1.4	The Model	112
7.2	Deriving Evaluation Settings	114
7.2.1	Task Execution Times & I/O Data Sizes	115
7.2.2	Delay Tolerance	116
7.2.3	Data Rate	116
7.2.4	Energy Consumption	117
7.3	Results & Analysis	118
7.3.1	Effect of the Number of Virtual Machines	120
7.3.2	Effect of Input & Output Data Sizes	122
7.3.3	Effect of Delay Tolerance	124
7.4	Conclusion	126

8 A Generalized Model for Energy and Monetary Cost Optimization127

8.1	Resource Augmentation Environments	128
8.1.1	Using a Centralized Broker	131
8.2	Task Scheduler Model	132
8.2.1	Assumptions	132
8.2.2	Notation	134
8.2.3	Cost Function	137
8.2.4	The Model	139
8.3	Deriving Evaluation Settings	142
8.3.1	VM Instance Types & Monetary Costs	143
8.3.2	Task Execution Times & I/O Data Sizes	145
8.3.3	Delay Tolerance Parameter	146
8.3.4	Data Rate	147
8.3.5	Energy Consumption	147
8.4	Results & Analysis	149
8.4.1	Infinite & Finite Resources	153
8.4.2	Offloading with & without Optimization	155

8.4.3	Input & Output Data Sizes	158
8.4.4	Delay Tolerance	160
8.5	Summary of Results & Discussions	166
8.6	Conclusion	167
9	Conclusion & Future Work	169
	Bibliography	173

List of Tables

6.1	Local Execution Time Ranges	94
6.2	Data Rates at different Data Sizes	95
6.3	Power Ratings of WiFi Radio & CPU at different states	97
6.4	Values of p_{ms} and k_{ms} for a low execution time task	99
6.5	Values of p_{ms} and k_{ms} for a medium execution time task	99
6.6	Values of p_{ms} and k_{ms} for a heavy execution time task	99
8.1	Instance Types: Configuration & Speed-up Factor	143
8.2	Monetary costs for different cloud resources	144

List of Figures

4.1	An example of an area map showing users' density. The numbers on the x axis and the y axis show the area dimension units.	49
4.2	Process for a mobile device to access services from a server.	53
4.3	Scanning the area.	54
4.4	Mobility model of a probing server during the scanning process. . . .	55
4.5	Different combinations of node concentration in the three blocks. . . .	58
4.6	Percentage of detected nodes at different node concentrations.	61
5.1	Baseline scenario.	68
5.2	Centralized broker scenario.	68
5.3	TCP handshaking and resource description file request/response protocol.	69
5.4	Service model of <i>ServerApp()</i> service.	70

5.5	Two Linux containers representing a service node and a mobile device connected through <i>ns-3</i> WiFi network.	72
5.6	Effect of number of servers in the <i>serverApp()</i> service on resource monitoring time.	78
5.7	(a) Comparison of resource monitoring time and scalability, (b) collisions in the WiFi channel, in baseline and broker scenarios.	79
6.1	Graphical representation of the task scheduler model.	86
6.2	Total energy consumption across all mobile devices.	103
7.1	Resource augmentation environment for MCC.	108
7.2	Total energy consumption across all mobile devices.	121
7.3	Total energy consumption across all mobile devices.	123
7.4	The effect of delay tolerance (λ_m) on the total energy consumption. .	125
8.1	Resource augmentation environments for MCC.	130
8.2	The effect of finite and infinite resources on the percentage saving in the total energy consumption when offloading with optimization. . . .	154
8.3	The total energy consumption when offloading with and without optimization in RAE using a local private cloud.	156

8.4	The total energy consumption and the total monetary cost when offloading with and without optimization in RAE using public clouds. .	157
8.5	Percentage saving in the total energy consumption when offloading with optimization at different data sizes, in both RAEs.	159
8.6	Selecting different cloud providers for data intensive tasks based on monetary costs in RAE using public clouds.	161
8.7	The effect of delay tolerance on the total energy consumption when data sizes (in MB) are in a range $\mathcal{U}(0, 40)$	162
8.8	The effect on the total energy consumption, the total monetary cost, and the number of offloaded tasks, when only <i>small</i> or <i>small, large</i> , and <i>xlarge</i> VM instances are available in RAE using public clouds. . .	163

List of Acronyms

3G	Third Generation
4G	Fourth Generation
5G	Fifth Generation
AP	Access Point
CFS	Cyber Foraging System
CPU	Central Processing Unit
CSMA	Carrier Sense Multiple Access
EC2	Elastic Compute Cloud
FCFS	First Come First Serve
GB	Giga Byte
GPS	Global Positioning System
HTTP	Hypertext Transfer Protocol
IaaS	Infrastructure-as-a-Service
I/O	Input/Output
ILP	Integer Linear Program

JVM	Java Virtual Machine
KVM	Kernel-Based Virtual Machine
LXC	Linux Containers
LTE	Long-Term Evolution
LAN	Local Area Network
MCC	Mobile Cloud Computing
MB	Mega Byte
MAC	Media Access Control
NIST	National Institute of Standards & Technology
ns-2	Network Simulator 2
ns-3	Network Simulator 3
OS	Operating System
PaaS	Platform-as-a-Service
RTOS	Real-Time Operating System
RAM	Random Access Memory
RPC	Remote Procedure Call
RMI	Remote Method Invocation
RAE	Resource Augmentation Environment
SaaS	Software-as-a-Service
SSID	Service Set Identification
SOAP	Simple Object Access Protocol

SN	Service Node
TCP	Transmission Control Protocol
TSP	Task Scheduling Problem
VM	Virtual Machine
UPnP	Universal Plug and Play
WAN	Wide Area Network
WLAN	Wireless Local Area Network
XML	Extensible Markup Language

Chapter 1

Introduction

Modern mobile devices have sensing and advanced computing capabilities, and wireless network technologies have enabled mobile users to move about with computing power and network resources. Nowadays, mobile devices can support a wide range of heavy applications such as image manipulation [65], [76], [52], [87], language translation [21], [42], [64], and speech synthesis, etc. These resource intensive applications require rich computational resources on mobile devices. Despite advancements in computing hardware and communication capabilities, mobile devices have resource constraints intrinsic to their size and weight [22]. Consequently, the available computing power, memory capacity, or battery energy are not enough for resource intensive applications [95]. Thus, mobile devices either cannot run these applications, or, even if able to run them, find that the required application fidelity cannot be achieved, and/or that the battery will not last as long compared to normal usage.

A number of resource enhancement approaches have been proposed for mobile devices to support resource intensive applications [81], [19], [93]. Offloading heavy applications to resource rich computers (i.e. service nodes) for remote execution is one of the solutions for resource augmentation of mobile devices [36], [38], [20]. In this research work, resource augmentation through task offloading is considered in an environment for a large number of mobile devices. The motivation for this type of environment is that most existing works on task offloading [36], [38], [43], [55], [71], [61] consider a single or a few mobile devices and service nodes in their scenario. However, there could be places at which a large number of mobile devices seek task offloading, such as a conference hall, a festival site, etc.

In this research work, the focus is on the challenges posed due to the presence of and the task offloading by a large number of mobile devices. The presence of a large number of mobile devices could cause congestion in the wireless network and the service nodes. The congestion in the wireless network may reduce the availability of the minimum required bandwidth for mobile devices to offload their tasks [36], [42], [38]. Also, the service nodes may not have enough CPU power, memory capacity, or storage resources for all mobile devices in the area. On the other hand, having a large number of mobile devices also poses challenges when task offloading is performed using dynamic task scheduling policies. Generally, task offloading is beneficial if the remote execution cost is less than the local execution cost at the mobile device [69], [75]. Task offloading systems use dynamic policies when deciding for offloading such

that the offloading is beneficial for the task and the mobile device. Therefore, when using dynamic task scheduling policies, mobile devices have to repeatedly contact the service nodes to obtain the up-to-date status of currently available resources [69], [75]. However, in the case of a large Resource Augmentation Environment (RAE), repeated resource monitoring of multiple service nodes by a large number of mobile devices can cause communication overhead in the wireless network [78]. This situation will cause delays for the mobile devices that are waiting to get the up-to-date status of resource availability from multiple service nodes through the congested wireless network.

1.1 Contributions

In this research work, resource augmentation of mobile devices through task offloading is considered in an environment having a large number of mobile devices and multiple service nodes. In this environment, the objective is to investigate and reduce the congestion and communication overhead caused by the presence of and task scheduling by a large number of mobile devices. A scanning method is presented for the placement of service nodes and Access Points (APs) in an area according to the density distribution of the users. The aim of this approach is to reduce congestion created due to the presence of a large number of mobile devices. Further, a centralized architecture for a large RAE is proposed: (i) to reduce the communication overhead due to repeated resource monitoring performed by a large number of mobile devices,

and (ii) to handle task scheduling on behalf of all mobile devices in the system. An overview of these contributions is as follows.

- The first contribution is to deal with congestion in the wireless network and reduction in the availability of minimum required bandwidth due to the presence of a large number of mobile devices. The approach to deal with the challenge is to place service nodes and wireless APs at strategic points in the area. Towards this approach, a scanning method is proposed that can map the density distribution of users in an area. The intended area mapped with the density distribution of users could help in identifying the strategic points for placing wireless APs and service nodes. The scanning setup uses a WiFi network simulated through network simulator *ns-3* and cyber foraging to get the position information from the mobile devices. Using this setup, there is no need to rely on a pre-installed infrastructure of APs or an already prepared database of the subject area. Therefore, this method can provide location information in any unprepared (random) area, which is the main limitation of the position methods using pre-installed infrastructure [39], [37]. The results of this work are presented in Chapter 4 and were published at the following conference.

- Manjinder Nir and Ashraf Matrawy, “A Density Mapping Algorithm for Supporting Cyber Foraging Service Networks”, in Proceedings of the *7th International Conference on Broadband, Wireless Computing, Communication, and Applications (BWCCA) - 14th International Symposium on*

Multimedia Network Systems and Applications (MNSA), IEEE Computer Society, Victoria, BC, Canada, 2012, pp. 578-583.

- The communication overhead caused when resource monitoring of multiple service nodes is repeatedly performed by a large number of mobile devices is investigated. To lower the communication overhead, a centralized broker-node framework for a large RAE is proposed. The centralized broker-node manages resource monitoring on behalf of all mobile devices in the system. This centralized management helps in lowering the communication overhead, and in turn, the resource monitoring time is lowered for the mobile devices. The centralized framework is a hybrid simulation and emulation setup using Linux Containers (LXC) and network simulator *ns-3*. In this setup, Java multi-threading is used to perform resource monitoring by a large number of mobile devices. The results of this work are presented in Chapter 5 and were published at the following conference.
 - Manjinder Nir and Ashraf Matrawy, “Centralized Management of Scalable Cyber Foraging Systems”, in Proceedings of the *4th International Conference on Emerging Ubiquitous Systems and Pervasive Networks (EUSPN)*, Elsevier Procedia Computer Science, vol. 21, Niagara Falls, ON, Canada, 2013, pp. 265-273.
- The proposed centralized broker-node approach is further utilized to handle

task scheduling on behalf of all mobile devices in the system. A mathematical formulation for the centralized task scheduling problem is modeled for the local resources case and the Mobile Cloud Computing (MCC) case. The task scheduler models, in both the cases, optimally solve the task scheduling problem with the aim to minimize the total energy consumption across all mobile devices. The main difference between the two models is that unlimited computing resources (Virtual Machines (VMs)) are assumed to be available in the MCC case, in contrast to limited computing resources in the local resources case. The integer linear program of the task scheduling problem is implemented and evaluated using IBM's linear programming solver called CPLEX. The results for both the models are presented in Chapters 6 and 7, respectively, and were published at the following conferences.

- Manjinder Nir, Ashraf Matrawy, and Marc St-Hilaire, “Optimizing Energy Consumption in Broker-Assisted Cyber Foraging Systems”, in Proceedings of the *28th International Conference on Advanced Information Networking and Applications (AINA)*, IEEE, Victoria, BC, Canada, 2014, pp. 576-583.
- Manjinder Nir, Ashraf Matrawy, and Marc St-Hilaire, “An Energy Optimizing Scheduler for Mobile Cloud Computing Environments”, in proceedings of the *33rd Annual International Conference on Computer Communications (INFOCOM) - Workshop on Mobile Cloud Computing (MCC)*, IEEE, Toronto, ON, Canada, 2014, pp. 404-409.

Further, the previous task scheduler models are extended to a generalized task scheduler model. This model is evaluated in two RAEs for mobile cloud computing. In the first environment, service nodes are available from a local private cloud accessible through a WiFi network. However, in the second environment, service nodes are available from public clouds accessible through the Internet. Compared to the previous models, this model can deal with multiple tasks to offload per mobile device, and the monetary cost is considered when using resources from public clouds. More precisely, the generalized task scheduler model is constructed to find an optimal solution with the aims: (i) to minimize the total energy consumption when applied to an environment using a local private cloud, and (ii) to minimize the total energy consumption and the total monetary cost when applied to an environment using public clouds. The results of this work are presented in Chapter 8.

1.2 Thesis Outline

The thesis is organized into nine chapters. In Chapter 2, background work on resource augmentation for mobile devices is presented. The problem statement and the research objectives for this work are defined in Chapter 3. The density mapping algorithm to find the density distribution of mobile devices in an area is presented in Chapter 4. A framework for broker-assisted centralized management of resource monitoring is presented in Chapter 5. In Chapter 6, a task scheduler model for the

local resources case is presented, followed by a task scheduler model for the mobile cloud computing case in Chapter 7. A generalized task scheduler model for optimizing energy and monetary cost is presented in Chapter 8. The thesis is concluded in Chapter 9.

Chapter 2

Background

In 1991, Mark Weiser introduced [101] the concept of Ubiquitous Computing, commonly known as Pervasive Computing. In his vision, “The most profound technologies are those that disappear. They weave themselves into a fabric of everyday life until they are indistinguishable from it”. A decade later Satyanarayanan [92] elaborated the vision of pervasive computing as, “the creation of environments saturated with computing and communication capability, yet gracefully integrated with human users”. According to the author, a pervasive computing environment has plenty of computing and network resources, and should support the mobility of users to enable these resources to be part of their everyday lives, such that the technology can disappear from the users’ consciousness.

Advancements in computing hardware and wireless network technologies have enabled mobile users to move about with good computing power and network resources.

Nowadays, mobile devices are in abundance, and people are increasingly using them for a wide range of tasks. The demand for running heavy applications is increasing at a fast pace [5]. For example, present day mobile devices can support resource intensive applications such as image manipulation [65], [76], [52], [87], optical character recognition [21], language translation [21], [42], [64], video recordings (e.g. as in crowd computing [94]), speech synthesis, augmented reality, real-time gaming applications, and wearable computing for cognitive applications [95], etc.

The resource intensive applications require rich computational resources from mobile devices. The computing power and battery life of mobile devices are increasing day-by-day [22]. However, the intrinsic mobility characteristics of mobile devices put limits on their size and weight. Consequently, mobile devices may not have enough available computing power, memory capacity, or battery energy [95] for resource intensive applications. Thus, mobile devices either cannot run these applications, or even if they are able to run them, the required application fidelity cannot be achieved, and/or their battery will not last as long compared to normal usage.

A number of resource enhancement approaches have been proposed to enable mobile devices to support resource intensive applications. One approach is to compromise with the application fidelity, when enough resources are not available on the mobile device itself [81], [19]. In this approach there is a trade-off between the resource consumption and the application fidelity. Another approach is to re-write applications for resource constrained devices [44], [93]. Another approach enables mobile devices

to save only battery energy by executing applications at remote locations [90], [81], [19], [93]. Cyber foraging [92] technology introduces a RAE that enables mobile devices to find available computing nodes in the surrounding area. Thus, mobile devices can augment their resources temporarily by remotely executing (i.e. task offloading) their tasks on the surrounding computing nodes.

In this research work, the focus is on resource augmentation of mobile devices through task offloading. In the following sections, the background related to resource augmentation through task offloading and our research direction is discussed.

2.1 Resource Augmentation

Resource augmentation through task offloading is an approach whereby a mobile device can offload its resource intensive application to a rich computing node. The computing node executes the offloaded task on behalf of the mobile device. Therefore, task offloading alleviates mobile devices from resource constraints by enhancing their CPU power and memory capacity, and by saving battery energy. Mobile devices can opt for task offloading when their available resources are not adequate either to execute their tasks or to achieve the desired performance (e.g. short execution time and/or low battery energy requirements) for the tasks.

In a RAE, computing nodes, referred to as service nodes, are available to mobile devices for task offloading. The service nodes are rich in computing resources, and

provide their CPU power, memory, storage, file system, etc., to mobile devices. Mobile devices can access the resources of single [42] or multiple service nodes [66], [85]. In an environment, location and mobility are the two characteristics of service nodes with respect to the mobile devices. The service nodes can be located in the vicinity of mobile devices using short range wireless technologies (i.e. WLAN) [92], [95], [43]; on the other hand, these can also be located at Internet/WAN latency using broadband wireless technologies (4G, 5G, LTE, etc.). Most of the existing works on resource augmentation use static service nodes [95], [96], [43], [71], in a designated area, while some use mobile service nodes [84], [52], [26]. The mobility characteristic of service nodes poses some challenges. For example, challenges include maintaining a connection between a mobile device and its currently serving mobile service node, or saving/transferring the current state of running tasks from a service node to another service node, etc., [43], [96], [95], [34].

2.1.1 Enabling Technologies

Wireless Technologies

Advancements in wireless technologies have made the idea of pervasive computing a reality. Mobile devices use wireless technologies such as WiFi, 4G, 5G, LTE, etc., to access service nodes and offload tasks. Among other factors, the liveliness of a RAE also depends on the wireless technology used. Different wireless networks have

different bandwidth and network latency. For example, tasks that require high responsiveness can not be offloaded using long latency wireless networks. Similarly, data intensive tasks can not be offloaded using low data rate wireless networks. Therefore, while offloading a task the selection of a wireless technology depends on the type of the task and the goals of offloading the task.

Mobile Agents

A mobile agent is a piece of code (process) that can transport its state from one environment to another. Thus, mobile agents facilitate the migration of tasks from mobile devices to remotely located service nodes [63], [54], [104]. The platform independent languages like Java and XML have motivated the design of mobile agents. Therefore, mobile agents can be migrated across heterogeneous platforms. On the other hand, a mobile client, which is an interface, is used to access service nodes and offload tasks. A mobile client can be thin or thick. A thin mobile client does no computation on the offloading task. It is meant for connecting to a service node through a network. Thus, it either uses a pre-installed task on the service node or it offloads the entire task to the service node based on static offloading policies [43], [18], e.g. Google's Gmail for mobile devices, or Facebook's location aware service, etc. On the other hand, a thick mobile client has the capability of doing computations on tasks before offloading. Using dynamic offloading policies it can do dynamic partitions of the tasks and can make dynamic task scheduling decisions [20], [42].

Virtualization Technology

Virtualization technology is also an enabling technology for task offloading. Virtualization separates the physical and the virtual resources of a physical machine. Thus, multiple operating systems can be run simultaneously on the same physical machine. Moreover, different users can be isolated from each other while allowing them to share the same infrastructure [12], [24], [77], [89]. Virtualization is a key technology for building an elastic resource provisioning infrastructure in cloud computing environments. The elastic nature of infrastructure resources enables cloud providers to efficiently use the under-utilized resources at their data centres, and also allows their users to temporarily utilize the computing infrastructure over the network. In other words, virtual resources can be allocated faster and dynamically depending on the users' demand.

2.1.2 Applications

Augmented mobile devices can provide a broad spectrum of applications such as in medical science, security systems, personal assistance, entertainment, etc.

- **Medical Science:** Doctors with hand held devices can have ubiquitous access to the centrally placed data of patients. The complex computations on a patient's data can be done on a powerful remote server through a mobile device, and the patient can take instantaneous treatment or suggestions.

- **Security Systems:** The security agents can have ubiquitous access to secure data. For example, at airports or any other sensitive areas, the security agents can do image processing and recognition of wanted culprits right at that place.
- **Disaster Management:** Natural calamities like earthquakes, hurricanes, flooding, tsunami, etc., sometimes cause a complete destruction of the communication system in an area. The disaster management team can have mobile devices to ascertain the conditions in a disaster area, and can process their data at remote servers.
- **Entertainment:** A RAE can enable mobile users to play games, stream videos, etc., anywhere and on-the-go. For example, users can stream music collection to iPhone or Android through Ubuntu One [9]. Similarly, Google Apps [2] provides services in education, business, marketplace, and government, etc. Thus, the battery of a mobile device is drained minimally even when running heavy applications through remote servers.
- **Cognitive Applications:** When using cognitive applications (e.g. language translator, face recognition, etc.) on mobile devices, the users desire a response which is comparable to their normal cognitive capabilities. Therefore, for better user experiences the responsiveness of cognitive applications should be high. Task offloading can be exploited to use cognitive applications ubiquitously by executing them remotely on computing nodes. The RAEs in which service nodes

are available with short network latencies (e.g. WiFi networks) can serve users for cognitive applications in a better way by providing high responsiveness.

2.1.3 Challenges

Resource augmentation through task offloading poses challenges for mobile devices. The various challenges could be; the availability of adequate resources from service nodes and network; adaptation to a change in the availability of resources at service nodes and the requirement of resources by the offloading tasks; and heterogeneity between the resources of mobile devices and service nodes, etc.

Availability and Requirement of Resources

The foremost challenge [92] of task offloading is the mismatch between the requirement for resources at the offloading mobile devices and the availability of the resources at the available service nodes. In this context, the author proposed three strategies to use when the availability of resources at the mobile device and the service nodes is less than the requirement at the mobile device. The three strategies include: (i) the client can guide the task to use fewer resources, (ii) the client can ask for a guarantee of adequate resources, and (iii) the client can suggest a corrective action to the mobile device. Apart from the intrinsic reason of mismatching between the availability and the requirement of resources, another reason is a variation in the available resources due to a change in the environmental conditions [42] such as a change in the location

of service nodes. In this situation, the wireless network at the new location could have less bandwidth and signal strength than the network at the old location. Therefore, a mobile client should be able to adapt to the availability and requirement of resources and should be able to take an appropriate decision for the task execution.

Heterogeneity in Resources

Heterogeneity [49] in the resources between the offloading mobile devices and the available service nodes is another challenge for task offloading. There could be heterogeneity in an application run time environment (JVM, .NET, C#, etc.) or operating system (Linux, Windows, Android, etc.). On the other hand, adaptation to diverse platforms can reduce the offloaded task's fidelity.

Network Parameters

The network latency between a mobile device and a service node depends on the network type (e.g. WiFi, 4G, 5G, LTE, etc.) and the service node's location in the network [36], [95]. When using public clouds, the Internet/WAN latency to the resources incurs high communication cost, which could be critical for data intensive tasks [95], [75]. In general, merely offloading a resource intensive task onto a service node might not benefit the mobile device. For example, remote execution may reduce the execution time of a task; however, data transfer during the remote execution may consume more battery energy of the mobile device than during the local execution.

Task Partitioning and Offloading Decision

Effective task partitioning and offloading decisions are among the challenges of task offloading. There are partitioning challenges based on the availability of computation resources at the mobile device and the service nodes [41], [42]; and offloading decision challenges are based on the input/output data of the task and the currently available network resources [36]. Another partitioning challenge is the selection of a programming model for coarse-grained [111] and fine-grained partitioning [33].

Mobility of Mobile Devices

In a RAE that uses local area networks (e.g. WLAN), the short range of the networks puts mobility restrictions on the mobile devices. In this case, if a mobile device moves to a new location outside the network then it has to find a service node in its new neighbourhood to keep its application running. This poses a further challenge in that, at the new location, the service nodes may or may not have support for the mobile device's application.

2.2 Existing Resource Augmentation Environments

2.2.1 Cyber Foraging

Cyber foraging technology enables mobile devices to opportunistically use computing nodes, called surrogate nodes, present in the vicinity, for task offloading [92]. The

surrogate nodes can be accessed using a low latency wireless technology (e.g. WLAN). The opportunistic use means that mobile devices could use the resources of surrogate nodes when these nodes are available and are willing to share their resources. A number of Cyber Foraging Systems (CFS) exist [38], [36], [56], [55]. Using cyber foraging, mobile devices can discover unknown surrogate nodes in the vicinity, and can automatically establish service contact with them using a service discovery technique. For example, some service interfaces are based on Remote Procedure Calls (RPC) [112], or Salutation [8], and some discovery protocols use downloadable code, e.g. in Jini [4]; some protocols transfer data, e.g. in Universal Plug and Play (UPnP) [10], using XML, SOAP, or HTTP, etc.

2.2.2 Cloud Computing

According to NIST [73], cloud computing is defined as, “a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g. networks, servers, storage, applications, services, etc.) that can be rapidly provisioned and released with minimal management effort or service provider interaction.” A cloud computing environment manages a large pool, commonly called a ‘*cloud*’, of computing, data processing/storage, and hardware resources, as compared with a few service nodes in a non-cloud environment [35], [69], [57].

The deployment models for cloud computing are broadly classified as; private cloud, public cloud, and hybrid cloud. Infrastructure wise, there may not be a dif-

ference between private and public clouds. The main difference between them is the ownership and accessibility of cloud resources. Public clouds are owned by private cloud providers; however, access to the resources is open for public use. Public cloud examples include; Amazon's Elastic Compute Cloud (EC2), IBM's Blue Cloud, Google AppEngine, and Window's Azure, Office365, etc. On the other hand, a private cloud is owned and managed by a single organization and access to its resources is only open to the designated users. A hybrid cloud is a combination of two or more clouds, most commonly public and private clouds. In this model multiple clouds are bound together by standardized technologies but remain isolated entities. This model offers its users the benefits of multiple deployment models.

The service models for cloud computing environments include; Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS), and Infrastructure-as-a-Service (IaaS). The resources at the infrastructure level are provisioned from a virtualized environment as VMs. The commonly used virtualization technologies are Xen, VirtualBox, KVM, VMware, or Hyper-V. The provisioning of VMs in a virtualized environment is elastic [14]; thus, users can rapidly scale up and down their resource requirements on the fly. Moreover, users can have ubiquitous access to VMs from desktops or mobile devices, and can choose among the different pricing methods [30], [31], [82], [99].

2.2.3 Mobile Cloud Computing

Broadly, mobile cloud computing is the convergence of mobile computing, cloud computing, and networking [91]. Mobile cloud computing is an infrastructure in which both the data storage and the data processing happen outside of mobile devices. Mobile applications move the data processing and storage onto a cloud. Mobile users can access their applications, data, and cloud services through the Internet using a thin mobile client or a web browser, etc., [40]. More precisely, cloud computing is a networked and distributed computing model. On the other hand, mobile cloud computing is the amalgamation of mobile computing, cloud computing and wireless technologies. For examples, there are Google's Gmail, Facebook's location aware service, and Google Maps, etc. Another approach towards mobile cloud computing is by Satayanarayanan [95] who proposed that, rather than relying on a remote cloud, a nearby resource-rich cloudlet could be used. This architecture provides its users one hop latency and high data rate when accessing a nearby cloudlet, which incorporates further connectivity to remote cloud servers through the Internet.

2.2.4 Challenges of using Cloud Computing for Resource Augmentation

In a RAE when using resources from a cloud, the location of the resources is based on the deployment model of the cloud [73]. When using a local private cloud the

cloud resources are accessed through a local area network. However, resources from a public cloud are located at Internet/WAN latency and these are accessible through the Internet [28], [47], [15], [60]. There are some challenges when using resources from public clouds, which are as follows.

- **Latency:** The resources in a public cloud are accessed by mobile devices through the Internet. Therefore, the mobile devices experience high WAN latency due to distance, and low data rates compared with the resources accessed through WLAN from a local cloud. Moreover, cellular technologies used to connect to the Internet consume more energy than WLAN wireless technologies [75], [95], [69], [36]. Thus, task offloading to a public cloud may consume more battery energy than actually executing the task on the mobile device.
- **Bandwidth:** The Internet data rates and network latency are still lower than that of WLAN networks. Therefore, task offloading onto public cloud resources may not be beneficial for all kinds of tasks. For example, offloading a data intensive task incurs significant amount of additional data communication that involves energy consumption at the mobile device and increased completion time for the task. Therefore, when resources are located in a public cloud, data intensive tasks do not benefit from the remote execution.
- **Cost:** Using Internet resources comes with a monetary cost as compared with a WiFi network, which, in most cases, is free to its users. Also, the comput-

ing resources from a public cloud involve a monetary cost compared with the resources from a local private cloud.

2.3 Task Scheduling

Mobile devices decide to offload tasks when they are unable to run the tasks on their own. This inability could be due to the non-availability of required resources, or could be because the local execution can not yield desired task fidelity. When scheduling task offloading, the offloading decision is dictated by user-defined offloading goals that may include: (i) saving the battery energy on a mobile device, (ii) saving the monetary cost of using the computation resources, (iii) enhancing task execution time, and (iv) achieving any combination of the above. Thus, the performance of an offloaded task is judged based on the offloading goals set by the user. In the following sections, task offloading decision types, performance parameters, and methods are explained.

2.3.1 Task Offloading Decision

Static Offloading Decision

When static offloading policies are used then tasks are offloaded at the start of their execution. In this case, the requirement and availability of resources for the tasks is not estimated. There could be a situation in which a statically chosen service node

may not provide sufficient resources at the run time of the task. Therefore, the static task offloading decision may not always benefit the mobile devices, even though the offloaded tasks are executed on a resource rich service node.

A static offloading decision could be beneficial if the various parameters needed for the offloading decision could be accurately predicted in advance. Various prediction algorithms include probabilistic prediction [88], history based prediction [48], [51], and fuzzy control [45]. Using a static offloading decision, there is a possibility that achieving one offloading goal may affect the realization of the other offloading goals. For example, executing a task on a service node might decrease the execution time of the task; however, it might not conserve the mobile device's battery energy.

Dynamic Offloading Decision

The task offloading environment is a dynamic environment. For example, the requirement of resources for a task may change with a change in its input data, and/or a change in the user-defined goals (e.g. delay time, battery consumption, etc.). Also, the availability of resources may change at the service nodes (e.g. available CPU power, memory, file cache, etc.) and at the wireless network (e.g. bandwidth, network latency, etc.). In this situation, task offloading decisions based on prediction algorithms may not benefit the mobile devices and their tasks. Therefore, it is imperative to dynamically decide on a remote execution location based on the current requirements and availability of resources [109], [36], [111], [33], [68].

A task offloading decision using dynamic policies is based on the user-defined offloading goals and the current status of the availability and requirements for resources for the task. A task scheduler running on the mobile device gets the resource description of the service nodes and the network from a resource monitoring process running inside the task scheduler. A mobile client (running on the mobile device) maintains the current status of the available resources on the mobile device and the metrics of the task being offloaded (i.e. the task's input/output data size, code size, etc.). The client sends an offloading request to the task scheduler along with the current status of the resources and the offloading goals set by the user. The task scheduler decides the task execution location based on the offloading goals and the current status of resources obtained from the resource monitor and the mobile client. Based on the decision, the task is offloaded either onto a service node or it is executed locally at the mobile device. Therefore, for beneficial offloading, the resources at the mobile device and the service nodes must be monitored prior to offloading, such that a trade-off between the various task offloading goals can be made. For example, sometimes a short remote execution time for a task is more important though more battery energy is spent while offloading than while executing it on the mobile device, and vice-versa.

2.3.2 Task Offloading Performance

Generally, task offloading is beneficial if the cost of executing the task at a remote location is less than the cost of executing it at the mobile device. The cost of execution

for a task is a user-defined metric. It could be a task's execution time, usage of CPU power, and consumption of battery energy by the task, etc. The performance of an offloading task depends on the metrics of the task and the resources at the service node and the network. Equation (2.1) [103] gives a relationship between the local execution time of a task and its remote completion time when it is executed at a service node.

Consider that a task is offloaded onto a service node using a network. Let D be the amount of input/output data and C_l be the local execution of the task. The data rate available from the network is β . Then, the transfer time for data D is given as $\frac{D}{\beta}$. Let C_r be the remote execution time of the task when executed at the service node. Then, the remote completion time of the task includes: C_r , time to execute the task at the service node; and $\frac{D}{\beta}$, time to transfer the data to the service node. Let the user sets the offloading metric that the task will be offloaded if the remote completion time is less than the local execution time. This offloading condition can be expressed by the following inequality.

$$C_l > C_r + \frac{D}{\beta} \quad (2.1)$$

The computing speed of the service node is higher than the mobile device. If α is the fraction of a task's local execution time C_l that the remote execution takes at the service node, i.e. $\alpha = \frac{C_r}{C_l}$, then the computing speed-up factor (F) of the service node

compared with the mobile device is defined as $F = \frac{1}{\alpha}$. Therefore, Equation (2.1) can also be expressed as Equation (2.2) [103].

$$C_l > C_l \alpha + \frac{D}{\beta} \quad (2.2)$$

Equation (2.2) can be re-written as Equation (2.3) [103].

$$\beta > \frac{D}{C_l - \alpha C_l} \quad (2.3)$$

Let B be the critical network bandwidth for which the inequality in Equation (2.3) holds true. Then, the decision to determine whether offloading a task to a remote location is beneficial or not, can be given by Equation (2.4) [103].

$$\beta = \begin{cases} > B & \text{offloading is beneficial} \\ < B & \text{offloading is not beneficial} \end{cases} \quad (2.4)$$

This suggests that, at a given data size, the task offloading is beneficial, if the available bandwidth is greater than the minimum required bandwidth, and vice-versa.

In summary, Equation (2.1) shows that the remote completion time of an offloaded task depends on: (i) the computing power of the service node, (ii) the available network bandwidth, and (iii) the size of input/output data for the task. The performance of an offloaded task is the user-defined metric, which could be: (i) high responsive-

ness i.e. remote completion time smaller than the local execution time, or (ii) saving battery energy of the mobile device through remote execution of the task, or (iii) a combination of both. Based on Equation (2.1) the performance of an offloaded task depends upon the following parameters.

Amount of data

The amount of data for an offloaded task includes: (i) the size of the task's code and input data when initially the task is offloaded for remote execution, (ii) the amount of code offloaded when native and remote modules communicate during remote execution, and (iii) the amount of output data generated from the remote execution of the task. Equation (2.3) shows that the remote completion time directly depends on the amount of offloaded data. Thus, the amount of data can increase the task's remote completion time and the monetary cost (if any) of using network resources. Also, a mobile device's battery will deplete while sending/receiving data [33], [34].

A resource augmentation system [43] presents an example of offloading a small amount of data. In this system, the mobile client has root access to a service node; therefore, instead of offloading a task's code, it offloads a very lightweight shell script (a few bytes) to the service node. The script can download, install, and run arbitrary tasks on the service node from the Internet, and the results are submitted back to the mobile client. This system assumes that offloading is beneficial if a task's local and remote methods are loosely coupled, and, during remote execution the commu-

nication between them is small. On the other hand, if native and remote methods of an offloaded task are tightly coupled [74], then the communication between these methods will increase the amount of offloaded data. In this case, the performance of the offloaded task will degrade.

The above discussion suggests that the amount of data transferred affects the performance of the offloaded task. However, authors in work [75] emphasized on a high communication efficiency of the offloaded data, which depends more on the pattern of data transfer rather than the bandwidth of the network. The bulk amount of data transfer would provide more communication efficiency than the small amount of data. Also the work in paper [83] suggests that parallel data transfer could also be used to save the energy of mobile devices, if response time is not a metric for remote execution.

Network Parameters

Equation (2.1) shows the dependency of the remote completion time of an offloaded task on the transfer time of the data, which further depends on the available network bandwidth. Broadly, network parameters include: (i) available bandwidth, and (ii) network latency. The network latency between two nodes constitutes both the physical and the communication latency. Existing local area (WiFi) and cellular wireless technologies (LTE, 4G or 5G) provide different data rates and network latency. A WiFi network has a small network range, while 4G or 5G are ubiquitous. On the other

hand, WiFi induces less network latency than 4G or 5G [36]. Thus, energy consumption on a mobile device when transferring data using a WiFi network is less than in the case of 4G or 5G networks. In WAN networks the communication delays induced by long network latency and low bandwidth would lower the offloading performance [95]. Therefore, the selection of wireless technologies depends on the kind of offloading tasks and the offloading goals. For example, to get ubiquitous connectivity one has to use 4G or 5G networks. However, due to longer network latency, and smaller data rates than a WiFi network, the data transfer is slower and the task's response time will be higher. Similarly, if a task with critical response-time requirement is offloaded then network latency should be small. If a computation intensive task with a small amount of data is offloaded, then high network latency and response-time may not be critical for the task performance.

Speed-up Factor

The execution time of a task depends on the processor clock speed (t), the number of cores (N) and a factor X . The factor X depends on the memory, pipelining, and parallel execution of the task in the system. Thus, the speed-up factor (F) can be given by the following Equation (2.5) [69].

$$F = \frac{C_l}{C_r} = \frac{t_r \times N_l \times X_l}{t_l \times N_r \times X_r} \quad (2.5)$$

In the following discussion, the performance of an offloaded task is compared in a single service node and a cloud computing environment, on the basis of speed-up factor. In a single service node computing resources are limited. However, in a cloud computing environment computing resources are served as VMs and their availability is not limited. Further, virtualization technology can support parallel execution of a task on multiple VMs. Therefore, speed-up factor in cloud computing environment will be more than in a single node. In summary, Equations (2.4) and (2.5) show that the performance of an offloaded task depends on the metrics of the task (amount of data), network (available bandwidth), and service node (speed-up factor). A task offloading algorithm [109] suggests that randomly choosing only the network metrics is not sufficient for enhancing the performance of an offloading task, rather that the choice of a combination of parameters provides better performance. For example, the algorithm selects a remote server with low bandwidth but with very high speed-up factor.

2.3.3 Task Offloading Methods

Client-Server Communication

Mobile devices can use a client-server communication model between service nodes for task offloading. The communication protocol in this model can be; Remote Procedure Calls (RPC) [42], [20], [76], Remote Method Invocation (RMI), or sockets communi-

cation. These task offloading approaches can be exploited when the tasks of mobile devices are pre-installed on the service nodes. Therefore, these approaches restrict the users to offload only on specific service nodes. Another limitation is that when the mobile devices move to another location, the service nodes at the new location may or may not have the desired task pre-installed on them.

Virtual Machine Migration

In this approach, instead of offloading a task, a memory image of the VM that hosts the task is migrated to a service node. The VM migration workload is heavy and it incurs time and energy overhead on the mobile devices. MobiCloud [50] and Cloudlet [95] are task offloading systems that support VM migration. Cloudlet is a cluster of multi-core computers located at one-hop latency accessible through a WiFi network. An energy saving task offloading system called MAUI [36] uses a combination of VM migration and code offload using a WiFi or 3G network. Similarly, CloneCloud [33], [34], migrates a mobile device's clone to offload its task.

Mobile Code Migration

In this approach, the mobile code of a resource intensive part (code) of the task is migrated to a service node, and the results generated from the remote execution are sent back to the main task running on the mobile device. Scavenger [67], [68] uses a mobile code migration approach to distribute tasks to remote locations. The mobile

code migration approach has some advantages compared to VM migration approach. The size of a code being migrated is smaller than the image of a VM. Therefore, the overhead of mobile code migration is lower than VM migration. Also, the compile time and deployment time for a piece of code are faster than creating an image of a VM and starting its execution at a remote location.

2.4 Task Offloading Algorithms

In this section, various task offloading algorithms proposed in literature are discussed. The offloading algorithms are categorized based on: (i) a change in the availability of and requirements for resources, (ii) execution or response time, and (iii) network parameters.

2.4.1 Algorithms Based on Change in Resources

Spectra [41] [42], a task offloading algorithm, monitors variations in the available resources due to a change in the environmental conditions. For example, a change in the location of a mobile device will cause a change in the availability of resources to the device. The algorithm schedules task offloading based on the current usage of the resources by the task and the availability of the resources at the mobile device and the service nodes. This algorithm trades-off between battery energy of the mobile device and the performance of the offloading task.

A task's offloading goals may change due to a change in environmental conditions. In Spectra, the current requirement of resources for a task is specifically modified by its developer. However, Chroma [20], a tactics-based task offloading algorithm eliminates the run-time dependency of the developer of a task. It modifies a task by partitioning its possible modules that the developer knows in advance. Based on this task specific knowledge, Chroma improves its model of resource usage. Unlike Chroma, previously available task aware systems [53], [25], [13], can work, only if the current resources of a task do not change.

An offloading algorithm proposed in [45] adapts to the application patterns and resource fluctuations and makes a composite partitioning plan. To capture the application's pattern, classes of the application are annotated to see how many times the methods and data fields of the classes have been accessed. Thus, based on the application's execution graph, the partitioning engine decides which classes should be executed at a remote location or locally at the mobile device. This algorithm is an extension of the work in [74] - an Adaptive Infrastructure for Distributed Execution (AIDE). AIDE is a fine-grained run-time system that can decide when to offload and which offloading policy to use. An adaptive offloading algorithm in [44] is an extension of the algorithm proposed in [45]. This algorithm can dynamically partition an application at run time without any prior knowledge of the application. It supports memory intensive applications; therefore, offloading starts when the memory requirement reaches the mobile device's maximum memory capacity.

Mobile devices' hardware and technology is changing rapidly. An offloading algorithm [21] can rapidly modify an offloading task to adapt to a change in mobile devices' technology. The algorithm uses a little language approach and makes a static description of all the meaningful partitions of the task. The runtime system supports a static description by providing the dynamic components necessary to adapt to the fluctuating operating conditions. In this model, an unskilled developer can modify the task, and the performance of the task is comparable to an expert modified task.

2.4.2 Algorithms Based on Execution or Response Time

CloneCloud [33] is an application partitioner. It uses a static profiling algorithm to find the resource constraints and migration points in an application. However, it uses a dynamic profiling algorithm to build a cost model that optimally chooses the migration points to optimize the execution time.

An offloading algorithm [103] takes an offloading decision based on the predicted network bandwidth. The authors consider that computation offloading is a statistical decision and that the bandwidth data considered for decision making is modeled by a random process. This algorithm calculates the remote completion time of a task based on the statistical value of the network bandwidth between the mobile device and the remote location. An algorithm [108] uses time-out criteria to make the offloading decision for a task. This algorithm first sets a break-even time for the task execution, and if the task is not executed in the stipulated time, offloading is initiated. Authors

perceive that the execution time of a task could not be predicted since it can vary significantly for different instances of the execution. Moreover, it depends on the contents of the data rather than the size. Also instantaneous execution time-out is difficult to predict and the calculation for the prediction comes as energy overhead. Therefore, the authors consider a statistical approach to find optimal execution time-out for saving energy.

The algorithm introduced in [109] aims to reduce the response time of tasks and to conserve energy on mobile devices. To reduce network latency, the algorithm offloads tasks to a nearby service node. The algorithm performs better since it considers the speed-up factor of the service node along with other parameters. Therefore, this algorithm will find a service node with high speed-up factor when bandwidth of the network is low.

2.4.3 Algorithms Based on Network Parameters

Users desire short response time for cognitive applications. Therefore, when a cognitive application is executed at a remote location, long [95] latency of the network is not beneficial for the application. The authors proposed a remote execution system for cognitive applications, called “Cloudlet”, which has high wireless LAN bandwidth and one hop network latency.

An offloading system called MAUI [36] focuses on maximizing the energy savings under current networking conditions. Different from Spectra and Chroma, this system

accounts for the communication cost of each method of the task's code. The offloading decisions dictate how to partition the task at runtime with minimum burden on the programmer. MAUI does fine-grained and energy-aware offload of mobile code to a service node.

2.5 Related Work

In this research work, resource augmentation through task offloading is considered in a scenario having a large number of mobile devices and multiple service nodes. In this section, the challenges when resource monitoring is performed by a large number of mobile devices, are discussed and related work is presented.

As explained earlier, it is imperative to use dynamic offloading decisions such that task offloading is beneficial [20], [86], [34], [95], [42]. When an offloading decision is dynamic, the task scheduler needs an up-to-date information about the required and the available resources at the mobile device and the service nodes. Therefore, prior to offloading, a resource monitoring process running on the mobile device would repeatedly contact the service nodes to obtain the up-to-date status of currently available resources. Existing task offloading works based on dynamic offloading decision, consider offloading from a single mobile device to a server.

However, using a dynamic offloading decision has limitations when a large number of mobile devices and multiple service nodes are considered, and task scheduling is

performed within each mobile device of the system. In this scenario, the repeated communication between a large number of mobile devices and service nodes can cause communication overhead in the network. This situation will cause delays for the mobile devices that are waiting to get the up-to-date status of resource availability from multiple service nodes through the congested wireless network. During these delays, the battery of the mobile devices may drain due to continuous attempts to use the congested wireless network. Therefore, when the scheduler makes a task offloading decision on the basis of the resources of congested APs and service nodes, the offloading decision can be negative even though there are free resources in the system.

In the following sections, work related to the proposed solution for performing resource monitoring and task scheduling for a large number of mobile devices is discussed.

2.5.1 Resource Monitoring

The aforementioned challenges suggest that resource monitoring should be managed at a central node on behalf of mobile devices. Managing resource monitoring at a central node could help in avoiding repeated communication between a large number of mobile devices and multiple service nodes. This approach could lower the communication overhead in the wireless network and, consequently, it could lower the delays that the mobile devices experience due to congestion in the wireless network.

Some RAEs [36], [43] utilized a centralized stationary computer. In [36], task offloading related processes are performed at a central place to save memory capacity, time and energy of the offloading mobile device. It solves a call graph for partitioning an application at method level, and optimally solves the task offloading problem to minimize the total energy consumption across all methods of the application. In [43], a stationary computer, referred to as a registry server, is utilized to find appropriate service nodes. There are systems that utilize a broker node in a RAE to find service providers and negotiate for the required resources on behalf of mobile devices [29], [32]. Similarly, a resource broker entity in a grid computing environment [11] helps in finding the appropriate resources on the grid. These systems do not consider resource monitoring when a large number of mobile devices and multiple service nodes can be present in a system.

2.5.2 Task Scheduling

The previous work [78] on managing resource monitoring at a centralized node helped in avoiding the repeated communication of a large number of mobile devices with multiple service nodes. This solution lowered the communication overhead in the wireless network. Consequently, it lowered the time delays that the mobile devices might have experienced due to congestion in the wireless network while offloading their task. Furthermore, the centralized architecture could benefit mobile devices by optimally selecting resources for offloading with the aim to minimize the total

energy consumption and the total monetary cost across all mobile devices in the system. A hybrid cloud computing system uses a central scheduler that optimizes private and public cloud resource allocation with deadline and monetary constraints [98], [27]. The work in [107] provides a system for saving energy in smartphones in mobile cloud computing environments. The work in [102] minimizes the energy consumption in mobile devices by optimally scheduling the transmission rate with time delay constraints. The work in [46] proposes a task scheduling model to minimize the processing time cost of an application. Similarly, the work in [106] does qualitative analysis to decide whether to offload or not, with an aim to save energy.

Existing task scheduling models for resource optimization do not consider a resource augmentation scenario for a large number of mobile devices. Therefore, this is the motivation in this research work, to perform task scheduling for a large number of mobile devices at a central node. When mobile devices contact the central task scheduling service for task offloading, it decides the appropriate offloading location on behalf of mobile devices by optimally minimizing the total energy consumption or the total monetary cost across all mobile devices.

Chapter 3

Problem Statement

Resource augmentation through task offloading alleviates mobile devices from resource constraints by enhancing their CPU power and memory capacity, and saving battery energy [92], [102], [61], [36]. A mobile device can decide for task offloading when its available resources are not adequate either to execute the task or to achieve the desired performance of the task, such as short execution time or high responsiveness etc.

Resource augmentation through task offloading poses some challenges. When the offloading decision is static, then the current status of the available resources at the mobile device and at the service nodes is not checked. Even though a static offloading decision incurs low overhead on the mobile device [59], [70], [110], [86], merely offloading the task without knowing the current status of available resources may not benefit the mobile device.

The requirement of resources for a task may change with a change in the task's input data or the user-defined metrics (e.g. task completion time, battery consumption, etc.). The availability of resources may also change at the service nodes (e.g. available CPU power, memory, storage, etc.) and at the wireless network (e.g. data rate, network latency, etc.) [18], [68], [100]. Moreover, task offloading involves additional data communication which may increase the task's remote completion time and/or energy consumption when transferring the task related data [36], [58], [75]. Under these situations, the static offloading decision may not be appropriate and the mobile device may not achieve the desired performance for the offloaded task [42]. Therefore, it is imperative to offload a task using a dynamic offloading decision, based on the current requirements and availability of resources.

3.1 Research Question

In an RAE, service nodes are available for mobile devices to offload their tasks using a network in the area. Most existing systems for resource augmentation do not consider an area in which a large number of mobile devices could be present, i.e. they do not consider the scalability of their RAE.

In this research, the **motivation** is to consider an area which exhibits the presence of a large number of mobile devices seeking to offload their tasks. For example, a university campus or a corporate office (around 100 mobile devices) [62], [16], [23], or

a conference hall [17], or a festival (may be more than 200 mobile devices).

The presence of a large number of mobile devices poses challenges for task offloading, which are as follows.

- In the situation of a large number of users there could be a high density of users, which may cause congestion in the wireless network and service nodes. Consequently, the congestion in the wireless network may lead to the non-availability of minimum required bandwidth for mobile devices for beneficial offloading (as explained earlier in Chapter 2, Section 2.3.2). This situation may increase the response time of the offloaded tasks and decrease the battery lifetime of the offloading mobile devices. Also, due to the high density of users, the service nodes may not have enough resources for all mobile devices in the area [36], [42], [38].
- When the task offloading is dynamic, prior to offloading, the need for up-to-date information about the required and the available resources requires continuous running of a resource monitoring process in the mobile devices. Thus, in the situation of a large number of mobile devices, repeated resource monitoring by a large number of mobile devices can cause communication overhead in the wireless network. This situation will cause delays for the mobile devices that are waiting to get the up-to-date status of resource availability from multiple service nodes through the congested wireless network. During these delays, the

battery of the mobile devices may be drained due to continuous attempts to use the congested wireless network. In this case, the offloading decision, based on the resources of congested APs and service nodes, can be negative even though there are free resources in the system.

3.2 Research Objectives

The main objective of this research is to develop an RAE for a large number of mobile devices and multiple service nodes. More precisely, the aim of the research work is to:

- develop a way to get current information on users' density in an area for placing wireless APs and service nodes such that mobile devices could get computing resource from service nodes and the minimum required bandwidth from the wireless network for beneficial offloading;
- develop a centralized node architecture, (i) to investigate the communication overhead due to repeated resource monitoring by a large number of mobile devices, and (ii) to manage resource monitoring at the central node on behalf of all mobile devices;
- develop a task scheduler model for the central broker-node architecture to handle task scheduling on behalf of all mobile devices in the system, such that,

when optimally solving the centralized task scheduling problem, the total energy consumption and the total monetary cost across all mobile devices in the system could be minimized.

Chapter 4

Mapping Mobile Devices in the Local Area Network

In this chapter, the first step in the direction of supporting resource augmentation for a large number of mobile devices in an area is presented. The finite wireless range and fixed location of service networks could render them unable to provide services in certain situations in which a dense zone of users is created in the area. This could be in the event of a conference or a festival. Alternatively, in extreme cases such as a major power shutdown or network collapse in an area, the network may not be able to serve anyone at all. To deal with such situations, cyber foraging is proposed to establish a service network when no other service network is functioning in the area.

Cyber foraging [92], [18], has been used to enable mobile device users to get services from a unknown service network. Existing approaches for cyber foraging systems

use pre-installed and fixed service nodes at different places in the coverage area of the network. The placement of service nodes at fixed locations could have some limitations such as that the users are confined within the area of service. Consider two situations in a service network of an area. First, services are required outside the network area. This would be true, for example, in the situation of bad weather conditions, if the communication infrastructure is disrupted or there is a power blackout or a disaster in the area. Second, high density zones are created in the network due to a large number of the users. In the high density zones the pre-installed and fixed service nodes are incapable of providing services to users. Therefore, there arises a need: (i) for the mobility of service networks such that services could be provided in an area where there is no pre-existing network, and (ii) for a mechanism to rapidly establish a service network in the area.

When establishing a network, serving the users with limited resources is the aim of the work in this chapter. The approach used is to place service nodes and APs at strategic points in the area. The first step towards this approach is to gather current information on users' density distribution in the area. Mapping the intended service area with this information could help in identifying the strategic points for the placement of service nodes and APs. In this chapter, a scanning algorithm that can provide an approximate distribution of the users in an area is presented. More precisely, a scanning algorithm and simulation results are explained to show how the approach could be used.

The goal of gathering information on the density distribution is not to locate every single user in the area. However, the goal is to map the density of the users such that strategic points could be identified. Figure 4.1 shows the density distribution map of mobile devices obtained after scanning an area using the proposed scanning algorithm. A dot in the figure represent the presence of a mobile device at that point in the area. It is obvious from Figure 4.1 that high density areas need more resources than the other areas. Therefore, the information on density distribution is helpful in the area when setting up a service network without relying on any pre-installed infrastructure, and also helpful to serve the users with limited resources. When setting up a service network in an area, obtaining a density map of the area through the scanning process is independent of other network planning steps. The density map can be used as an input to other network planning techniques, which could decide where exactly to place service nodes and APs in the area based on the density map. However, the work in this chapter only deals with the scanning process; thus, other network planning steps for setting up a service network are beyond the scope of this work.

The main contributions in this chapter can be summarized as follows. The scanning method used to gather density information does not rely on any pre-existing communication infrastructure. It is a step in the direction of rapidly establishing a service network through cyber foraging. In this work, the scanning algorithm is presented and it is simulated to show how it could be used to gather the density

Figure 4.1: An example of an area map showing users' density. The numbers on the x axis and the y axis show the area dimension units.

distribution of the users in an area.

4.1 Related Work

Cyber foraging enables users to find unknown service nodes in their vicinity through a wireless local area network. Existing works on cyber foraging use pre-existing network infrastructure [67], [71], [95], [96]. However, the focus of the work in this chapter is on: (i) how to provide services to the users where there is no pre-installed network, and (ii) how to serve a large number of users with limited resources.

As mentioned earlier, to effectively serve a large number of users in an area, the density distribution of the users is required. Estimating the location of the users in the area could help in developing their density map. Their type of location information could be physical, symbolic, absolute or relative location [72]. Existing location

estimation systems are broadly classified as: (i) outdoor location estimation systems that normally use cellular networks or GPS systems, and (ii) indoor systems that use WLAN technologies [37], [39] to locate the users in an area.

The proposed scanning method does not use any pre-existing infrastructure in the area to create an approximate density map of the mobile devices. However, a cellular positioning system requires pre-installed cellular infrastructure and uses cellular communication technology [97]. Therefore, this system can locate only those mobile device that are supported by the cellular network. A GPS based system can not work for indoor location finding due to lack of line-of-sight between indoor wireless units and the GPS satellite. Similarly, ad-hoc sensor networks for location finding are required to be pre-installed and maintained in the subject area. Existing indoor WLAN based positioning systems use: (a) triangulation, (b) scene analysis, or (c) proximity methods [39]. Therefore, both indoor and outdoor positioning systems require pre-installed infrastructure for position estimation of mobile devices.

In summary, the work in this chapter considers how to establish a service network in an area when there is no pre-installed network. Therefore, the proposed scanning method does not rely on any pre-installed network or already prepared database of the area to estimate the location of the users. In this work, approximate location information of the mobile devices is required for developing a density distribution map of the users. The location information could be a building name, a room number, or any other descriptive information supplied by the users. Therefore, a positioning

system that provides pinpoint position information of the users in an area, is not the requirement in this work.

4.2 Scanning & Mapping the Area

In this section, how the scanning method scans a certain area and maps the density of the users is explained. The area is scanned by probing servers, equipped with a WiFi AP. In the wireless range of the probing servers, cyber foraging enables the mobile devices (referred to as nodes) to establish a connection with the probing servers and make a service contact with one of the probing servers. After making the service contact, the nodes use a pre-installed application to send their approximate location information to the probing server. In the simulations the pre-installed application is assumed for simplicity. However, as an alternative to the pre-installed application, web services could be used to send users' location information.

The focus is to map the density of nodes rather than locating the nodes. Therefore, the supplied (or discovered if possible) location information need not to be the exact location of the nodes. The location information could be a building name or a room number or some other descriptive information. In other words, knowing an approximate number for the nodes in a given area is important rather than their exact locations. After scanning the area, the probing servers consolidate the gathered information and map the area according to the density of the nodes.

In general, the messaging sequence involved in a cyber foraging network when a mobile device finds a server and makes a service contact with the server, is explained in Figure 4.2. It comprises of the following steps that are generic steps for a service discovery process in a service network [43], [113].

1. The access point advertises its Service Set Identifier (SSID).
2. A mobile device selects the SSID and requests a connection.
3. Connection is established with the access point.
4. The mobile device connects with the registrar through the WiFi access point.
The registrar maintains a list of available services in the service network.
5. The mobile device sends a request to the registrar to get a list of the available services.
6. The registrar sends the list of services with associated IP address and port number of servers on which these services are provided.
7. The mobile device selects a service from the list and accesses it on the particular server.
8. The service receives data from the mobile device and processes it.

Generally, in a cyber foraging network, the WiFi AP, the registrar and the servers are all placed at fixed positions. However, in the simulation setup, the probing servers

Figure 4.2: Process for a mobile device to access services from a server.

are not at fixed positions; rather these are mobile and a WiFi AP is attached to each of them. A probing server could be a mobile device such as a laptop, a smartphone, etc. The probing server moves in the scanning area according to the mobility model given in Figure 4.4, which is explained later in this section. The above mentioned steps for service discovery are used in cyber foraging when a node takes services from an unknown server. However, for the sake of simplicity in the simulation setup, the IP address and port number of the application running on the probing servers is explicitly provided to the nodes. Therefore, when a node communicates with a

probing server, steps 4, 5 and 6 are not implemented. Figure 4.2 illustrates the time a node takes to access a probing server and its service, and it is given by $t_c + t_l + t_s$.

The steps involved during the scanning process are explained as follows. A $K \times K$ area intended for scanning is divided into square blocks, as shown in Figure 4.3. The columns of the square blocks are scanned by probing servers denoted as $P_1, P_2, \dots, P_4$. The size of each square block is chosen such that it is completely covered under the wireless range of a probing server. If the wireless range of a probing server has a diameter of $2r_p$ then the side of a square block that can fit in it, is given by $1.41r_p$. For example, in block $B11$, the dotted circle is the wireless range of the probing server and the square block is completely covered under it.

Figure 4.3: Scanning the area.

Algorithm 1: Scanning by probing server P_j in j th column

```

1  $B_{jn} \leftarrow$  Consider  $j$ th column of the area with  $n$  blocks;
2  $T_{st} \leftarrow$  Stop time of the probing server in the block;
3  $cenB_{ji} \leftarrow$  Center of block  $B_{ji}$ ;
4  $cenB_{j1} \leftarrow$  Initial position of the probing server (center of first block in the
   column);
5  $cenB_{jn} \leftarrow$  Final position of the probing server (center of last block in the
   column);
6 for ( $i = 1 \rightarrow n$ ) do
7   The probing server stays at  $cenB_{ji}$  for stop time  $T_{st}$  and communicates
   with the nodes in its coverage area;
8   if ( $i < n$ ) then
9     The probing server moves to the center  $cenB_{j(i+1)}$  of the next block
      $B_{j(i+1)}$  in time  $T_{wt}$ , and communicates with the nodes in its coverage
     area;
10  end
11 end

```

Figure 4.4: Mobility model of a probing server during the scanning process.

The intended scanning area is divided into blocks such that all the blocks must be scanned by the probing servers. This approach will avoid scanning in a wrong direction and gathering false information on density distribution of the users. To simplify evaluation of the scanning process, scanning in only one of the columns is considered in Algorithm 1. Therefore, it is not considered in the algorithm whether the nodes found the closest server or not.

The performance of the scanning algorithm is evaluated by performing scanning in the consecutive three blocks of the last column of Figure 4.3. The last block in the last column is called B_{11} , the next B_{12} and so on. Likewise, the center of B_{11} is $cenB_{11}$, for B_{12} it is $cenB_{12}$ and so on. Algorithm 1 describes the scanning process using a mobility model for the probing server as given in Figure 4.4. The algorithm shows that probing server P_j starts scanning in the jth column. The total scan time taken by the probing server depends on its mobility model in the blocks. The mobility model given in Figure 4.4 includes: T_{st} , the stop time for the probing server at the center of a block; T_{wt} , the walk time that the probing server takes when it moves from the center of a block to the center of the next consecutive block. Following the mobility model, initially the probing server is at the center ($cenB_{j1}$) of the first block (B_{j1}) of the jth column and stays there for a time interval equal to stop time T_{st} . When the stop time in the first block expires, the probing server moves to the center ($cenB_{j2}$) of the next consecutive block (B_{j2}) in time T_{wt} , and again stays there for stop time T_{st} . This process continues until the probing server reaches the center of the last block ($cenB_{jn}$) and stays there for stop time T_{st} . When the scanning probing server moves, some nodes get out of the range of the probing server and get disconnected, while new nodes come into its range and get connected to it. Using this mobility model the probing server spends less time in the first and last blocks of any column (i.e. $T_{st} + 0.5T_{wt}$) than it spends in intermediate blocks (i.e. $T_{st} + T_{wt}$).

4.3 Simulation & Results

The simulations are carried out to demonstrate how the scanning process given, by Algorithm 1, works. The goal is to show how a density map would look after scanning and collecting the density information of the users in an area. The network simulation tool used is *ns-3* [7], which is a discrete-event network simulator. The motivation to use this tool is its detailed wireless 802.11x models as compared with *ns-2* [6].

In the simulation results, the effect of the node concentration in one block is observed in the percentage of detection in the other blocks. To observe this effect, three consecutive blocks are considered for scanning, and different node concentrations are set for the blocks. The node concentration is varied in one of the blocks (marked V), whereas, in the other two blocks, it is fixed at low (L) or at high (H). In three blocks, with three values for the node concentration in each block, there are 12 combinations of the node concentrations in three blocks, shown in the tables in Figure 4.5. For example, notation VLL (Figure 4.5(a)) shows that the first block ($B11$) has varying concentration of nodes (V) while the second ($B12$) and the third block ($B13$) have low concentration of nodes (L). In the simulation results, out of the 12 combinations, results for only 9 combinations are shown. The results for the other three combinations show that there was no effect of the node concentration on the percentage of detection (explained later in this section).

Figure 4.6 shows the percentage of detected nodes in three consecutive blocks

B11	V	V	V	V
B12	L	L	H	H
B13	L	H	L	H

(a) Varying nodes in the first block.

B11	L	L	H	H
B12	V	V	V	V
B13	L	H	L	H

(b) Varying nodes in the second block.

B11	L	L	H	H
B12	L	H	L	H
B13	V	V	V	V

(c) Varying nodes in the third block.

Figure 4.5: Different combinations of node concentration in the three blocks.

$B11$, $B12$ and $B13$ at different node concentrations in each block. The WiFi model used in the simulations is 802.11a. Using default parameters of this model in *ns-3*, while scanning, it is observed that the diameter of the wireless range of the probing server is 240 *m*. Thus, the side of a square block that can fit in this range will be approximately 170 *m* ($1.41r_p$). Therefore, the area of the three consecutive blocks considered in the simulation process is $170 \times 510 \text{ m}^2$. In the scanning process the

standard *ns-3* mobility models are not used for the mobility of the probing server, rather the mobility model illustrated in Figure 4.4 is simulated in *ns-3*. The stop time (T_{st}) for the probing server at the centre of each block is 20s, and the probing server starts to move at a speed of 7m/s on the line from the centre of the first block to the centre of the next consecutive block until it reaches the centre of the last block. Using the default simulation parameters, initial simulations showed that the maximum number of nodes that the probing server could detect in one block is around 130 – 140 nodes. The nodes in excess of this number could not be detected by the probing server. The drop in the percentage of detected nodes is due to the congestion, which is reasonable to assume in a situation in which a large number of nodes try to connect with the probing server at the same time to get some services. Based on this observation, the node concentration in a low concentration block (L) is set to 30, and in a high concentration block (H) it is set to 140. In a varied block (V), the number of nodes is varied from 60 to 180. In the simulations, it is assumed that (i) the movement of the nodes is negligible, (ii) nodes in each block are scattered randomly and uniformly over the whole area of the block, and (iii) the application on the probing server with which nodes make a service contact, is always on.

A newly introduced WiFi model 802.11ac provides a higher data rate and greater wireless range compared to the previous wireless technologies. Therefore, utilizing this in the scanning setup, higher scalability can be achieved compared to when utilizing 802.11a. With the availability of higher data rates, more nodes can be detected in

a given area. Moreover, due to a larger wireless range, a square block of larger area can be covered, and consequently, a given area can be scanned in less time. Thus, utilizing this WiFi model, a larger number of nodes can be detected in a larger area and in a shorter time compared to when using 802.11a.

Effect of Node Concentration on the Scalability of Detection

It is observed from the simulation results that, when the node concentration in two blocks is low, then the varied node concentration (low or high) in the third block does not affect the percentage detection in any of the blocks, e.g. in the case of VLL , LVL and LLV . Therefore, out of the 12 combinations of node concentrations mentioned in Figure 4.5, the graphs for these three combinations (i.e. VLL , LVL and LLV) are not shown in Figure 4.6.

It is also observed that, when varying node concentration (V) and high node concentration (H) are in consecutive blocks in either order, then the percentage of detection is low in the second block (whether it is V or H). However, there are exceptions to this observation as in the case of LVH (Figure 4.6(d)) and LHV (Figure 4.6(g)). In these cases, the percentage detection does not drop even though V and H are in consecutive blocks. The reason for the small drop is that there is low concentration (L) in the first block. Thus, the simulated probing server did not suffer from severe congestion when it scanned in the first block. In this case, when the probing server moves to the second block, it is free to entertain the high node

Figure 4.6: Percentage of detected nodes at different node concentrations.

concentration (V or H) of this block. Similar behaviour is observed between the second and third block in these two special cases, i.e., when there is little congestion in the second block, detection is high in the third block.

However, in other results such as in VHL (Fig. 4.6(b)), VHH (Fig. 4.6(c)), HVL (Fig. 4.6(e)), HVH (Fig. 4.6(f)) and HHV (Fig. 4.6(i)), the percentage of detected nodes is very low. In these cases, congestion at the probing server could be due to high concentration of nodes in the first block and the probing server is not cleared even when it reaches the second or third block.

4.4 Conclusion

In this chapter, the aim was to serve a large number of mobile devices with limited resources. The approach utilized was to place service nodes and APs at strategic points in the area. Therefore, a scanning algorithm was proposed to determine users' distribution map in the area such that strategic points could be identified. The proposed scanning approach could be beneficial in a situation in which no communication network is present in an area. In this situation, the focus was to determine users' density distribution in the area but not to locate every single user. Therefore, this work is not intended for critical applications such as victims' location identification in a rescue operation.

The simulation model and its parameters are for a simplified case. The model should be enhanced with more realistic parameters and obstacles. While the simu-

lation results show how it works and how it could be useful, its performance in real environments could not be predicted yet and this should be studied further. It is still to be seen how this algorithm will perform in a real life situation when other factors could affect the performance. The factors that could affect the performance may include: congestion, disconnection, and authenticity, etc. These factors will affect the scanning process, which will result in gathering incorrect information on the density distribution of the users. Further, the incorrect density distribution will lead to improper allocation of service nodes and APs in the area.

Chapter 5

Broker Assisted Centralized Management

As explained in Chapter 4, there could be places in which a large number of mobile devices seek task offloading. In this chapter, a framework for a large RAE is presented. The RAE considers the presence of a large number of mobile devices and multiple service nodes. The scalability of the resource monitoring process in a large RAE is the main aim of the work in this chapter. In this environment, a centralized node is proposed to manage resource monitoring on behalf of all mobile devices in the system. The simulation results show that managing resource monitoring at the centralized node lowers the communication overhead in the wireless network, and also lowers the resource monitoring time experienced by the mobile devices.

Most existing works on resource augmentation [20], [38], [36], [43], [71], base their

experimental evaluation or performance analysis on a single or a few mobile devices and service nodes. These systems do not consider an area in which a large number of mobile users could be present and have a need to use the system, i.e. they do not study the scalability of their systems. The scalability of a RAE is a big challenge. Satyanarayanan [92] mentioned that the issues with the localized scalability in a RAE could result from: (i) multiple interactions from the mobile devices to the service nodes of the system, or (ii) the presence of multiple mobile devices in the system. The second issue provides the notion of a RAE in which a large number of mobile devices could be present. In the situation of a large number of mobile users, there could be a high density of users over the entire area, or over small part(s) of the area. This situation could cause congestion at the service nodes, and/or on the wireless network. In the congested area, the wireless access points may not provide the minimum required bandwidth, which is required for beneficial task offloading. Also, the service nodes may not have enough resources for all the users in the area [42], [38], [36].

Consider a large RAE in which mobile devices offload tasks using dynamic offloading policies. Therefore, in this case, continuous running of the resource monitoring process in each of the mobile devices is required. Consequently, all service nodes in the system are repeatedly contacted by a large number of mobile devices. The repeated resource monitoring by a large number of mobile devices poses challenges. This situation may incur significant communication overhead in the wireless network.

In turn, it may cause delays for the mobile devices that are waiting to get updated resource monitoring information from multiple service nodes through the congested wireless network. During these delays, the batteries of the mobile devices may drain due to the continuous attempt to use the congested wireless network.

The main contributions in this chapter can be summarized as follows. The overhead of the resource monitoring process when it is performed by a large number of mobile devices is presented. In this work, a centralized node-based framework is proposed that manages resource monitoring on behalf of all mobile devices in the system. This approach helps in lowering the communication overhead that could have manifested due to resource monitoring by a large number of mobile devices. Lesser communication overhead, in turn, lowers the resource monitoring time for the mobile devices.

5.1 Proposed Framework for Centralized Management of Resource Monitoring

In this chapter, managing resource monitoring on behalf of all the mobile devices at a centralized node instead of performing it at each of the mobile devices, is proposed as a solution to lower the communication overhead due to resource monitoring in a large RAE. On behalf of all the mobile devices, the centralized node gathers resource descriptions from all the service nodes in the system. The centralized node consolidates

all the resource descriptions and uses this information when scheduling task offloading on behalf of all mobile devices in the system. Thus, the centralized node can find the best available resources for all the mobile devices based on their requirements and offloading goals. In this way, the mobile devices need not repeatedly contact all the service nodes for resource monitoring. Therefore, the centralized node is referred to as broker-node. Using the broker-node the communication overhead in the wireless network may be lowered that might have occurred due to repeated resource monitoring by a large number of mobile devices. The time delay in the mobile devices may also be lowered when waiting to get updated resource monitoring information from multiple service nodes.

To evaluate the proposed framework, the average resource monitoring time taken by the mobile devices, and the average number of collisions occurring in the wireless channel in the broker-based scenario (Figure 5.2) are compared with a baseline scenario (Figure 5.1). The two scenarios differ at: (i) infrastructural level, (ii) the location of the *TaskScheduler()* service, and (iii) the mechanism for the resource monitoring process. The points of difference are as follows.

- At infrastructural level, the centralized broker scenario includes a centralized broker-node. However, the baseline scenario does not include a broker-node.
- The *TaskScheduler()* service that manages task offloading is utilized at the mobile devices in the case of the baseline scenario. However, in the broker

Figure 5.1: Baseline scenario.

Figure 5.2: Centralized broker scenario.

scenario this service is utilized at the broker-node. The mobile devices in the broker scenario employ an *RB_Client()* service that communicates with the *TaskScheduler()* service at the broker-node.

- In the baseline scenario, all the mobile devices perform resource monitoring on their own using the *ResourceMonitor()* service. However, in the broker scenario, the centralized broker-node manages resource monitoring on behalf of all the mobile devices using the *ResourceMonitor()* service. The *RB_Client()* service in the mobile devices can get resource descriptions of all the service nodes just

by contacting only the *ServerApp()* service in the broker-node.

Figure 5.3: TCP handshaking and resource description file request/response protocol.

In both scenarios, the service nodes use a *ServerApp()* service that accepts resource description requests from the *ResourceMonitor()* service. In the broker scenario, the broker-node also uses the *ServerApp()* service to accept resource description requests from the *RB_client()* service in the mobile devices. The *ServerApp()* service is modeled as an M/M/c/K queue, and the queueing discipline is First-Come-First-Serve (FCFS). The length of the queue is finite (K), and the incoming requests to the queue are serviced by multiple servers (c) (Figure 5.4). The arrival rate of the incoming requests follows a Poisson distribution, and therefore, the inter-arrival time

between the requests follows an exponential distribution.

Figure 5.4: Service model of *ServerApp()* service.

The request/response protocol between the *ResourceMonitor()* and the *ServerApp()* service, or the *RB_Client()* and the *ServerApp()* service is the same (Figure 5.3). A client service requesting a resource description from a service node first establishes a TCP connection with the *ServerApp()* service of the service node, through a three-way handshake. After the TCP connection is established, the client sends a resource description request to the *ServerApp()* service, and the service in response to the request sends an XML descriptor file. The client, after downloading the requested file, terminates the TCP connection. The transfer time of the resource description file of size approximately 200 bytes, will have very little impact on the resource monitoring time. The time taken by a mobile device to get a resource description from one service node is the time interval between the instant t_1 when the device sends a request for TCP connection and the instant t_2 when the mobile device receives the

resource description file (Figure 5.3).

In the baseline scenario, when a mobile device wants to offload a task to a service node, the *TaskScheduler()* service in the device finds an appropriate service node for the mobile device. To find an appropriate service node the *TaskScheduler()* service first gathers a resource description from all the available service nodes, the network, the mobile device and the task. Then it estimates the cost of offloading the task based on the gathered information and the offloading goals set by the mobile device user. On the other hand, in the centralized broker scenario, when a mobile device wants to offload its task, it first contacts the broker-node. The broker has the up-to-date information of the available resources at the service nodes. The broker takes the current information of the availability and requirement of resources at the mobile device, and the offloading goals set by the user. Based on the information, the task scheduler in the broker-node decides on an appropriate remote execution location on behalf of the mobile device.

In this chapter, the objective is to investigate the overhead of the resource monitoring process in a large RAE. Therefore, at this stage, task offloading is not considered, rather only resource monitoring is performed. The resource monitoring process in both scenarios is as follows.

- In the baseline scenario, the *TaskScheduler()* service in a mobile device invokes the *ResourceMonitor()* service that sends resource description requests to the *ServerApp()* service in the service nodes. In response to these requests the

Figure 5.5: Two Linux containers representing a service node and a mobile device connected through *ns-3* WiFi network.

ServerApp() service in each service node sends a resource description of their service nodes to the *ResourceMonitor()* service.

- In the broker scenario, the mobile devices do not directly contact the service nodes for resource monitoring. However, the *RB_Client()* service in a mobile device first sends a resource description request to the *ServerApp()* service in the broker-node. The *ServerApp()* further invokes the *TaskScheduler()* service to get a resource description directly from the resource description files stored by the *ResourceMonitor()* service. In this scenario, the *ResourceMonitor()* service

in the broker-node is scheduled to be invoked periodically to get a resource description from all the service nodes. Thus, the mobile devices can get the resource description of all the service nodes by only contacting the broker-node.

5.2 The Hybrid Simulation & Emulation Experimental Setup

The simulations are carried out to compare the performance of the centralized broker scenario (Figure 5.2) with the baseline scenario (Figure 5.1). The performance evaluation is based on the resource monitoring time experienced by the mobile devices and the scalability of the resource monitoring process, in both scenarios. The experimental setup for a large RAE is a hybrid of emulation and simulation techniques of the Linux Operating System (OS) and the *ns-3* network simulation tool. The service nodes, the broker-node and the mobile devices are emulated using Linux VMs, also called Linux Containers (LXC) [3]. It is not practically feasible to use 200 real mobile devices, therefore, mobile devices are emulated using Linux containers. Using this technique multiple light weight VMs can be created and run on the same host. An LXC combines the resource management and resource isolation (*cgroup* & *namespaces*) of the Linux kernel. The containers have their own private view of the OS, the file system, and the network interfaces. Thus, containers can be constrained to use a defined amount of resources.

The network simulator tool *ns-3* [7] is used since it has detailed wireless 802.11x models as compared with *ns-2* [6]. Moreover, in *ns-3*, TAP *NetDevice* integrates a simulated CSMA or WiFi network, and Linux containers. The TAP mechanism uses *TapBridge NetDevice* to make connections from *ns-3* to Linux containers. As shown in Figure 5.5, the *TapBridge* arrangement connects the I/O of *ns-3 NetDevice* (WiFi *NetDevice* in *ns-3* node) to the I/O of TAP *NetDevice* of the Linux container. This arrangement is made, in order to appear, as if containers are directly connected to a simulated *ns-3* network.

The setup in Figure 5.5 shows a service node and a mobile device emulated using Linux containers. Both are connected through a WiFi network simulated in *ns-3*. The hybrid setup has a large number of mobile devices and multiple service nodes. The setup is implemented in a single server machine with an Intel Xeon(R) E5420 2.50GHz, quad core CPU and 8GB of RAM. The Linux OS distribution is Ubuntu 12.04.2 LTS precise. The CPU strength of the service nodes is set higher than the mobile devices using the *cgroup* utility in Linux OS. The WiFi network in both scenarios (Figures 5.1 & 5.2) is simulated using 802.11a model with its default parameters in *ns-3*. The *TaskScheduler()*, *ResourceMonitor()*, *ServerApp()* and *RB_Client* services are implemented in Java. The resource description requests from a large number of mobile devices were sent to multiple service nodes using Java multi-threading.

5.3 Evaluation

5.3.1 Simulation Experiments

In the simulations, resource monitoring time is measured for the mobile devices in a large RAE using the baseline scenario and the broker scenario. As defined in Section 5.1, the time interval $(t_2 - t_1)$ (Figure 5.3) is the resource monitoring time taken by a mobile device to get a resource description from one service node. In the simulation results, each point is the average of the total resource monitoring time of all the mobile devices. The total resource monitoring time for a mobile device is the time interval between the instant the mobile device starts sending resource description requests to service nodes and the instant it has received resource description responses from all the service nodes. In the simulations, task offloading is not considered. Therefore, the resource monitoring time in an actual situation could be less than in the simulations, because some mobile devices may not be monitoring since they have already offloaded their computing requests.

In the simulations, the objective is to observe how resource monitoring is affected by: (i) the number of mobile devices that are currently doing resource monitoring, (ii) the number of service nodes that are being monitored by mobile devices, (iii) the size of the wait queue, and (iv) the number of servers used to serve the incoming requests in the queue of the *ServerApp()* service. In the simulations, an appropriate value for the queue size and the number of servers in the *ServerApp()* service is determined

at the beginning. Then, the number of mobile devices and the number of service nodes are varied to see their effect on: (i) the total resource monitoring time of the mobile devices, and (ii) the total number of mobile devices that could do resource monitoring, i.e. the scalability of the system.

The resource monitoring time is measured for different number of mobile devices when there are 3, 5 or 7 Service Nodes (SNs) in the system. In the legends the number of service nodes when there are 3, 5 and 7 service nodes, is denoted as SN3, SN5 and SN7 respectively. The baseline scenario is denoted as - 'Baseline', and the centralized broker scenario as - 'Broker'. The queue discipline of the wait queue is FCFS. The Poisson interval for the incoming requests to the queue is set to 15s. It is set to a small value to see the effect of a large number of requests generated over a small period of time. Therefore, all the mobile devices send resource description requests to all the service nodes or the broker-node over an interval of 15s. The size of the queue is set to a value such that the incoming requests over the Poisson interval should not find the wait queue full, and none of them is dropped-off. In the results, the total resource monitoring time of a mobile device is obtained by calculating the average over 30 iterations, and each point in the results is the average over total resource monitoring time of all the mobile devices.

5.3.2 Analysis of Results

Effect of Queue Size & Number of Servers in the *ServerApp()* service

It is observed while measuring resource monitoring time that even when small sizes for a wait queue (i.e. $K \in \{20, 50, 100\}$) are considered, the queue is never full. The number of incoming requests increases with an increase in number of mobile devices; however, the incoming requests to the queue do not drop-off. Therefore, the size of the wait queue in the *serverApp()* service is set to an arbitrary value of 500 (i.e. $K = 500$). The results in Figure 5.6 show the effect on the resource monitoring time of varying the number of servers in the *serverApp()* service (described in Section 5.1 and Figure 5.4). This effect is observed in the baseline scenario when there are 5 (Figure 5.6(a)) or 7 (Figure 5.6(b)) service nodes in the system. In each case the resource monitoring time is observed by varying the number of mobile devices when the number of servers (c) in the *serverApp()* service is 20, 50, 100 or 300. It is observed that, for a given number of service nodes and mobile devices, the resource monitoring time is almost the same for a different number of servers (c) in the *serverApp()* service. Therefore, during further observations in both scenarios, the size of a wait queue is considered as 500 (i.e. $K = 500$), and the number of servers in a *serverApp()* service as 20 (i.e. $c = 20$).

Figure 5.6: Effect of number of servers in the *serverApp()* service on resource monitoring time.

Resource Monitoring Time & Networking Overhead

Further, the average resource monitoring time of all the mobile devices and the scalability of the system is compared in the baseline and the broker scenarios. Figure 5.7(a) shows the average resource monitoring time when 3, 5 or 7 service nodes are present in the baseline scenario. However, in the broker scenario, the average resource monitoring time is shown when 5 service nodes are present. In this case, a very small difference in the average resource monitoring times is noticed when there are 3, 5 or 7 service nodes in this scenario.

The results in Figure 5.7(a) show that in the baseline scenario, when the number of service nodes or the number of mobile devices increases, the average resource monitoring time increases. However, the broker scenario this time is not affected

Figure 5.7: (a) Comparison of resource monitoring time and scalability, (b) collisions in the WiFi channel, in baseline and broker scenarios.

much by the change in the number of service nodes, as mentioned above. Even when the number of mobile devices increases, the increase in the resource monitoring time is negligible as compared with the baseline scenario. It is also observed that, in the broker scenario, when the number of service nodes increases, the scalability is not affected; on the other hand, the scalability decreases in the baseline scenario. Moreover, the scalability in the broker scenario is higher than the baseline scenario. In the broker scenario, Figure 5.7(a), resource monitoring for upto 220 mobile devices is shown when there are 5 service nodes in the system. However, the scalability may go higher than this number, i.e. a higher number of mobile devices could do resource monitoring. In the baseline scenario, when the number of service nodes is 3, 5 or 7,

the number of mobile devices that could do resource monitoring is 180, 140 or 100 respectively.

The results show that the average resource monitoring time experienced by the mobile devices in the broker scenario is much less than the baseline scenario. In summary, average resource monitoring time increases and scalability decreases in the baseline scenario, when the number of services nodes or the number of mobile devices increases. In this scenario, the degradation in the performance is due to the increase in the amount of communication traffic between the service nodes and the mobile devices. However, this is not the case when using the proposed centralized broker-node scenario. In this case, all the mobile devices are only communicating with the broker-node and only the broker-node is communicating with the service nodes. Thus, the amount of communication traffic does not increase much when the number of mobile devices or service nodes increases. Consequently, the resource monitoring time experienced by the mobile devices is lower and scalability is higher than the baseline scenario.

Effect of Wireless Network

Further, the results in Figure 5.7(b) reveal that, in a large RAE, a large number of mobile devices and multiple service nodes cause congestion in the WiFi network. Consequently, the congestion in the WiFi network causes collisions that account for: (i) the increase in the resource monitoring time, and (ii) the decrease in the scalability

of the system. In a WiFi channel, whenever there is a collision, the frame in the MAC layer that was ready to be transmitted to the channel is backed-off for a random time. Therefore, due to an increase in collisions, the frames in the MAC layer are queued for a longer time. Figure 5.7(b) shows the average number of collisions in the WiFi channel during the resource monitoring in both scenarios. The results show that in the baseline scenario, the average number of collisions increases when either the number of service nodes or the number of mobile devices increases. However, in the broker scenario, the average number of collisions is small compared with the baseline scenario, and it does not increase much when the number of mobile devices increases. The smaller number of collisions in the broker scenario is due to the lower amount of communication traffic between the broker-node and the mobile devices as compared with the large amount of communication traffic in the baseline scenario.

5.4 Conclusion

The aim of the centralized broker-node architecture was to manage resource monitoring on behalf of all the mobile devices in the system such that: (i) the resource monitoring time and communication overhead is low, and (ii) the scalability of the system is high. The simulation results showed that when resource monitoring was performed by a large number of mobile devices, the large amount of communication traffic between the mobile devices and the multiple service nodes caused congestion

in the WiFi network. The congestion in the network increased the average resource monitoring time experienced by the mobile devices and, consequently, decreased the scalability of the system. However, when resource monitoring was managed at the centralized broker-node on behalf of all the mobile devices, only a small amount of communication was generated between the mobile devices and the broker-node. Consequently, only a small number of collisions occurred in the WiFi network. Therefore, using the proposed centralized broker architecture, the average resource monitoring time is smaller and the scalability of the system is better than the baseline scenario. The results are based on simulations, in actual implementations there could be other impairments or factors affecting the performance. Therefore, further tests in actual networks are needed before deployment to see if the same performance obtained in the simulations will be obtained in actual networks.

Chapter 6

Energy Optimization: The Local Resources Case

In Chapter 5, a centralized broker-node framework for a large RAE was presented. The centralized broker-node was utilized to manage resource monitoring on behalf of all mobile devices in the system. The centralized broker-node approach helped in: (i) reducing the communication traffic created by a large number of mobile devices during resource monitoring, and (ii) reducing the time delay experienced by mobile devices when getting resource descriptions from multiple service nodes.

In this chapter, the centralized broker-node approach to manage resource monitoring presented in Chapter 5 is extended. The broker-node is proposed for handling task scheduling on behalf of all mobile devices in the system. Further, for the centralized task scheduling problem, a mathematical model subject to various constraints is

proposed such that the total energy consumption across all the mobile devices could be minimized. The task scheduler model optimally solves the task scheduling problem (task assignment) and provides a significant reduction in the total energy consumption compared with the total energy consumption when tasks are offloaded from the centralized scheduler without optimization.

Task scheduling for resource augmentation of a mobile device is different from the standard task scheduling in operating systems. Unlike in operating systems, in this case, a task scheduler at the application layer schedules tasks onto the available service nodes. The task scheduling occurs when it finds that the available resources at the mobile device are not adequate either to execute the task or to achieve the desired performance of the task. The task offloading is more of a task assignment rather than task scheduling. The scheduling in Real-Time Operating Systems (RTOS) happens in a different way, which is based on various scheduling algorithms such as co-operative, pre-emptive, and round-robin scheduling, etc. On the other hand, when the offloaded task reaches the destination service node, scheduling the execution of the task is upto the service node.

It is observed that the proposal of a broker does not necessarily mean relying only on one node, where it becomes a single point of failure. Multiple brokers, where load balancing and reliability are considered, should be part of the system design. However, these issues of load balancing and reliability are outside the scope of this research work.

The solving time for the optimization problem in the current proposal takes a relatively long time if it is intended to be used in a real-time system. Therefore, the current form is not suitable for real-time implementations. The proposed model for the centralized task scheduling problem is formulated in the following section.

6.1 Task Scheduler Model

The task scheduler model is constructed for a large RAE which uses cyber foraging for resource augmentation. Therefore, the service nodes in the system are referred to as surrogate nodes. These nodes are accessed by mobile devices in the surrounding area through a WiFi network. The goal of the proposed task scheduling model is to find an optimal solution for task assignment such that the total energy consumption across all mobile devices in the system could be minimized. Figure 6.1 illustrates task offloading from a mobile device onto a surrogate node. The figure shows the various cost and constraints parameters associated with the mobile devices and the surrogate nodes of the system. The model uses these parameters to find an optimal solution for the task scheduling problem. To this end, a mathematical model based on the following assumptions and notation is proposed.

Figure 6.1: Graphical representation of the task scheduler model.

6.1.1 Assumptions

- To simplify initial analysis, it is assumed that each mobile device has a single task to offload/execute at any given time.
- All mobile nodes can communicate with the broker-node.
- In an actual task offloading process that uses dynamic offloading policies, the resource monitoring process periodically contacts the surrogate nodes to get their up-to-date resource description. In this model, only task offloading is considered. Therefore, it is assumed that the broker knows the real-time status of the available resources at all the surrogate nodes.

6.1.2 Notation

The following notation is composed of sets, cost parameters, constraints parameters, and decision variables of the model.

Sets

- M , the set of all mobile devices, where mobile device $m \in M$.
- S , the set of all available surrogate nodes, where surrogate node $s \in S$ has the following properties.
 - p_s , the CPU power (expressed in percentage) currently available at the surrogate node $s \in S$.
 - k_s , the memory capacity (expressed in percentage) currently available at the surrogate node $s \in S$.

Cost Parameters

- e_m , energy consumption in mobile device m when its task is executed locally on the mobile device.
- e_{ms} , energy consumption in mobile device m when its task is executed remotely on surrogate node s .

Constraints Parameters

- t_m , execution time of task from mobile device m when the task is executed locally on the mobile device.
- t_{ms} , execution time of task from mobile device m when the task is executed remotely on surrogate node s .
- l_{ms} , remote completion time of task from mobile device m when the task is executed remotely on surrogate node s . The task completion time is the sum of the time spent in sending the task, executing it on the remote surrogate node, and receiving the results back.
- p_{ms} , percentage of processing power p_s of surrogate node s currently used to process task from mobile device m .
- k_{ms} , percentage of memory k_s of surrogate node s currently used to process task from mobile device m .

Decision Variables

- x_{ms} , is a binary variable such that $x_{ms} = 1$ if and only if the task from mobile device $m \in M$ is offloaded to surrogate node $s \in S$; otherwise, $x_{ms} = 0$ if the task is not executed on the surrogate node s . The task runs on mobile device m , if and only if

$$\sum_{s \in S} x_{ms} = 0 \quad (6.1)$$

The values of x_{ms} are subject to multiple assignment constraints (6.7), which are explained later in Subsection 6.1.4. X is a set of all the decision variables, where a decision variable $x_{ms} \in X$, $m \in M$ and $s \in S$.

6.1.3 Cost Function

The cost function $C(X)$ (6.2) represents the total energy consumption across all mobile devices whether executing their tasks locally on the mobile devices or remotely on surrogate nodes.

- The first term in the cost function represents the total energy consumption by the tasks that are offloaded to surrogate nodes. In this case, the task assignment decision variable $x_{ms} = 1$. It represents the energy consumed e_{ms} when a task from mobile device $m \in M$ is offloaded onto surrogate node $s \in S$.
- The second term represents the total energy consumption by the tasks that are executed locally on the mobile devices. In this case, the task assignment decision variable $x_{ms} = 0$. It represents the energy consumption e_m when a task from mobile device $m \in M$ is executed locally on the mobile device.

$$C(X) = \sum_{\substack{m \in M \\ s \in S}} e_{ms} x_{ms} + \sum_{\substack{m \in M \\ s \in S}} e_m (1 - x_{ms}) \quad (6.2)$$

6.1.4 The Model

The objective function of the Task Scheduling Problem (TSP) is to minimize the total energy consumption, which is equivalent to maximizing the total energy saving across all mobile devices. The objective function of the model is as follows.

$$\min \quad (C(X)) \quad (6.3)$$

The model is subject to various constraints (6.4), (6.5), (6.6), and (6.7).

Overloading Constraints

The number of surrogate nodes ($|S|$) is very small compared with the number of mobile devices ($|M|$). Each surrogate node $s \in S$ has a limited amount of CPU power (p_s) and memory capacity (k_s). The overloading constraints prevent each of the surrogate nodes from resource overloading. For example, when tasks are assigned to surrogate node s then the overloading constraints ensure that the total amount of the CPU power (6.4) and the total memory capacity (6.5) that is required by all the tasks assigned to the surrogate node, should be less than the surrogate node's CPU

power (p_s) and memory capacity (k_s) respectively. The overloading constraints are given as follows.

$$\sum_{m \in M} x_{ms} p_{ms} < p_s \quad \forall s \in S \quad (6.4)$$

$$\sum_{m \in M} x_{ms} k_{ms} < k_s \quad \forall s \in S \quad (6.5)$$

More precisely, the overloading constraints will prevent the execution of too many tasks on a given surrogate node based on its CPU and memory specifications.

Remote Task Completion Time Constraints

Intuitively, when a resource intensive task from a mobile device m is offloaded to a resource rich surrogate node s , then the execution time of the task at the surrogate node (t_{ms}) should be smaller than its execution time at the mobile device (t_m). However, there may be a case when the remote task completion time l_{ms} is greater than the local execution time t_m . This is due to the fact that time is spent in transferring the input/output data of the task to/from the surrogate node. Therefore, when offloading a task, it is a user-defined metric that determines how much of the increased remote task completion time is tolerable as compared with its local execution time. In this model, a fixed delay tolerance parameter λ is considered for every task. It defines the tolerable remote completion time of the tasks compared with their local

execution time. The remote completion time constraints are given as follows.

$$l_{ms} < (1 + \lambda)t_m \quad \forall m \in M, s \in S \quad (6.6)$$

In the model, parameter λ is a unit-less quantity and it is expressed as a percentage of the local execution time of the tasks. For example, suppose the delay tolerance (λ) for a task is 20%. It means that when the task is executed at a remote location the user is willing to accept the remote completion time of the task equal to 20% increase in the local execution time of the task.

Multiple Assignment Constraints

This constraint stipulates that a given task from mobile device m is not offloaded to multiple surrogate nodes. The multiple assignment constraints are as follows.

$$\sum_{s \in S} x_{ms} \leq 1 \quad x \in \{0, 1\}, \quad \forall m \in M \quad (6.7)$$

6.2 Deriving Evaluation Settings

Generally, a resource intensive task is considered for offloading. An offloading task could be either CPU, memory, or I/O resource intensive, or a combination of these resources. When a task executes locally on the mobile device then the local execution

time of the task is considered to determine the energy consumption in the mobile device. However, when a task executes at a remote location then the input and output data sizes of the task are considered to determine the energy consumption in the mobile device. Therefore, to evaluate the model, four metrics of the tasks are considered, which are: (i) local execution time, (ii) resource intensiveness (CPU or memory intensive, or both), (iii) input data size, and (iv) output data size.

The tasks are divided into three types based on local execution time such as low, medium and heavy execution time tasks. Further, each type of task could be either CPU or memory intensive, or both; each type of task could have input and output data sizes either small or large, or one data size could be small and the other could be large. The following sections describe the values set for various parameters of the model.

6.2.1 Execution Times & I/O Data Sizes

Local Execution Time

The range for low, medium and high local execution time tasks is set, as shown in Table 6.1.

Remote Execution Time

The remote execution time (t_{ms}) of a task from mobile device (m) when it is executed on a surrogate node (s) is based on its local execution time (t_m) and the speed-up

Table 6.1: Local Execution Time Ranges

Type of Task	Local Execution Time Range
Low	< 10 sec.
Medium	10 - 20 sec.
High	> 20 sec.

factor (f) of the surrogate node with respect to the mobile device. It is expressed by the following equation.

$$t_{ms} = \frac{t_m}{f} \quad (6.8)$$

In the evaluation setting for the model, it is assumed that all surrogate nodes in set S are four times faster than each of the mobile devices in set M (i.e. $f = 4$).

I/O Data Sizes

The small input or output data sizes are set in a range of 5bytes – 10bytes. However, large data sizes are set in a range of 10kbytes – 500kbytes.

6.2.2 Remote Completion Time

When a task from mobile device m is executed on surrogate node s , then the task's remote completion time l_{ms} includes: t_{send} , time to send input data for the task; t_{ms} , time to remotely execute the task, and; t_{rec} , time to receive the results back from

the surrogate node. The remote completion time can be expressed by the following equation.

$$l_{ms} = t_{send} + t_{ms} + t_{rec} \quad (6.9)$$

Section 6.2.1 shows how the remote execution time (t_{ms}) for a task is set. Intuitively, during data transfer, collisions occur at the lower layers of the network and re-transmissions take place. Consequently, due to re-transmissions, additional time is incurred for the data transfer time. However, in the evaluation settings, when estimating the transfer time for piece of data, re-transmissions are not considered. Therefore, it is assumed that transfer time t_{send} or t_{rec} linearly depends on the data size and the available data rate. The available data rate for an offloading task is set based on its amount of data. The available data rate when transferring small amount of data is lower than when transferring large amount of data. This fact is due to the slow TCP sliding window when transferring small amount of data [38]. The data rate values set for the tasks based on their data size are shown in Table 6.2.

Table 6.2: Data Rates at different Data Sizes

Data Size (KB)	Data Rate (KB/s)
< 10	500
between 10-50	700
> 50	900

6.2.3 Energy Consumption

In general, energy consumption (E) is the product of the current (I) drawn across a voltage (V) for a given time (t). It is given by the following relation [105].

$$E = VIt \quad (6.10)$$

Therefore, power rating (VI) can be defined by the following equation.

$$PowerRating(VI) = \frac{E \text{ (in J)}}{t \text{ (in s)}} \quad (6.11)$$

The power rating of a WiFi radio in the send/receive states is estimated based on its current rating in these states and the voltage rating of the mobile device's battery. The current rating assumed during the send/receive states is $0.0857A$ and $0.0528A$ respectively [105], and the voltage rating of the mobile device's battery is $3.8V$. It is also assumed that the power rating of a CPU in the compute state is higher than the send or receive state of the WiFi radio. An arbitrary value for the CPU power rating in the compute state is used. Based on the above assumptions, the power ratings of WiFi radio and CPU in their respective states are shown in Table 6.3.

Remote Energy Consumption

The remote energy consumption (e_{ms}) includes: e_{send} , energy consumed when sending input data; e_{Idle} , energy consumed when in idle state; and e_{rec} , energy consumed

Table 6.3: Power Ratings of WiFi Radio & CPU at different states

Device	State	Power Rating (J/s)
WiFi Radio	Send	0.325
WiFi Radio	Receive	0.2
CPU	Compute	0.7

when receiving output data. The remote energy consumption can be expressed by the following equation.

$$e_{ms} = e_{send} + e_{Idle} + e_{rec} \quad (6.12)$$

Energy consumption in the idle state (e_{Idle}) of a mobile device (while waiting for the results) is negligible and can be ignored. Moreover, during this time the mobile device can do other tasks. Based on (6.10), the energy consumption in a mobile device when in send and receive state is given by the following equations.

$$e_{send} = t_{send} \times PowerRating_Send \quad (6.13)$$

$$e_{rec} = t_{rec} \times PowerRating_Receive \quad (6.14)$$

As mentioned in Section 6.2.2, when estimating transfer time (t_{send} or t_{rec}) for piece of data, the re-transmission time is not considered. Therefore, energy consumption e_{send} (6.13) or e_{rec} (6.14) does not include the energy consumption during re-transmissions.

Recently available WiFi 802.11ac provides a higher data rate than the previously available WiFi technologies (e.g. 802.11a/b/g/n). Therefore, with the availability of a high data rate, offloading tasks will incur less data transfer time. Consequently, the energy consumption during task offloading (e_{send} (6.13) or e_{rec} (6.14)) will be low when using 802.11ac. Moreover, higher data rates can facilitate the offloading of data intensive tasks even when lower response times are required.

Local Energy Consumption

Based on (6.10), the estimation of local energy consumption when task of mobile device m is executed locally on the mobile device is given by the following equation.

$$e_m = t_m \times PowerRating_Compute \quad (6.15)$$

6.2.4 Processing Power & Memory

The CPU power (p_s) and the memory capacity (k_s) of all the surrogate nodes are set as 100%. However, p_{ms} - the percentage of CPU power (p_s) and k_{ms} - the percentage of memory (k_s), currently used by the task from mobile device m on surrogate node s is set according to the type of the task (low, medium and heavy execution time), and resource intensiveness of the task, as shown in Tables 6.4, 6.5 and 6.6.

Table 6.4: Values of p_{ms} and k_{ms} for a low execution time task

Resource Intensive	p_{ms}	k_{ms}
CPU	10%	10%
Memory	10%	10%
CPU and Memory	10%	10%

Table 6.5: Values of p_{ms} and k_{ms} for a medium execution time task

Resource Intensive	p_{ms}	k_{ms}
CPU	30%	10%
Memory	10%	30%
CPU and Memory	30%	30%

Table 6.6: Values of p_{ms} and k_{ms} for a heavy execution time task

Resource Intensive	p_{ms}	k_{ms}
CPU	50%	10%
Memory	10%	50%
CPU and Memory	50%	50%

6.3 Results

To implement and evaluate the TSP Integer Linear Program (ILP) IBM's linear programming solver CPLEX [1] is used with all the default parameters. The task scheduling problem is solved with multiple data inputs. The evaluation of the model is performed on a single server machine having Intel Xeon(R) E5420 @ 2.50GHz, quad core CPU and 8GB of RAM. The Linux OS distribution on the server is Ubuntu 12.04.2 LTS precise 64bit.

The problem sizes for the task scheduling problem are generated based on the number of surrogate nodes and the number of mobile devices. The number of surrogate nodes ($|S|$) is varied from 5, to 10, 15, and 20. For each number of surrogate nodes the number of mobile devices ($|M|$) is varied from 5, to 10, 15, 20, 25, 30, 35, 40, 50, 60, 70, 80, 90, 100, 150, 200, 250, and 300. The task scheduler model optimizes the total energy consumption across all mobile devices of the system. The work only deals with the energy consumed in the offloading process and the energy consumed in processing the tasks. The performance of the task scheduler is evaluated by observing the total energy consumption across all the mobile devices, in three scenarios, which are as follows.

1. **Energy consumed without offloading:** The first scenario provides a baseline for the total energy consumption across all mobile devices $|M|$. In this case, the broker-node is not involved in scheduling task offloading, rather, all tasks

are processed locally on mobile devices. Therefore, $x_{ms} = 0$ for every task, and the total energy consumption across all mobile devices is given by (6.2) when $x_{ms} = 0$. Thus, the cost function $C(X)$ is the sum of the local energy consumption (e_m) by tasks across all the mobile devices. The cost function $C(X)$ in this scenario is given by (6.16).

$$C(X) = \sum_{m \in M} e_m \quad (6.16)$$

2. **Energy consumed with offloading without optimization:** In the second scenario, the task scheduler in the centralized broker-node schedules tasks without optimizing the total energy consumption. The values (x_{ms}) in the set X will be obtained using one-by-one scheduling of all tasks in sequence while satisfying constraints (6.4), (6.5), (6.6), and (6.7). A given task will be offloaded to the first available surrogate node, provided that the task satisfies all the constraints. The total energy consumption in this case is expressed by (6.2).
3. **Energy consumed with offloading with optimization:** In the third scenario, the task scheduler at the centralized broker-node utilizes the proposed task scheduler model. It schedules tasks while optimizing energy consumption across all mobile devices (6.3). The values (x_{ms}) in the set X will be obtained as a result of the optimization process with the aim to minimize the total energy consumption across all mobile devices while satisfying constraints (6.4),

(6.5), (6.6), and (6.7). The difference between the second and third scenarios depends on the offloading decisions (i.e. the values of x_{ms}). The total energy consumption in this case is also given by (6.2). Intuitively, a lower energy consumption is expected than the other two scenarios since the optimization process will offload tasks that will allow the best savings.

6.3.1 Analysis of Results

The graphs in Figures 6.2(a), 6.2(b), 6.2(c), and 6.2(d) show the total energy consumption across all mobile devices in three scenarios described earlier. The results show that the amount of energy consumed when there is no offloading (i.e. the black line) is always higher than the other two scenarios. This was expected since this is the actual amount of energy consumed when tasks are processed locally on the mobile devices. When the centralized broker-node is added to the architecture and the broker-node is allowed to offload tasks to surrogate nodes (i.e. green and red lines) then the amount of energy consumed is reduced significantly. It is also observed that, for a given number of surrogate nodes $|S|$, the amount of energy consumption is the same up to a certain number of mobile devices $|M|$. The reason for this similar consumption is that, when the number of tasks to be offloaded is lower than the number of tasks that can be handled by the surrogate nodes, the scheduler will always be able to offload the tasks. However, as the number of tasks is increasing, it is evident that carefully selecting which tasks to offload can provide a significant advantage. For ex-

Figure 6.2: Total energy consumption across all mobile devices.

ample, when $|S| = 15$ and $|M| = 150$ and the scheduler is used without optimization, the energy consumed will be $955J$ versus $535J$ when using the proposed scheduler with optimization. This is a difference of $\sim 44\%$.

In the first scenario, in which there is no broker, the mobile devices can still potentially offload by searching for a surrogate node on their own. Then, the total energy consumption in the system will approach the total energy consumption when tasks are offloaded without optimization (i.e. the green line). However, in this scenario, the mobile devices have to perform resource monitoring individually, which causes time and communication overhead in a large RAE. In summary, the performance of the proposed task scheduler model is better than a task scheduler which offloads tasks without optimization.

6.4 Conclusion

In this chapter, handling task scheduling at a centralized broker-node on behalf of all the mobile devices in a large RAE was proposed. The service nodes in the RAE are accessible through a local network. A mathematical formulation for the task scheduling problem was proposed to find an optimal solution for the problem (task assignment) such that the total energy consumption across all mobile devices could be minimized. The results show that, using a task scheduler model at the centralized broker-node, the total energy consumption across all the mobile devices is less than the

total energy consumption when tasks are offloaded using a centralized task scheduler without optimization.

In the next chapter, the task scheduler model for the local resources case is extended to a task scheduler model for the mobile cloud computing case.

Chapter 7

Energy Optimization: The Mobile Cloud Computing Case

In Chapter 6, a task scheduler model was proposed for the centralized task scheduling problem in a large RAE when service nodes were available in a local network. The model provides optimal solutions for the task assignment problem by minimizing the total energy consumption across all mobile devices in the system. In this chapter, the centralized task scheduler model presented in Chapter 6 is modified such that it can be applied to MCC environments. This task scheduler model, like the previous model in Chapter 6, also minimizes the total energy consumption across all mobile devices of the system. However, compared with the previous model, this model is subject to different constraints. The main differences between the two task scheduler models are with respect to the MCC environment. In MCC, the service nodes, referred to as VMs,

are located in public clouds and are accessed through the Internet. In this model, it is assumed that cloud providers have unlimited computing resources (VMs) in contrast to limited computing resources available from surrogate nodes. Therefore, this model is not subject to overloading constraints. In the previous model, the available data rate between the mobile devices and the surrogate nodes, and the delay tolerance for tasks were fixed. However, in this model, delay tolerance for every task is different, and this, in turn, defines different constraints for the minimum required data rate for tasks at a given input/output data size. More precisely, a mathematical model for the centralized task scheduling problem in MCC environments is proposed. The task scheduler model finds optimal solutions for the task assignment problem and provides a significant reduction in the total energy consumption compared with the total energy consumption when tasks are offloaded from the centralized scheduler without optimization.

7.1 Task Scheduler Model

Figure 7.1 illustrates task offloading from a mobile device onto a VM from a cloud. The figure shows the various cost and constraints parameters associated with the mobile devices and VMs. The model uses these constraint parameters to find an optimal solution for the task scheduling problem. To this end, a mathematical model based on the following assumptions and notation is proposed.

Figure 7.1: Resource augmentation environment for MCC.

7.1.1 Assumptions

- To simplify initial analysis, it is assumed that each mobile device has a single task to offload/execute at any given time.

- All the mobile devices and the broker-node communicate via a WiFi network.

The mobile devices and the broker-node access cloud resources through the Internet connectivity via a WiFi access point (Figure 7.1).

- The monetary cost of renting computation resources (VMs) and transferring data to/from the clouds is not considered in the model
- The clouds have infinite infrastructure resources. Therefore, none of the tasks is denied from offloading due to lack of available resources at the clouds.
- It is assumed that all mobile devices in the system have the same data rate to a given VM. In an actual MCC environment the mobile devices may experience different data rate to a VM. In this model, assuming different values for data rate to a VM will only change the results; however, the model will not be affected.

7.1.2 Notation

The following notation is composed of sets, cost parameters, constraints parameters and decision variables of the model.

Sets

- $\mathcal{M}$, the set of all mobile devices, where mobile device $m \in \mathcal{M}$.
- $\mathcal{V}$, the set of all VMs available from clouds, where VM $v \in \mathcal{V}$.
 - b_v , available data rate between all mobile devices in $\mathcal{M}$ and VM v . All the mobile devices have the same data rate to a given VM.

Cost Parameters

- e_m , energy consumption in mobile device m when its task is executed locally on the mobile device.
- e_{mv} , energy consumption in mobile device m when its task is executed remotely on VM v .

Constraint Parameters

- t_m , execution time of task from mobile device m when executed locally on the mobile device.
- t_{mv} , execution time of task from mobile device m when executed remotely on VM v .
- μ_m , amount of input data required from the task of mobile device m .
- ρ_m , amount of output data generated by the execution of the task from mobile device m .
- d_{mv} , delay of the task from mobile device m when executed on VM v . The task's delay includes: t_{send} , time to send input data (μ_m) of the task; t_{mv} , time to remotely execute the task; and t_{rec} , time to receive the output data (ρ_m) from the instance. The delay can be expressed by the following equation.

$$d_{mv} = t_{send} + t_{mv} + t_{rec} \quad (7.1)$$

- λ_m , delay tolerance parameter of the task from mobile device m when executed on a VM.

Decision Variables

- ψ_{mv} , binary variable such that $\psi_{mv} = 1$ if and only if the task from mobile device $m \in \mathcal{M}$ is offloaded to VM $v \in \mathcal{V}$; otherwise, $\psi_{mv} = 0$. The task runs on mobile device m , if and only if

$$\sum_{v \in \mathcal{V}} \psi_{mv} = 0 \quad (7.2)$$

The values of ψ_{mv} are subject to the multiple assignment constraints (7.7), which are explained later in Subsection 7.1.4. Ψ is the set of all the decision variables, where a decision variable $\psi_{mv} \in \Psi$, $m \in \mathcal{M}$ and $v \in \mathcal{V}$.

7.1.3 Cost Function

The cost function $\mathcal{G}(\Psi)$ (7.3) represents the total energy consumption across all mobile devices whether executing their tasks locally on the mobile devices or remotely on VMs.

- The first term in the cost function represents the total energy consumption by the tasks that are offloaded to VMs. In this case, the task assignment decision variable $\psi_{mv} = 1$. It represents the energy consumed e_{mv} when the task from mobile device $m \in \mathcal{M}$ is offloaded to VM $v \in \mathcal{V}$.

- The second term represents the total energy consumption by the tasks that are executed locally on the mobile devices. In this case, the task assignment decision variable $\psi_{mv} = 0$. It represents the energy consumption e_m when the task from mobile device $m \in \mathcal{M}$ is executed locally on the mobile device.

$$\mathcal{G}(\Psi) = \sum_{\substack{m \in \mathcal{M} \\ v \in \mathcal{V}}} e_{mv} \psi_{mv} + \sum_{\substack{m \in \mathcal{M} \\ v \in \mathcal{V}}} e_m (1 - \psi_{mv}) \quad (7.3)$$

7.1.4 The Model

The objective function of the Task Scheduling Problem for Cloud Computing Environment (TSPCCE) is to minimize the total energy consumption, which is equivalent to maximizing the total energy saving across all the mobile devices. The objective function of the model is as follows.

$$\min \quad \mathcal{G}(\Psi) \quad (7.4)$$

The model is subject to various constraints (7.6) and (7.7).

Data Rate Constraints

The delay tolerance (λ_m) for a task is the user-defined constraint when offloading the task to a remote location. It puts a limit on the delay (d_{mv}) of the task (7.1). The delay constraint is given by the following inequation.

$$d_{mv} \leq \lambda_m t_m \quad (7.5)$$

Intuitively, during data transfer, collisions occur at the lower layers of the network and re-transmissions take place. Consequently, due to re-transmissions additional time is incurred with the data transfer time. However, in the evaluation settings, when estimating the transfer time for a piece of data, re-transmissions are not considered. Therefore, it is assumed that the transfer time for the data linearly depends on the data size and the available data rate. Let b_v be the available data rate between a mobile device and VM v . Then, the time to send (t_{send}) the input data (μ_m) for the task can be expressed as $\frac{\mu_m}{b_v}$. Similarly, the time to receive (t_{rec}) the output data (ρ_m) generated from the execution of the task at the VM can be expressed as $\frac{\rho_m}{b_v}$. After solving (7.1) and (7.5) for b_v , the data rate constraints can be defined by the following inequation.

$$b_v \geq \frac{\mu_m + \rho_m}{\lambda_m t_m - t_{mv}} \quad \forall m \in \mathcal{M}, \forall v \in \mathcal{V} \quad (7.6)$$

Multiple Assignment Constraints

These constraints stipulate that a given task from mobile device m is not offloaded to multiple VMs. The multiple assignment constraints are given by the following

inequation.

$$\sum_{v \in \mathcal{V}} \psi_{mv} \leq 1 \quad \psi \in \{0, 1\}, \quad \forall m \in \mathcal{M} \quad (7.7)$$

7.2 Deriving Evaluation Settings

In computation offloading, generally, resource intensive tasks are considered for offloading. A task could be either CPU, memory or I/O resource intensive, or a combination of these resources. The tasks considered for scheduling could be intensive in either of the resources, or a combination of these. For example, a task could be CPU and input data intensive, and another task could be only input and output data intensive. The type of resource intensiveness in a task determines the energy consumption on the mobile device during local execution on the mobile device and during remote execution on a virtual instance. When a task is CPU or memory intensive, it accounts for the energy consumption during its local execution at the mobile device. The input and output data intensiveness of a task accounts for the energy consumption in the mobile device during the task offloading. The TSPCCE problem is solved with multiple data inputs, and the various values set are explained in the following sections.

7.2.1 Task Execution Times & I/O Data Sizes

Local Execution Time

The local execution time of the tasks for the mobile devices in set $\mathcal{M}$ are according to a uniform distribution $\mathcal{U}(a, b)$ within the range $a = 0$ to $b = 100$ seconds.

Remote Execution Time

The remote execution time (t_{mv}) of a task from mobile device (m) when it is executed on a VM (v) is based on its local execution time (t_m) and the speed-up factor ($\mathcal{F}$) of the VM with respect to the mobile device. It is expressed by the following equation.

$$t_{mv} = \frac{t_m}{\mathcal{F}} \quad (7.8)$$

It is assumed that the VMs in set $\mathcal{V}$ are four times faster than each of the mobile devices in set $\mathcal{M}$ (i.e. $\mathcal{F} = 4$).

I/O Data Sizes

The input and output data sizes (in MB) of a task are set according to a Uniform distribution $\mathcal{U}(a, b)$ within the range $a = 0$ to $b \in \{1, 10, 20, 30, 40\}$ MB.

7.2.2 Delay Tolerance

The delay tolerance parameter is a user-defined parameter. It is a unit-less quantity. When a task is offloaded, the value of delay tolerance for the task sets a limit on the completion time of the task. From (7.1), (7.5) and (7.8), the delay tolerance can be given by the following inequation.

$$\lambda_m \geq \frac{1}{\mathcal{F}} + \frac{\mu_m}{b_v t_m} + \frac{\rho_m}{b_v t_m} \quad (7.9)$$

The minimum value of the delay tolerance parameter for a task is defined when the input (μ_m) and output (ρ_m) data sizes of the task are negligible. In this case, delay tolerance can be given as $\lambda_m \geq \frac{1}{\mathcal{F}}$. The values of the delay tolerance parameter of the tasks across all mobile devices are set according to a Uniform distribution $\mathcal{U}(a, b)$ within the range $a = \frac{1}{\mathcal{F}}$ to $b \in \{0.3, 0.5, 1.0, 1.5, 2.0\}$.

7.2.3 Data Rate

The data rates available between mobile devices and VMs are set according to a normal distribution $\mathcal{N}(\mu, \sigma)$ with a mean μ of $500KB/s$ and standard deviation σ of $300KB/s$. Higher data rates are available when accessing public clouds through the Internet using LTE cellular networks compared with when using 3G networks. Therefore, the data transfer times for the offloading tasks will be lower, and consequently, energy consumption due to data transfer will be lower (e_{send} (6.13) or e_{rec} (6.14)) as

well. In this model, when considering higher data rates, more tasks can satisfy the data rate constraints (7.6) and the total energy consumption will be lower.

7.2.4 Energy Consumption

The power rating of each mobile device's (in set $|\mathcal{M}|$) WiFi radio and CPU in their respective states is given by Table 6.3.

Remote Energy Consumption

The remote energy consumption (e_{mv}) includes the following parameters: e_{send} , energy consumption when sending input data; e_{Idle} , energy consumption when in idle state; and e_{rec} , energy consumption when receiving output data. The remote energy consumption can be expressed by the following equation.

$$e_{mv} = e_{send} + e_{Idle} + e_{rec} \quad (7.10)$$

Energy consumption in the idle state (e_{Idle}) of a mobile device (while waiting for the results) is negligible and can be ignored. Moreover, during this time the mobile device can do other tasks. The energy consumption in send (e_{send}) and receive (e_{rec}) states is given by (6.13) and (6.14) respectively.

Local Energy Consumption

The estimation of local energy consumption when a task is executed locally on the mobile device is given by (6.15).

7.3 Results & Analysis

The TSPCCE Integer Linear Program (ILP) is implemented by using IBM's linear programming solver called CPLEX [1] with all the default parameters. The evaluation of the model is performed on a single server machine having Intel Xeon(R) E5420 @ 2.50GHz, quad core CPU and 8GB of RAM. The Linux OS distribution on the server is Ubuntu 12.04.2 LTS precise 64bit.

The problem sizes for the centralized task scheduling problem are generated based on the number of VMs and the number of mobile devices. The number of mobile devices ($|\mathcal{M}|$) is varied by the values 20, 40, 60, 80, 100, 150, 200, 250, and 300 for each number of VMs. In the model, it is assumed that an infinite number of VMs are available. However, to generate results and to make sure that none of the mobile devices is denied offloading due to a lack of resources, the number of VMs is varied by the values 400, 500 and 600. To evaluate the performance of the task scheduler the energy consumption across all mobile devices is observed in three scenarios, which are explained as follows.

1. **Energy consumed without offloading:** The first scenario provides a baseline for the total energy consumption across all mobile devices $|\mathcal{M}|$. In this case, the broker-node is not involved in scheduling task offloading, rather all tasks are processed locally on the mobile devices. Therefore, $\psi_{mv} = 0$ for every task, and the total energy consumption across all mobile devices is given by (7.3) when $\psi_{mv} = 0$. Thus, the cost function $\mathcal{G}(\Psi)$ is the sum of the local energy consumption (e_m) of all the tasks across all the mobile devices. The cost function $\mathcal{G}(\Psi)$ in this scenario is given by (7.11).

$$\mathcal{G}(\Psi) = \sum_{m \in M} e_m \quad (7.11)$$

2. **Energy consumed with offloading without optimization:** In the second scenario, the task scheduler at the centralized broker-node schedules tasks but without optimizing the total energy consumption. The values (ψ_{mv}) in the set Ψ will be obtained using one-by-one scheduling of all tasks in sequence while satisfying constraints (7.6) and (7.7). A given task will be offloaded to the first available VM provided the task satisfies all the constraints. The total energy consumption in this case is expressed by (7.3).
3. **Energy consumed with offloading with optimization:** In the third scenario, the task scheduler at the centralized broker-node schedules tasks while optimizing the total energy consumption across all mobile devices (7.4). The

values (ψ_{mv}) in the set Ψ will be obtained as a result of the optimization process with the aim to minimize the total energy consumption across all the mobile devices while satisfying constraints (7.6) and (7.7). The total energy consumption in this case is also given by (7.3).

In the second and third scenarios, the effect on the total energy consumption across all mobile devices $|\mathcal{M}|$ is observed based on: (i) the number of VMs ($|\mathcal{V}|$), (ii) the size of the input (μ_m) and output (ρ_m) data, and (iii) the delay tolerance parameter (λ_m). In all sections of the results, the total energy consumption across all the mobile devices ($|\mathcal{M}|$) is the average over 30 iterations. In the results, it is observed that the difference in the total energy consumption in the three scenarios is very big; therefore, it is shown using a log scale.

7.3.1 Effect of the Number of Virtual Machines

The effect of (i) the number of VMs, and (ii) offloading with and without optimization, is observed on the total energy consumption. To observe this effect, the number of VMs ($|\mathcal{V}|$) is varied by the values 400, 500 and 600. The amount of input (μ_m) and output (ρ_m) data (in MB) and the delay time tolerance parameter are set according to a Uniform distribution $\mathcal{U}(0, 1)$ and $\mathcal{U}(\frac{1}{f}, 1.0)$ respectively. It is assumed that each mobile device has one task to offload/execute at a given time. Also, one VM executes one task at a given time. Thus, the number of VMs ($|\mathcal{V}|$) could be greater

Figure 7.2: Total energy consumption across all mobile devices.

than the maximum number of mobile devices (i.e. $|\mathcal{M}| = 300$), and this is considered in the system. The results show that the total energy consumption in the second or third scenario does not vary much when the number of VMs is varied by the values 400, 500 and 600. Thus, in Figure 7.2, the results are shown only when the number of VMs is 400 (i.e. $|\mathcal{V}| = 400$).

The results (Figure 7.2) show that the total amount of energy consumption when tasks are processed locally on mobile devices (i.e. black line) is always higher than the other two scenarios. A significant reduction in energy consumption is observed (i.e. green and red line), when a centralized broker-node is added in the architecture and it is allowed to perform task scheduling on behalf of all mobile devices. Further, a difference in the total energy consumption is observed when tasks are offloaded with-

out optimization (i.e. green line) and with optimization (i.e. red line). The results suggest that the proposed task scheduler minimizes the total energy consumption across all mobile devices by carefully offloading tasks to VMs. For example, when $|\mathcal{M}| = 200$, if the scheduler is used without optimization the energy consumption will be $169.61J$ versus $39.17J$ when using the scheduler with optimization. This represents an improvement of 77%.

7.3.2 Effect of Input & Output Data Sizes

The effect of the input & output data sizes of the offloading tasks is observed on the total energy consumption. The number of VMs ($|\mathcal{V}|$) is set at 400, and the delay tolerance parameter (λ_m) is set according to a Uniform distribution $\mathcal{U}(\frac{1}{\mathcal{F}}, 1.0)$. The input (μ_m) and output (ρ_m) data sizes (in MB) are set according to a Uniform distribution $\mathcal{U}(0, b)$, $b \in \{10, 20, 30, 40\}$. The results in Figures 7.3(a), 7.3(b), 7.3(c) and 7.3(d) show that, as the size of input and output data increases the total energy consumption across all mobile devices increases. Also, with the increase in data size the total energy consumed is approaching the energy consumed in the non-offloading scenario (i.e. black line). This observation suggests that, though task offloading saves on computation energy consumption in the mobile devices, transferring the task and its related data incurs energy consumption. Therefore, offloading data intensive tasks may or may not be beneficial when energy saving is the goal of task offloading.

Figure 7.3: Total energy consumption across all mobile devices.

7.3.3 Effect of Delay Tolerance

In this section, the effect of delay tolerance is observed on the total energy consumption across all mobile devices in the third scenario, i.e. offloading with optimization. The number of VMs ($|\mathcal{V}|$) is set at 400. The delay tolerance parameter (λ_m) is set according to a Uniform distribution $\mathcal{U}(\frac{1}{\mathcal{F}}, b)$, $b \in \{0.3, 0.5, 1.0, 1.5, 2.0\}$. The effect of the delay tolerance of the tasks at different data sizes is observed on the total energy consumption when offloading with optimization (i.e. the third scenario). The delay tolerance of a task is the user-defined parameter to set acceptable delay for the task when executed at a remote location. The value of λ_m does not affect the energy consumption of the task, rather it only affects the constraints (7.6).

In Figure 7.4(a), the input/output data sizes (in MB) are set in a range $\mathcal{U}(0, 1)$. The results show that the total energy consumptions for various values of λ_m are almost similar. The reason for having the same total energy consumption could be that the data sizes are small, and for every value of λ_m the constraints (7.6) are satisfied. Thus, in this case, almost all the tasks of the mobile devices are offloaded. Further, to see the effect of λ_m when there are higher data sizes, the data sizes (in MB) are set in a range $\mathcal{U}(0, 20)$ (Figure 7.4(b)) and $\mathcal{U}(0, 30)$ (Figure 7.4(c)). The results show that, at higher data rates, the amount of energy consumption increases as the value of λ_m becomes small. The results in Figures 7.4(b) and 7.4(c) reveal that when input/output data sizes are large and λ_m is small then the number of tasks that do not satisfy the constraints of (7.6) is high. In this case, the number of tasks

(a) μ_m & ρ_m (in MB) = $\mathcal{U}(0, 1)$.(b) μ_m & ρ_m (in MB) = $\mathcal{U}(0, 20)$. (c) μ_m & ρ_m (in MB) = $\mathcal{U}(0, 30)$.Figure 7.4: The effect of delay tolerance (λ_m) on the total energy consumption.

that are executed locally increases, and this leads to an increase in the total energy consumption.

7.4 Conclusion

In this chapter, a mathematical model for the centralized task scheduling problem in a MCC case was proposed. The task scheduler model finds an optimal solution for the problem (task assignment) and minimizes the total energy consumption across all mobile devices in the system. The results show that using the task scheduling model, the total energy consumption across all mobile devices is less than the total energy consumption when tasks are offloaded using a centralized task scheduler without optimization.

In the next chapter, a generalized task scheduler model is constructed to optimize the total energy consumption and the total monetary cost for RAEs in MCC.

Chapter 8

A Generalized Model for Energy and Monetary Cost Optimization

In previous chapters, task scheduler models for the proposed centralized broker-node architecture were presented. The models found an optimal solution for the centralized task scheduling problems and minimized the total energy consumption across all mobile devices in the local resources case (Chapter 6) and the mobile cloud computing case (Chapter 7). In this chapter, the previous task scheduler models for the total energy minimization are extended to consider the total monetary cost minimization as well, across all mobile devices in a large RAE. The proposed model is evaluated in two RAEs for MCC. In the first environment, computation resources are available from a local private cloud accessible through a WiFi network. In the second environment, computation resources are available from public clouds accessible through

the Internet. More precisely, a mathematical model is proposed for the centralized task scheduling problem (task assignment), which optimally minimizes: (i) the total energy consumption when applied to a local private cloud, and (ii) the total energy consumption and the total monetary cost when applied to public clouds. In the model, user-defined delay tolerance is considered for every task, which puts a limit on the delay of the offloaded task. Therefore, delay tolerance defines the constraints for the minimum required data rate for each task at a given input/output data size of the tasks. The extended model considers another enhancement from the previous models, in which a single task per mobile device was assumed at any given time to offload/execute. However, in this model, multiple tasks per mobile device are assumed. The task scheduler model at the centralized broker optimally offloads tasks and provides significant reduction in the total energy consumption and monetary cost compared with when tasks are offloaded from the centralized scheduler without optimization.

8.1 Resource Augmentation Environments

The proposed task scheduler model is evaluated in two RAEs for MCC, as shown in Figure 8.1. In the first environment, computation resources are available from a local private cloud, which are accessible to mobile devices through a WiFi network, as shown in Figure 8.1(a). On the other hand, computation resources in the sec-

ond environment are available from public clouds, which are accessible through the Internet, as shown in Figure 8.1(b).

The basic paradigm of the two RAEs is to provide computation resources to mobile devices. However, the two paradigms differ in providing computation resources, as explained below.

- **Paying for resources:** Mobile users do not pay when using computation resources from the local private cloud; however, they pay when using resources from the public clouds. Thus, offloading a task onto the local private cloud involves only the energy consumption when transferring data. On the other hand, offloading onto the public clouds involves: (i) energy consumption incurred when transferring data, and (ii) monetary cost incurred when using computing resources per unit time, and transferring data per unit bytes.
- **Availability of resources:** The computation resources available from the local private cloud are limited. Therefore, when a large number of mobile devices requests task offloading, at some point some tasks will be denied offloading due to the finite amount of resources. On the other hand, infinite computation resources are available from public clouds. Thus, when offloading tasks from any number of mobile devices, none of the tasks will be denied offloading due to a lack of resources.
- **Accessibility of resources:** The computation resources from the local pri-

(a) RAE using a local private cloud.

(b) RAE using public clouds.

Figure 8.1: Resource augmentation environments for MCC.

vate cloud are present in the vicinity of the mobile devices and are accessed through a WiFi network. However, the resources from public clouds are at WAN latency from the mobile devices and are accessed through the Internet. Therefore, the mobile devices experience a higher data rate when accessing resources from the local private cloud than from the public clouds.

8.1.1 Using a Centralized Broker

In our previous work [80] and [79], performing task scheduling at a centralized node on behalf of all mobile devices in the system, was proposed. The centralized node was referred to as broker-node. Also, in this work, the location of the task scheduler in both RAEs is the centralized broker-node, as shown in Figure 8.1. The various steps involved while scheduling tasks from mobile devices using the resources in a cloud are also illustrated in the figure, which include: (1) mobile devices contact the centralized task scheduling service at the broker-node, (2) the task scheduler decides the appropriate offloading location on behalf of the mobile devices by minimizing the total energy consumption and the total monetary cost across all mobile devices subject to various constraints, and (3) based on the task scheduling decision, each task is either offloaded to resources on the cloud or it is executed locally on the mobile device. Multiple broker-nodes, load balancing, and reliability should be part of the system design in order to avoid having a single point of failure. However, these issues of load balancing and reliability along with security and access authentication

to cloud providers are outside the scope of this work.

8.2 Task Scheduler Model

The proposed task scheduler model is evaluated in two different RAEs for MCC, as shown in Figure 8.1. When applied to RAE using a local private cloud (Figure 8.1(a)), the model finds an optimal solution for the total energy consumption across all mobile devices in the system. When applied to RAE using public clouds (Figure 8.1(b)), the model finds an optimal solution for the total energy consumption and the total monetary cost across all mobile devices in the system. To this end, a mathematical model based on the following assumptions and notation is proposed.

8.2.1 Assumptions

- Each mobile device in the system has multiple tasks. Based on the task offloading decision, a task is executed either locally on the mobile device or on cloud resources, independently of other tasks in the mobile device.
- In RAE using the local private cloud, all the mobile devices, the broker-node, and the cloud nodes communicate through a WiFi network, as shown in Figure 8.1(a).
- In RAE using public clouds, all the mobile devices and the broker-node com-

municate through a WiFi network. The mobile devices and the broker-node access resources from the clouds through the Internet via a WiFi access point, as shown in Figure 8.1(b).

- In both RAEs, the resource monitoring process in the broker-node periodically contacts the clouds to get their up-to-date resource description. Thus, the broker knows the current status of the resources from the clouds.
- The public clouds have infinite computation resources (VMs). Therefore, none of the tasks is denied offloading due to a lack of available resources at the clouds. However, the local private cloud has limited resources and some tasks could be denied offloading (even if they satisfy the other constraints) due to a lack of resources.
- In both RAEs, VMs of the same instance types across all the cloud providers have the same configuration. The configuration of the various instance types of VMs are explained later in Table 8.1.
- It is assumed that all mobile devices in the system have the same data rate to a given VM instance, in both RAEs. In an actual MCC environment the mobile devices may experience different data rate to a VM. In this model, assuming different values for data rate to a VM will only change the results; however, the model will not be affected.

8.2.2 Notation

The following notation is composed of sets, cost parameters, constraints parameters, and decision variables of the model.

Sets

- $\mathcal{M}$, the set of all mobile devices, where mobile device $m \in \mathcal{M}$.
- $\mathcal{K}_m$, the set of all tasks in mobile device $m \in \mathcal{M}$, where task $k_{mj} \in \mathcal{K}_m$ and $j : 1 \rightarrow |\mathcal{K}_m|$.
- $\mathcal{C}$, the set of cloud providers, where cloud provider $c \in \mathcal{C}$.
 - β_c , number of CPUs available at cloud provider $c \in \mathcal{C}$.
 - γ_c , amount of memory available at cloud provider $c \in \mathcal{C}$.
- $\mathcal{I}$, the set of all VM instance types in each cloud provider in $\mathcal{C}$, where VM instance type $i \in \mathcal{I}$.
- $\mathcal{V}_{ci}$, the set of all VMs of instance type $i \in \mathcal{I}$ from cloud provider $c \in \mathcal{C}$, where VM $v_{cir} \in \mathcal{V}_{ci}$ and $r : 1 \rightarrow |\mathcal{V}_{ci}|$.
 - $\mathcal{F}_{v_{cir}}$, speed-up factor of VM v_{cir} with respect to the execution speed of each of the mobile device in $\mathcal{M}$.

- $b_{v_{cir}}$, available data rate between VM v_{cir} and each of the mobile device in $\mathcal{M}$.

Cost Parameters

- $e_{k_{mj}}$, energy consumed (in joules) when task k_{mj} is executed locally on mobile device m .
- $e_{k_{mj}v_{cir}}$, energy consumed (in joules) when task k_{mj} is executed remotely on VM v_{cir} .
- $s_{k_{mj}}$, required number of slots of unit time for task k_{mj} when executed remotely.
- p_c^{in} , monetary cost of unit bytes of inbound traffic to cloud provider $c \in \mathcal{C}$.
- p_c^{out} , monetary cost of unit bytes of outbound traffic from cloud provider $c \in \mathcal{C}$.
- $p_{v_{cir}}^{res}$, monetary cost per unit time period of using VM v_{cir} of instance type $i \in \mathcal{I}$ from cloud provider $c \in \mathcal{C}$.

Constraints Parameters

- $t_{k_{mj}}^{exec}$, execution time of task k_{mj} when executed locally on mobile device $m \in \mathcal{M}$.
- $t_{k_{mj}v_{cir}}^{exec}$, execution time of task k_{mj} when executed remotely on VM v_{cir} .
- $\mu_{k_{mj}}$, amount of input data required to process task k_{mj} .

- $\rho_{k_{mj}}$, amount of output data generated by the execution of task k_{mj} .
- $d_{k_{mj}v_{cir}}$, delay of task k_{mj} when executed on VM v_{cir} . The task's delay includes the following components: $t_{k_{mj}v_{cir}}^{send}$, time to send input data ($\mu_{k_{mj}}$) of the task to the VM; $t_{k_{mj}v_{cir}}^{exec}$, time to remotely execute the task on the VM instance; and $t_{k_{mj}v_{cir}}^{rec}$, time to receive the output data ($\rho_{k_{mj}}$) of the task from the VM instance. The delay can be expressed by the following equation.

$$d_{k_{mj}v_{cir}} = t_{k_{mj}v_{cir}}^{send} + t_{k_{mj}v_{cir}}^{exec} + t_{k_{mj}v_{cir}}^{rec} \quad (8.1)$$

- $\lambda_{k_{mj}}$, delay tolerance of task k_{mj} when executed on a VM instance. It is a unit-less quantity.
- $\beta_{k_{mj}v_{cir}}$, number of CPUs used by task k_{mj} when using VM v_{cir} of instance type $i \in \mathcal{I}$ on cloud provider $c \in \mathcal{C}$.
- $\gamma_{k_{mj}v_{cir}}$, amount of memory used by task k_{mj} when using VM v_{cir} of instance type $i \in \mathcal{I}$ on cloud provider $c \in \mathcal{C}$.

Decision Variables

- $\phi_{k_{mj}v_{cir}}$, is a binary variable such that $\phi_{k_{mj}v_{cir}} = 1$ if and only if task k_{mj} from mobile device m is offloaded to VM v_{cir} , otherwise, $\phi_{k_{mj}v_{cir}} = 0$. The task runs

locally on mobile device m if and only if

$$\sum_{c \in \mathcal{C}} \sum_{i \in \mathcal{I}} \sum_{v_{cir} \in \mathcal{V}_{ci}} \phi_{k_{mj}v_{cir}} = 0 \quad (8.2)$$

The values of $\phi_{k_{mj}v_{cir}}$ are subject to the multiple assignment constraints (8.9), which are explained later in Subsection 8.2.4. Φ is the set of all the decision variables, where a decision variable $\phi_{k_{mj}v_{cir}} \in \Phi$: $m \in \mathcal{M}, k_{mj} \in \mathcal{K}_m; c \in \mathcal{C}, i \in \mathcal{I}, v_{cir} \in \mathcal{V}_{ci}$.

8.2.3 Cost Function

The model optimizes the total energy consumption when it is evaluated in RAE using a local private cloud. However, it optimizes the total energy consumption and the total monetary cost when it is evaluated in RAE using public clouds. The two cost functions considered in the model are explained as follows.

1. The first cost function, denoted as $(\mathcal{E}(\Phi))$ (8.3), represents the total energy consumption across all mobile devices whether executing their tasks locally on the mobile devices or remotely on the clouds .

- The first term represents the total energy consumed by the tasks that are offloaded onto VMs. In this case, the task assignment decision variable $\phi_{k_{mj}v_{cir}} = 1$. It represents the energy consumed $e_{k_{mj}v_{cir}}$ when task $k_{mj} \in$

$\mathcal{K}_m$ from mobile device $m \in \mathcal{M}$ is offloaded to VM $v_{cir} \in V_{ci}$ of instance type $i \in \mathcal{I}$ from cloud provider $c \in \mathcal{C}$.

- The second term is the total energy consumed by the tasks that are executed locally on the mobile devices. In this case, the task assignment decision variable $\phi_{k_{mj}v_{cir}} = 0$. It represents the energy consumed $e_{k_{mj}}$ when task $k_{mj} \in \mathcal{K}_m$ from mobile device $m \in \mathcal{M}$ is executed locally on the mobile device.
2. The second cost function, denoted as $(\mathcal{P}(\Phi))$ (8.4), represents the total monetary cost of the offloaded tasks from all the mobile devices. In this case, the monetary cost of those tasks is considered for which the task assignment decision variable $\phi_{k_{mj}v_{cir}} = 1$.
- The first term represents the total monetary cost of sending and receiving input and output data from clouds. In this case, the task assignment decision variable $\phi_{k_{mj}v_{cir}} = 1$. It represents the monetary cost of transferring input data $\mu_{k_{mj}}$ and output data $\rho_{k_{mj}}$ of task $k_{mj} \in \mathcal{K}_m$ from mobile device $m \in \mathcal{M}$ to cloud provider $c \in \mathcal{C}$, which charges a per unit byte monetary cost p_c^{in} for inbound data and p_c^{out} for outbound data.
 - The second term represents the total monetary cost for using VM instances from different cloud providers. In this case, the task assignment decision variable $\phi_{k_{mj}v_{cir}} = 1$. It represents the monetary cost of using VM $v_{cir} \in V_{ci}$

of instance type $i \in \mathcal{I}$ from cloud provider $c \in \mathcal{C}$, for a number of slots $s_{k_{mj}}$ of unit time by task $k_{mj} \in \mathcal{K}_m$ from mobile device $m \in \mathcal{M}$.

$$\begin{aligned} \mathcal{E}(\Phi) = & \sum_{m \in \mathcal{M}} \sum_{k_{mj} \in \mathcal{K}_m} \sum_{c \in \mathcal{C}} \sum_{i \in \mathcal{I}} \sum_{v_{cir} \in \mathcal{V}_{ci}} e_{k_{mj}v_{cir}} \phi_{k_{mj}v_{cir}} \\ & + \sum_{m \in \mathcal{M}} \sum_{k_{mj} \in \mathcal{K}_m} \sum_{c \in \mathcal{C}} \sum_{i \in \mathcal{I}} \sum_{v_{cir} \in \mathcal{V}_{ci}} e_{k_{mj}} (1 - \phi_{k_{mj}v_{cir}}) \end{aligned} \quad (8.3)$$

$$\begin{aligned} \mathcal{P}(\Phi) = & \sum_{m \in \mathcal{M}} \sum_{k_{mj} \in \mathcal{K}_m} \sum_{c \in \mathcal{C}} \sum_{i \in \mathcal{I}} \sum_{v_{cir} \in \mathcal{V}_{ci}} (\mu_{k_{mj}} p_c^{in} + \rho_{k_{mj}} p_c^{out}) \phi_{k_{mj}v_{cir}} \\ & + \sum_{m \in \mathcal{M}} \sum_{k_{mj} \in \mathcal{K}_m} \sum_{c \in \mathcal{C}} \sum_{i \in \mathcal{I}} \sum_{v_{cir} \in \mathcal{V}_{ci}} p_{v_{cir}}^{res} s_{k_{mj}} \phi_{k_{mj}v_{cir}} \end{aligned} \quad (8.4)$$

8.2.4 The Model

The objective function of the Task Scheduling Problem for Resource Augmentation Environments for Mobile Cloud Computing (*TSPRAEMCC*) is to minimize the total energy consumption and the total monetary cost across all mobile devices. The objective function of the model is given as follows.

$$\min \quad [\mathcal{E}(\Phi) + \alpha \mathcal{P}(\Phi)] \quad (8.5)$$

In the model, a parameter α is introduced to give weightage to both cost functions in (8.5). It is defined as the ratio of the total energy consumption to the total monetary cost (8.6) when offloading using a centralized broker-node but without

optimization (i.e. the second scenario as explained later in Section 8.4), multiplied by weightage factor w_{ep} . The parameter α is defined as follows.

$$\alpha = w_{ep} \times \frac{\text{total energy consumption}}{\text{total monetary cost}}, \quad w_{ep} \geq 0. \quad (8.6)$$

The various values taken by α based on w_{ep} are explained as follows. (i) When $w_{ep} = 1$, then the value of α balances both cost functions in (8.5) and they are optimized equally. (ii) When $w_{ep} = 0$, then α is also zero, and only the total energy consumption is optimized. (iii) When $0 < w_{ep} < 1$, then the total energy consumption has more weightage than the total monetary cost in (8.5). (iv) When $w_{ep} > 1$, then the total energy consumption has less weightage than the total monetary cost in (8.5). The objective function exposed in (8.5) is subject to various constraints, which are defined below.

Data Rate Constraints

The delay tolerance ($\lambda_{k_{mj}}$) for an offloading task is a user-defined constraint. It puts a limit on the delay ($d_{k_{mj}v_{cir}}$) of the task, as given by the following inequation.

$$d_{k_{mj}v_{cir}} \leq \lambda_{k_{mj}} t_{k_{mj}}^{exec} \quad (8.7)$$

Intuitively, during data transfer, collisions occur at the lower layers of the net-

work and re-transmissions take place. Consequently, additional time due to re-transmissions is incurred in the transfer time of the data. However, in the evaluation settings, when estimating the transfer time for the data, re-transmissions are not considered. Therefore, it is assumed that the transfer time for data linearly depends on the size of the data and the available data rate. Let $b_{v_{cir}}$ be the available data rate between mobile devices in $\mathcal{M}$ and VM v_{cir} . Then, the time for the task to send $(t_{k_{mj}v_{cir}}^{send})$ input data $(\mu_{k_{mj}})$ can be expressed as $\frac{\mu_{k_{mj}}}{b_{v_{cir}}}$. Similarly, the time to receive $(t_{k_{mj}v_{cir}}^{rec})$ output data $(\rho_{k_{mj}})$ generated from the execution of the task at the VM can be expressed as $\frac{\rho_{k_{mj}}}{b_{v_{cir}}}$. After solving (8.1) and (8.7) for $b_{v_{cir}}$, the data rate constraints are defined by the following inequation.

$$b_{v_{cir}} \geq \frac{\mu_{k_{mj}} + \rho_{k_{mj}}}{\lambda_{k_{mj}} t_{k_{mj}}^{exec} - t_{k_{mj}v_{cir}}^{exec}}, \quad \forall m \in \mathcal{M}, \forall k_{mj} \in \mathcal{K}_m; \forall c \in \mathcal{C}, \forall i \in \mathcal{I}, \forall v_{cir} \in \mathcal{V}_{ci} \quad (8.8)$$

Multiple Assignment Constraints

These constraints ensure that each task is scheduled on only one VM and only one cloud provider. The multiple assignment constraints are given by the following inequation.

$$\sum_{c \in \mathcal{C}} \sum_{i \in \mathcal{I}} \sum_{v_{cir} \in \mathcal{V}_{ci}} \phi_{k_{mj}v_{cir}} \leq 1, \quad \phi \in \{0, 1\}, \forall m \in \mathcal{M}, \forall k_{mj} \in \mathcal{K}_m \quad (8.9)$$

Overloading Constraints

The overloading constraints are only applicable when the model is evaluated in RAE using a local private cloud. In this environment, it is assumed that there is only one cloud provider (i.e. $|\mathcal{C}| = 1$), and it provides only *small* instance types of VMs (i.e. $|\mathcal{I}| = 1$). It is also assumed that the private cloud has limited computing resources. Thus, the overloading constraints prevent executing too many tasks on the private cloud based on its CPU and memory resources. For example, when tasks are assigned to cloud provider c then the overloading constraints ensure that the total amount of the CPU power (8.10) and the memory capacity (8.11) required by all the tasks assigned to the cloud should be less than the cloud's CPU power (β_c) and memory capacity (γ_c) respectively. The overloading constraints are as follows.

$$\sum_{m \in \mathcal{M}} \sum_{k_{mj} \in \mathcal{K}_m} \phi_{k_{mj}v_{cir}} \beta_{k_{mj}v_{cir}} < \beta_c \quad \forall c \in \mathcal{C} \quad (8.10)$$

$$\sum_{m \in \mathcal{M}} \sum_{k_{mj} \in \mathcal{K}_m} \phi_{k_{mj}v_{cir}} \gamma_{k_{mj}v_{cir}} < \gamma_c \quad \forall c \in \mathcal{C} \quad (8.11)$$

8.3 Deriving Evaluation Settings

Generally, resource intensive tasks are considered for offloading. The tasks considered for the model evaluation are either CPU, memory or I/O resource intensive, or

a combination of these resources. For example, a task could be CPU and input data intensive, and another task could be only input and output data intensive. When executing a task locally on a mobile device, then the monetary cost is assumed to be zero. Thus, the type of resource intensiveness determines only the energy consumption. When executing remotely on cloud resources, the type of resource intensiveness determines the energy consumption and the monetary cost (if any). The *TSPRAEMCC* problem is solved with multiple data inputs, which are explained in the following subsections.

8.3.1 VM Instance Types & Monetary Costs

In a RAE using public clouds, three cloud providers (i.e. $|\mathcal{C}| = 3$) are considered, namely *c1*, *c2*, and *c3*. Each cloud provider has VMs of *small*, *large* and *xlarge* instance types (i.e. $|\mathcal{I}| = 3$). In the environment using a local private cloud, only one cloud provider (i.e. $|\mathcal{C}| = 1$) is considered, which provides VMs of only *small* instance type (i.e. $|\mathcal{I}| = 1$).

Table 8.1: Instance Types: Configuration & Speed-up Factor

Instance Type	# of CPUs	Memory (GB)	Speed-up Factor
<i>small</i>	1	1.7	4
<i>large</i>	4	7.5	6
<i>xlarge</i>	8	17	10

The configuration of VM instances available from the cloud providers is defined by the number of CPUs and the amount of memory, as summarized in Table 8.1. It is assumed that VM instance types provided by different cloud providers have the same configuration. Another parameter for VM instances is the speed-up factor. For a VM instance, this is given as how fast is the the computing capability of the instance with respect to mobile devices in the system. In the evaluation settings of the model, the speed-up factor for *small*, *large* and *xlarge* instances is set to 4, 6 and 10 respectively, as shown in Table 8.1. The environment using a local private cloud provides only *small* instance type. Therefore, the number of CPUs (β_c) and the amount of memory (γ_c) in the private cloud is set according to the configuration of a *small* VM instance type. In this case, the number of CPUs used by a task is one (i.e. $\beta_{k_{mj}v_{cir}} = 1$) and the amount of memory used is $1.7GB$ (i.e. $\gamma_{k_{mj}v_{cir}} = 1.7GB$). Thus, if 10 VMs are considered in the private cloud, then there are 10 CPUs and $17GB$ of memory.

Table 8.2: Monetary costs for different cloud resources

Cloud Providers	VM Instance Types			Network Traffic	
	<i>small</i> \$/10s	<i>large</i> \$/10s	<i>xlarge</i> \$/10s	Inbound \$/MB	Outbound \$/MB
c1	0.085	0.34	0.58	0.08	0.08
c2	0.07	0.12	0.24	0.2	0.2
c3	0.10	0.40	0.70	0.08	0.08

It is assumed that the monetary cost of using resources from the private cloud is zero. However, the public cloud providers associate a monetary cost: (i) for using a VM of a specific instance type, and (ii) for sending/receiving network traffic, as

summarized in Table 8.2. The monetary cost of using a VM instance type is set per 10s time slot. For example, if task k_{mj} uses an instance type for 35s, then the number of time slots per 10s ($s_{k_{mj}}$) for the task is 3.5, which in our case is rounded upto the next integer value, i.e. $s_{k_{mj}} = 4$. Thus, the monetary cost of using a small instance in $c1$ for 35s is $4 * 0.085$ dollars. The monetary cost for the network traffic is set per MB of data transfer.

8.3.2 Task Execution Times & I/O Data Sizes

Local Execution Time

The local execution times of the tasks across all mobile devices in $\mathcal{M}$ are set according to a Uniform distribution $\mathcal{U}(a, b)$ within the range $a = 0$ to $b = 100$ seconds.

Remote Execution Time

The remote execution time ($t_{k_{mj}v_{cir}}^{exec}$) of a task (k_{mj}) when it is executed on a VM (v_{cir}) is based on its local execution time ($t_{k_{mj}}^{exec}$) and the speed-up factor ($\mathcal{F}_{v_{cir}}$) of the VM with respect to the mobile device (see Table 8.2). It can be expressed by the following equation.

$$t_{k_{mj}v_{cir}}^{exec} = \frac{t_{k_{mj}}^{exec}}{\mathcal{F}_{v_{cir}}} \quad (8.12)$$

I/O Data Sizes

The input ($\mu_{k_{mj}}$) and output ($\rho_{k_{mj}}$) data sizes (in MB) of the tasks are set according to a Uniform distribution $\mathcal{U}(a, b)$ within the range $a = 0$ to $b \in \{10, 20, 30, 40\}$.

8.3.3 Delay Tolerance Parameter

The delay tolerance parameter for a task ($\lambda_{k_{mj}}$) is a user-defined parameter. It is a unit-less quantity. When a task is offloaded, the delay tolerance parameter of the task sets a limit on the delay of the task. From (8.1), (8.7) and (8.12), the delay tolerance can be given by the following inequation.

$$\lambda_{k_{mj}} \geq \frac{1}{\mathcal{F}_{v_{cir}}} + \frac{\mu_{k_{mj}}}{b_{v_{cir}} t_{k_{mj}}^{exec}} + \frac{\rho_{k_{mj}}}{b_{v_{cir}} t_{k_{mj}}^{exec}} \quad (8.13)$$

The minimum value of the delay tolerance for a task can be given when the input ($\mu_{k_{mj}}$) and output ($\rho_{k_{mj}}$) data sizes in (8.13) are negligible; then $\lambda_{k_{mj}} \geq \frac{1}{\mathcal{F}_{v_{cir}}}$. The delay tolerance parameters for the tasks across all mobile devices are set according to Uniform distribution $\mathcal{U}(a, b)$ within the range, $a = \frac{1}{\mathcal{F}_{v_{cir}}}$ to $b \in \{0.3, 0.5, 1.0, 1.5, 2.0\}$. For the lower range of the distribution, the speed-up factor of *small* VM instance type (i.e. when $\mathcal{F}_{v_{cir}} = 4$) is used. Thus, $a = 0.25$ such that the constraints in (8.8) could be satisfied with the lowest configuration VM instance type as well (i.e. *small* VM instance type).

8.3.4 Data Rate

In general, due to the slow TCP sliding window, the transfer of small amounts of data experiences slower data rates as compared with large data sizes [38]. In our case, the same data rate is set for all data size ranges. On the other hand, the connectivity of mobile devices to public clouds resources through the Internet could result in high communication latency, low network bandwidth and more energy consumption than the connectivity to a local cloud through a WiFi network [36], [69], [75], [95]. In other words, mobile devices experience higher data rates while accessing resources from a local cloud than from public clouds. Thus, the data rates (in KB/s) in the environment using a local private cloud are set according to a Normal distribution $\mathcal{N}(700, 200)$, and are set at $\mathcal{N}(500, 300)$ when using public clouds. The smaller value of the standard deviation for the traffic model when using a local cloud compared with the high value when using the Internet shows that there is less variation in data rate when using a WiFi network compared with using the Internet.

8.3.5 Energy Consumption

The power rating of each mobile device's (in set $|\mathcal{M}|$) WiFi radio and CPU in their respective states is given by Table 6.3.

Remote Energy Consumption

The remote energy consumption $e_{k_{mj}v_{cir}}$ includes the following parameters: $e_{k_{mj}v_{cir}}^{send}$, energy consumed when sending input data; $e_{k_{mj}v_{cir}}^{Idle}$, energy consumed when in the idle state; and $e_{k_{mj}v_{cir}}^{rec}$, energy consumed when receiving output data. The remote energy consumption can be expressed by the following equation.

$$e_{k_{mj}v_{cir}} = e_{k_{mj}v_{cir}}^{send} + e_{k_{mj}v_{cir}}^{Idle} + e_{k_{mj}v_{cir}}^{rec} \quad (8.14)$$

Energy consumption in the idle state of a mobile device (while waiting for the results) is negligible and can be ignored. Moreover, during this time the mobile device can do other tasks. Based on (6.10), an estimation of the energy consumption in a mobile device when sending or receiving data is given by the following equations.

$$e_{k_{mj}v_{cir}}^{send} = t_{k_{mj}v_{cir}}^{send} \times PowerRating_Send \quad (8.15)$$

$$e_{k_{mj}v_{cir}}^{rec} = t_{k_{mj}v_{cir}}^{rec} \times PowerRating_Receive \quad (8.16)$$

As mentioned in Subsection 8.2.4, when calculating transfer time ($t_{k_{mj}v_{cir}}^{send}$ or $t_{k_{mj}v_{cir}}^{rec}$) for a piece of data, the re-transmissions time is not considered. Therefore, energy consumption $e_{k_{mj}v_{cir}}^{send}$ (8.15) or $e_{k_{mj}v_{cir}}^{rec}$ (8.16) does not include the energy consumption during re-transmissions.

Local Energy Consumption

Based on (6.10), the estimation of the energy consumption when task k_{mj} is executed locally, is given by the following equation.

$$e_{k_{mj}} = t_{k_{mj}}^{exec} \times PowerRating_Compute \quad (8.17)$$

8.4 Results & Analysis

The *TSPRAEMCC* Integer Linear Program (ILP) is implemented using IBM's linear programming solver called CPLEX [1] with all the default parameters. The evaluation of the model is performed on a single server machine with Intel Xeon(R) E5420 @ 2.50GHz, quad core CPU and 8GB of RAM. The Linux OS distribution on the server is Ubuntu 12.04.2 LTS precise 64bit.

The task scheduler model is evaluated in two RAEs for MCC, shown in Figures 8.1(a) and 8.1(b). In RAE using a local private cloud, it is assumed that mobile users do not pay for using the resources. The only cost considered while offloading a task is the energy consumption; thus, the value of α is set to zero (8.5). Therefore, in this case, the model finds an optimal solution for the task scheduling problem to minimize the total energy consumption ($\mathcal{E}(\Phi)$), subject to constraints (8.8), (8.9), (8.10), and (8.11).

On the other hand, in the RAE using public clouds, it is assumed that mobile

users pay for using the resources. The costs considered while offloading a task are the energy consumption and the monetary cost; thus, the value of α in (8.5) is set based on different values of the weightage factor (w_{ep}) in (8.6) (see Section 8.2.4). In this case, the model finds an optimal solution for the task scheduling problem to minimize the total energy consumption ($\mathcal{E}(\Phi)$) and the total monetary cost ($\mathcal{P}(\Phi)$), subject to the constraints in (8.8) and (8.9). The overloading constraints (8.10) and (8.11) do not apply to this environment since it is assumed that public clouds have infinite resources.

The problem sizes for the task scheduling problems are generated based on: (i) the number of tasks across all the mobile devices, and (ii) the number of VMs across all cloud providers. It is assumed that, at a given time, each mobile device has 5 tasks (i.e. $|\mathcal{K}_m| = 5$) to offload/execute, and one VM executes one task. Therefore, if the number of mobile devices ($|\mathcal{M}|$) is varied by 5, 10, 20, 40, and 60, then the total number of tasks in the system will be 25, 50, 100, 200, and 300. The number of VMs in both RAEs is set differently since the availability of resources is different (finite vs. infinite). Thus, the problem sizes are different as well. The availability of resources from the private cloud is assumed finite; therefore, the number of VMs in some problem sizes should be less than the number of tasks. Therefore, the number of VMs ($|\mathcal{V}_{ci}|$) in this case is set to 25, 50, 100, and 150. On the other hand, the availability of resources from the public clouds is assumed infinite. To that end, the number of VMs is set to be greater than the number of tasks in the system such

that none of the tasks is denied offloading due to a lack of resources. Thus, as the number of tasks considered in the system is 25, 50, 100, 200, and 300, the number of VMs is varied by 40, 50, 60 per instance type and per cloud provider. In this case, the number of VMs across all cloud providers is always greater than the number of tasks across all mobile devices. For example, three cloud providers are considered for the public clouds (i.e. $|\mathcal{C}| = 3$), and there are three instance types (i.e. $|\mathcal{I}| = 3$) in each of the cloud providers. Therefore, if 40 VMs per instance type and per cloud provider are set, then the number of VMs across all cloud providers is 360, which is greater than the maximum number of tasks (300) considered in the system. In summary, the number of tasks in the problem sizes is varied from 25, 50, 100, 200, to 300 in both RAEs. For each number of tasks, the number of VMs is varied from: (i) 25, 50, 100, to 150 when using the private cloud, and (ii) 360, 450, to 540 when using public clouds.

The task scheduler model is evaluated by observing the total energy consumption and/or monetary cost across all mobile devices in the following three scenarios.

1. **Energy consumption without offloading:** The first scenario provides a baseline for the total energy consumption across all mobile devices ($|\mathcal{M}|$). In this case, the broker-node is not involved in scheduling task offloading, and therefore, all the tasks are processed locally on the mobile devices. In this case, $\phi_{k_{mj}v_{cir}} = 0$ for every task, and the total monetary cost is zero as well. The

total energy consumption is given by (8.3) when $\phi_{k_{mj}v_{cir}} = 0$. Thus, the cost function $\mathcal{E}(\Phi)$ is the sum of the local energy consumption ($e_{k_{mj}}$) by all the tasks across all the mobile devices, which is given by the following equation.

$$\mathcal{E}(\Phi) = \sum_{m \in \mathcal{M}} \sum_{k_{mj} \in \mathcal{K}_m} e_{k_{mj}} \quad (8.18)$$

2. Energy consumption and monetary cost using offloading without optimization:

In the second scenario, the task scheduler at the centralized broker-node schedules tasks but without optimization. In both RAEs, the values ($\phi_{k_{mj}v_{cir}}$) in the set Φ will be obtained by scheduling tasks one-by-one in sequence while satisfying constraints for respective RAEs. A given task will be offloaded to the first available VM, provided the task satisfies all the constraints. The total energy consumption and the total monetary cost across all mobile devices in this case are expressed by (8.3) and (8.4) respectively. This scenario provides a baseline for the total monetary cost across all mobile devices in the system.

3. Energy consumption and monetary cost using offloading with optimization:

In the third scenario, the task scheduler at the centralized broker-node schedules tasks with optimization (8.5). In both RAEs, the values ($\phi_{k_{mj}v_{cir}}$) in the set Φ will be obtained as a result of the optimization process with the aim to minimize the total energy and monetary cost across all the mobile devices

while satisfying the constraints in each RAE. The total energy consumption and the total monetary cost in this case are also given by (8.3) and (8.4) respectively.

The task scheduler is evaluated by observing the effect of task offloading with and without optimization on the total energy consumption and the total monetary cost in the second and the third scenarios. Further, in the third scenario, the performance of the task scheduler is evaluated by observing: (i) the effect of infinite and finite available resources, data size, and delay tolerance parameter of tasks on the total energy consumption and the total monetary cost, and (ii) how it selects cloud providers and their resources for data intensive and delay critical tasks while optimizing the costs. In all sections of the results, the total energy consumption and the total monetary cost across all mobile devices are obtained by calculating the average over 30 iterations.

8.4.1 Infinite & Finite Resources

The effect on the total energy consumption when offloading with optimization is observed when: (i) finite resources are available from the private cloud (Figure 8.2), and (ii) infinite resources are available from the public clouds. In both cases, a percentage saving in the total energy consumption is observed. It is computed as the total energy consumption when offloading tasks with optimization ($E_{\text{off_opt}}$) (i.e. the third scenario) compared with when there is no offloading ($E_{\text{no_off}}$) (i.e. the first scenario).

Figure 8.2: The effect of finite and infinite resources on the percentage saving in the total energy consumption when offloading with optimization.

$$\text{Percentage saving} = \frac{E_{\text{no.off}} - E_{\text{off.opt}}}{E_{\text{no.off}}} \times 100 \quad (8.19)$$

The input/output data sizes (in MB) and delay tolerances of the tasks are set according to a Uniform distribution $\mathcal{U}(0, 10)$ and $\mathcal{U}(0.25, 1.0)$ respectively.

As mentioned earlier, when discussing the private cloud environment, the number of VMs in some problem sizes is less than the number of tasks. Thus, some tasks are denied offloading (even if they satisfy other constraints), and these are executed locally on the mobile devices. In this case, an increase in percentage saving of the total energy consumption is observed (Figure 8.2(a)) when more VMs are provided in the system. This effect was expected since with the availability of more VMs, more

tasks can be offloaded. However, this is not the case when using public clouds, where the number of VMs in each problem size is higher than the number of tasks. Thus, in each problem size, the variation in percentage saving of the total energy consumption is very subtle when more VMs are provided in the system (Figure 8.2(b)).

It is obvious that, in the private cloud (Figure 8.2(a)), more saving in the total energy consumption can be achieved by providing more than 150 VMs. However, it is assumed that a finite number of VMs are available in this environment; thus, the number of VMs should be less than the number of tasks in some problem sizes. Thus, in the following sections, the number of VMs in this case is set to 150. On the other hand, saving in the total energy consumption is almost the same in the public clouds when the number of VMs are 360, 450 and 540. Thus, the number of VMs in this case is set to 450.

8.4.2 Offloading with & without Optimization

In this section, the effect of task offloading with and without optimization is observed on the total energy consumption in RAE using a local private cloud (Figure 8.3). The same effect is observed on the total energy consumption and the total monetary cost in RAE using public clouds (Figures 8.4(a) & (b)). In this case, the weightage factor $w_{ep} = 1$, and the value of α given by (8.6) balances the total energy consumption and the total monetary cost in (8.5). The input/output data sizes (in MB) and delay tolerances of the tasks are set according to a Uniform distribution $\mathcal{U}(0, 10)$ and

$\mathcal{U}(0.25, 1.0)$ respectively.

Figure 8.3: The total energy consumption when offloading with and without optimization in RAE using a local private cloud.

The total energy consumption in both RAEs, (Figures 8.3 and 8.4(a)) when there is no offloading (i.e. black line) is always higher than the total energy consumption in the other two scenarios. This was expected since this is the actual amount of energy consumed when tasks are processed locally. A significant reduction in the total energy consumption (i.e. green and red line) is observed when a centralized broker-node is added in the architecture to handle task scheduling on behalf of all mobile devices. Further, a difference in the total energy consumption is observed when tasks are offloaded without optimization (i.e. green line) and with optimization (i.e. red line).

Similarly, the total monetary cost (i.e. blue line, Figure 8.4(b)) across all mobile devices is observed in RAE using public clouds when tasks are offloaded without

Figure 8.4: The total energy consumption and the total monetary cost when offloading with and without optimization in RAE using public clouds.

optimization (i.e. the second scenario). This monetary cost is a baseline of the total monetary cost across all mobile devices. A reduction in the total monetary cost (i.e. red line) is observed when tasks are offloaded while optimizing energy consumption and monetary cost (i.e. the third scenario).

It is observed that the task scheduler model minimized the total energy consumption and the total monetary cost across all mobile devices by carefully offloading tasks. However, the improvement in the total energy consumption in RAE using public clouds is more than in RAE using a local private cloud. This observation suggests that, although the available data rate to a local cloud is higher than in the case of a public cloud; the combined effect of finite versus infinite and slower verses

faster resources yields higher improvement in the total energy consumption when using public clouds. However, such improvement comes at a monetary cost. Therefore, when making a decision between a local and a public cloud for task offloading, there will be a trade-off between the energy, computation, and monetary cost.

8.4.3 Input & Output Data Sizes

When a task is executed locally on a mobile device, data transfer from the mobile device to an outside computing resources does not take place. However, during remote execution, task offloading to a remote location incurs additional communication cost when transferring the task and its related data. The data transfer accounts for the energy consumption (8.14), and the monetary cost if offloading onto public clouds. In this section, the effect of input/output data sizes of the offloading tasks is observed on the total energy consumption. The input ($\mu_{k_{mj}}$) and output ($\rho_{k_{mj}}$) data sizes (in MB) are set according to a Uniform distribution $\mathcal{U}(0, b)$, $b \in \{10, 20, 30, 40\}$. The delay tolerances are set according to a Uniform distribution $\mathcal{U}(0.25, 1.0)$. The results in Figures 8.5(a) & (b) show percentage saving in the total energy consumption (as defined by (8.19)) at different data sizes, in both RAEs.

The percentage saving in the total energy consumption is high in both RAEs when data sizes are small (i.e. red line) compared with when data sizes are large (i.e. black line). It is observed in Section 8.4.2 that task offloading reduces energy consumption in the mobile device by saving on computational energy. However, results in Figures

Figure 8.5: Percentage saving in the total energy consumption when offloading with optimization at different data sizes, in both RAEs.

8.5(a) & (b) suggest that, while task offloading, the data transfer incurs additional energy consumption which does not happen when the task is executed locally on the mobile device. Also, the energy consumption due to the data transfer could be more when executing locally on the mobile device. Thus, if energy saving is the goal of task offloading then offloading may not be beneficial for data intensive tasks. Also, offloading data intensive tasks to the public clouds may not be beneficial as compared with the local cloud since the available data rate to the public cloud is lower than the local cloud.

The task scheduler is also evaluated to see how it selects among various public cloud providers based on the monetary costs of various resources provided by them.

In this case, weightage factor $w_{ep} = 1$, and the value of α given by (8.6), balances the

total energy consumption and the total monetary cost in (8.5). In this evaluation, first, only the monetary cost of VM instances is considered, as given in Table 8.2. The monetary cost of network resources is taken as zero. Thus, with respect to the monetary cost of VM instances, cloud provider $c3$ is the most expensive and $c2$ is the least expensive. When scheduling task offloading, the task scheduler carefully selects cloud providers with the aim to minimize the total energy consumption and the total monetary cost. Therefore, more tasks are offloaded onto the resources of the least expensive cloud provider $c2$ (i.e. red bar) compared with the most expensive cloud provider $c3$ (i.e. green bar), as shown in Figure 8.6(a). Second, the monetary cost of network resources is introduced along with the monetary cost of VM instances, as given in Table 8.2. Now, with respect to the network costs, cloud provider $c3$ is the least expensive, while $c2$ is the most expensive. Thus, more tasks are offloaded to cloud providers $c1$ and $c3$ (i.e. blue and green bars) compared with $c2$ (i.e. red bar), as shown in Figure 8.6(b). This shows that the task scheduler when offloading with optimization, minimizes the total monetary cost as well as the total energy consumption. Therefore, it pushes the tasks towards the clouds that offer less expensive resources.

8.4.4 Delay Tolerance

The delay tolerance ($\lambda_{k_{mj}}$) of a task is a user-defined parameter, which sets an acceptable delay for the task when executed at a remote location. Thus, unlike data

Figure 8.6: Selecting different cloud providers for data intensive tasks based on monetary costs in RAE using public clouds.

sizes, the value of the delay tolerance of a task does not affect the energy consumed by the task (8.14), rather, it only, affects the data rate constraints in (8.8). In this section, the effect of delay tolerances of the tasks at different data sizes is observed on the total energy consumption when offloading with optimization (i.e. the third scenario).

The delay tolerances are set according to a Uniform distribution $\mathcal{U}(0.25, b)$, $b \in \{0.3, 0.5, 1.0, 1.5, 2.0\}$. When $\lambda_{k_{mj}}$ for an offloading task is in the range $0.25 < \lambda_{k_{mj}} \leq 1$, then it is a critical condition for the task. Under this condition, the additional time incurred due to data transfer of the task must be compensated by the computation time saved through remote execution such that the remote completion time (8.1) of the task is less than the local execution time (8.7). On the other hand, when $\lambda_{k_{mj}} > 1$, then it is a relaxed condition for the offloaded task. In this situation, a remote completion time greater than the local execution time can be tolerated. When

Figure 8.7: The effect of delay tolerance on the total energy consumption when data sizes (in MB) are in a range $\mathcal{U}(0, 40)$.

$\lambda_{k_{mj}} = \mathcal{U}(0.25, 2.0)$, then it is the most relaxed condition for the task, compared with the other distributions considered for delay tolerances in the model. The results show the total energy consumption, which is the difference between the total energy consumption when: (i) $\lambda_{k_{mj}} = \mathcal{U}(0.25, b)$, and $b \in \{0.3, 0.5, 1.0, 1.5\}$, and (ii) $\lambda_{k_{mj}} = \mathcal{U}(0.25, 2.0)$. The data sizes (in MB) are set according to a Uniform distribution $\mathcal{U}(0, b)$, $b \in \{10, 20, 30, 40\}$. It is observed that the effect of $\lambda_{k_{mj}}$ at different data sizes is similar to the total energy consumption; thus, results are shown only for the case when data sizes are in a range $\mathcal{U}(0, 40)$ (Figures 8.7(a) & 8.7(b)).

The total energy consumption (i.e. red line) is high when $\lambda_{k_{mj}}$ is small (i.e. $\lambda_{k_{mj}} = 0.3$) compared with the total energy consumption (i.e. green line) when $\lambda_{k_{mj}}$

(a) Total energy consumption.

(b) Total monetary cost.

(c) Number of tasks offloaded.

Figure 8.8: The effect on the total energy consumption, the total monetary cost, and the number of offloaded tasks, when only *small* or *small*, *large*, and *xlarge* VM instances are available in RAE using public clouds.

is large (i.e. $\lambda_{k_{mj}} = 1.5$). The reason for this is that, at the small values of $\lambda_{k_{mj}}$ (i.e. in the range $0.25 < \lambda_{k_{mj}} < 1$), a delay (or remote completion time) shorter than the local execution time is required. Consequently, fewer tasks may satisfy the constraints in (8.8) since, to achieve the desired short delay, fast computing resources and high data rates are required from clouds. Therefore, more tasks are executed locally on mobile devices, and the total energy consumption is high.

Further, the task scheduler is evaluated to observe how it selects VMs from the public clouds based on their speed-up factors, as shown in Figure 8.8. As mentioned earlier, the available VM instance types from the public clouds are *small*, *large* and *xlarge*. The speed-up factors of the VM instances with respect to the mobile devices are 4, 6 and 10 respectively. The weightage factor $w_{ep} = 1$, and the value of α given by (8.6) balances the total energy consumption and the total monetary cost (8.5). To evaluate this performance, tasks with large data sizes (i.e. in a range $\mathcal{U}(0, 40)$) and small delay tolerances (i.e. in a range $\mathcal{U}(0.25, 0.3)$) are considered. These tasks are very demanding - the remote completion time should be shorter than the local execution. Therefore, fast VMs and high data rates are required from the clouds such that the constraints of (8.8) are satisfied. Figure 8.8 shows that when only slow VMs (i.e. *small* VM instance types) are available for these demanding tasks, then most of the tasks do not satisfy the data rate constraints of (8.8). Therefore, the tasks are executed locally on mobile devices.

The energy consumption by the locally executed tasks is high (i.e. red line, Figure

8.8(a)), and the monetary cost is zero. Thus, the total monetary cost in this case is low (i.e. red line, Figure 8.8(b)). A reduction in the total energy consumption (i.e. blue line Figure 8.8(a)) is observed when fast VMs, (i.e. *large* and *xlarge* VMs) are introduced along with the already present *small* VMs in the system. When fast VMs are available, more tasks can satisfy the constraints in (8.8) compared with the situation when only slow VMs are available. Thus, more tasks can be offloaded onto clouds. The energy consumption due to the offloaded tasks is low and the monetary cost is high (i.e. blue line, Figure 8.8(b)). Figure 8.8(c) illustrates this effect when the number of tasks is 100, 200 and 300. The results show that the number of offloaded tasks is smaller when only *small* instance types are available (i.e. blue bar) compared with when *small*, *large* and *xlarge* (i.e. red bar) instance types are available.

In summary, it is the combined effect of the delay tolerance and the data size of an offloading task that will determine the required data rate (8.8). For example, in case of real-time time-critical data intensive tasks, the required delay is small and data sizes are large. These tasks are so demanding that, only a few tasks may satisfy the constraints in (8.8); thus, fast VMs and high data rates should be available to achieve small delays.

8.5 Summary of Results & Discussions

The proposed task scheduler decides on task offloading with the aim to minimize the total energy consumption and the total monetary cost. In both RAEs, an overall improvement in energy consumption and/or monetary cost is observed when offloading using a centralized broker-node with optimization compared with offloading without optimization.

Higher data rates are available while accessing resources from a local cloud than from public clouds. Thus, the energy consumption is high when offloading data intensive tasks onto public clouds compared with when offloading onto a local cloud. Therefore, offloading data intensive tasks may not be beneficial when using public clouds. The availability of higher data rates and lower network latency when using a local private cloud suggests that this environment is good for offloading real-time critical applications. However, the downside of a local private cloud is that it may have limited resources. Overall, an improvement in energy saving when offloading to public clouds is more compared with offloading to a local private cloud. The difference in the overall improvement can be attributed to the effect of the availability of infinite and faster computation resources in the public clouds versus finite and slower computation resources in the local private cloud.

In general, task offloading is beneficial when the cost of offloading is less than the cost of executing the task on the mobile device. The cost of task offloading may

include energy consumption, monetary cost, completion time of the task, etc. It is observed that the user-defined delay tolerance for an offloading task puts constraints on the remote completion time of the task. The data size of a task accounts for the energy consumption, monetary cost and the completion time of the task. When offloading a data intensive task, it may consume more energy than executing the task locally on the mobile device. On the other hand, offloading a task with small data size but with small delay tolerance may not be beneficial as well. Therefore, it is the combined effect of the delay tolerance and the data size of a task that influences task offloading decisions. Thus, every task may not be benefited from offloading; rather, it is a trade-off between the computation cost and the communication cost of the offloading task.

8.6 Conclusion

In this chapter, a centralized broker-node based architecture was utilized to handle task scheduling on behalf of a large number of mobile devices. A general mathematical model for the centralized task scheduling problem was proposed with an aim to minimize the total energy consumption and the total monetary cost across all mobile devices of the system. The model was evaluated in two RAEs for MCC, one using a local private cloud and the other using public clouds. The task scheduler model provided an optimal solution for the task scheduling problem (task assignment) and

minimized the total energy consumption when evaluated on the local private cloud environment, and the total energy consumption and the total monetary cost when evaluated on public cloud environments, subject to various constraints. The results showed that the total energy consumption and the total monetary cost across all mobile devices when offloading with optimization is less than when offloading without optimization using the centralized task scheduler.

Chapter 9

Conclusion & Future Work

In this thesis, the motivation was to consider a resource augmentation environment for a large number of mobile devices and multiple service nodes. More precisely, this research work was focused on the scalability of a resource augmentation environment.

In this research work, the challenges of task offloading requirements by a large number of mobile devices were presented. The placement of wireless access points and service nodes at strategic points in an area was proposed such that the congestion due to the presence of a large number of users could be avoided. The approach used to find strategic points was to map the area with current information on the density distribution of the users. A scanning algorithm was proposed to get current information on density distribution of the users in the area. The novelty of this algorithm is that using a WiFi network and cyber foraging the users' density distribution can be found in cases in which there is no communication network present, i.e. in any

unprepared (random) area. More precisely, the proposed approach did not rely on any pre-installed infrastructure of APs (or already prepared databases) in the subject area.

A centralized broker-node architecture was proposed to investigate the overhead of repeated resource monitoring by a large number of mobile devices. The results showed that, due to repeated resource monitoring, the communication traffic between the mobile devices and the multiple service nodes caused congestion in the WiFi network. The congestion in the network further increased the resource monitoring time for the mobile devices and decreased the scalability of the system. This situation caused delay for the mobile devices that were trying to get the up-to-date status of resource availability through the congested wireless network. The centralized broker-node architecture was utilized for managing resource monitoring on behalf of all mobile devices in the system. This approach helped in lowering the communication overhead that could have manifested due to resource monitoring by a large number mobile devices. Having less communication overhead, in turn, lowered the resource monitoring time for the mobile devices.

Further, the aim was to utilize the proposed centralized broker-node approach to handle task scheduling on behalf of all mobile devices in a resource augmentation environment. A mathematical model was proposed for the task scheduling problems using the centralized architecture for the local resources case and the mobile cloud computing case. The computing resources available in the local network (e.g. sur-

rogate nodes) were limited compared to with the unlimited resources from public clouds (e.g. VMs). The task scheduler models optimally solved the centralized task scheduling problems (task assignments) and minimized the total energy consumption across all mobile devices. The results showed that a significant reduction in the total energy consumption was achieved compared with the total energy consumption when tasks were offloaded from the centralized scheduler without optimization.

The previously proposed task scheduler models were extended and a generalized task scheduler model was proposed. The model was evaluated in two resource augmentation environments for mobile cloud computing. In the first environment, service nodes were available from a local private cloud and, in the second environment service nodes were available from public clouds. In both environments, an improvement in the total energy consumption and/or the total monetary cost was observed when offloading with optimization, compared to when offloading without optimization using the generalized task scheduler model. The available data rate when accessing resources from a local private cloud is higher than with public clouds. However, for given tasks, an improvement in the total energy consumption is greater when using resources from public clouds. This is the effect of unlimited and faster computing resources in public clouds as compared with limited and slower computing resources in a local private cloud. On the other hand, due to the availability of low data rates when accessing public clouds, offloading data intensive tasks may consume more energy than executing them locally on mobile devices. Apart from this, offloading may not

be beneficial for a task with small data sizes and shorter remote completion time than the local execution time of the task. Thus every task may not benefit from offloading; rather, it is a trade-off between the computation cost and the communication cost of the offloading task.

This research showed that, using the proposed centralized broker-node approach for a large resource augmentation environment: (i) the wireless network was alleviated from the communication overhead due to repeated resource monitoring, (ii) the mobile devices experienced less delay time when seeking resource descriptions from multiple service nodes, and (iii) the models found optimal solutions for the centralized task scheduling problems, and a significant reduction in the total costs was observed when offloading with optimization compared to when offloading without optimization using the centralized task scheduler.

Future Work: When managing resource monitoring at the centralized broker-node the results are based on simulations. In actual implementations there could be other impairments or factors affecting the performance. Therefore, further tests in actual networks are needed before deployment to see if the same performance obtained in the simulations will be obtained in actual networks. Another future work may include extending the scheduler model to consider network congestion, task priority and future mobile networks while scheduling task offloading.

Bibliography

- [1] Cplex: IBM's Linear Programming Solver. <http://www.ilog.com/product/cplex/>.
- [2] Google Apps. <https://google.com/apps/>.
- [3] HowTo Use LXC. http://www.nsnam.org/wiki/index.php/HOWTO_Use_Linux_Containers_to_set_up_virtual_networks.
- [4] Jini. <http://www.jini.org/>.
- [5] Mobile Cloud Computing \$9.5 Billion by 2014. http://www.juniperresearch.com/reports/mobile_cloud_applications_and_services.
- [6] Network Simulator ns-2. <http://www.isi.edu/nsnam/ns/>.
- [7] Network Simulator ns-3. <http://www.nsnam.org/documentation/>.
- [8] Salutation. <http://www.salutation.org/>.
- [9] Ubuntu One. <https://one.ubuntu.com/>.

- [10] UPnP, Universal Plug and Play forum. <http://www.upnp.org/>.
- [11] David Abramson, Rajkumar Buyya, and Jonathan Giddy. A Computational Economy for Grid Computing and its Implementation in the Nimrod-G Resource Broker. *Future Generation Computer Systems*, 18(8):1061–1074, 2002.
- [12] Keith Adams and Ole Agesen. A Comparison of Software and Hardware Techniques for x86 Virtualization. *ACM Sigplan Notices*, 41(11):2–13, 2006.
- [13] Khalil Amiri, David Petrou, Gregory R Ganger, and Garth A Gibson. Dynamic Function Placement for Data-Intensive Cluster Computing. In *Proceedings of the USENIX Annual Technical Conference, General Track*, pages 307–322, 2000.
- [14] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R.H. Katz, A. Konwinski, G. Lee, D.A. Patterson, A. Rabkin, I. Stoica, et al. Above the Clouds: A Berkeley View of Cloud Computing. *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-28*, 2009.
- [15] Elarbi Badidi and Ikbale Taleb. Towards a Cloud-Based Framework for Context Management. In *International Conference on Innovations in Information Technology (IIT)*, pages 35–40. IEEE, 2011.
- [16] G. Bai and C. Williamson. Simulation Evaluation of Wireless Web Performance in an IEEE 802.11b Classroom Area Network. In *Proceedings of 28th Annual IEEE International Conference on Local Computer Networks, (LCN)*, pages 663–672. IEEE, 2003.

- [17] A. Balachandran, G.M. Voelker, P. Bahl, and P.V. Rangan. Characterizing User Behavior and Network Performance in a Public Wireless LAN. In *ACM SIGMETRICS Performance Evaluation Review*, volume 30, pages 195–205. ACM, 2002.
- [18] Rajesh Balan, Jason Flinn, Mahadev Satyanarayanan, Shafeeq Sinnamohideen, and Hen-I Yang. The Case for Cyber Foraging. In *Proceedings of the 10th Workshop on ACM SIGOPS European Workshop*, pages 87–92. ACM, 2002.
- [19] Rajesh Krishna Balan. Powerful Change Part 2: Reducing the Power Demands of Mobile Devices. *Pervasive Computing, IEEE*, 3(2):71–73, 2004.
- [20] Rajesh Krishna Balan, Mahadev Satyanarayanan, So Young Park, and Tadashi Okoshi. Tactics-Based Remote Execution for Mobile Computing. In *Proceedings of the 1st International Conference on Mobile Systems, Applications and Services*, pages 273–286. ACM, 2003.
- [21] R.K. Balan, D. Gergle, M. Satyanarayanan, and J. Herbsleb. Simplifying Cyber Foraging for Mobile Devices. In *Proceedings of the 5th International Conference on Mobile Systems, Applications and Services*, pages 272–285. ACM, 2007.
- [22] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani. Energy Consumption in MobilePhones: A Measurement Study and Implications for Network Applications. In *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement*, pages 280–293. ACM, 2009.

- [23] M. Balazinska and P. Castro. Characterizing Mobility and Network Usage in a Corporate Wireless Local-Area Network. In *Proceedings of the 1st International Conference on Mobile Systems, Applications and Services*, pages 303–316. ACM, 2003.
- [24] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the Art of Virtualization. *ACM SIGOPS Operating Systems Review*, 37(5):164–177, 2003.
- [25] J. Basney and M. Livny. Improving Goodput by Co-scheduling CPU and Network Capacity. *International Journal of High Performance Computing Applications*, 13(3):220, 1999.
- [26] Y Mohamadi Begum and MA Maluk Mohamed. A DHT-Based Process Migration Policy for Mobile Clusters. In *Seventh International Conference on Information Technology: New Generations (ITNG)*, pages 934–938. IEEE, 2010.
- [27] Ruben Bossche, Kurt Vanmechelen, and Jan Broeckhove. Cost-Efficient Scheduling Heuristics for Deadline Constrained Workloads on Hybrid Clouds. In *Proceedings of the 3rd International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 320–327. IEEE, 2011.
- [28] Rajkumar Buyya, James Broberg, and Andrzej M Goscinski. *Cloud Computing: Principles and Paradigms*, volume 87. John Wiley & Sons, 2010.

- [29] Rajkumar Buyya, Rajiv Ranjan, and Rodrigo N Calheiros. Intercloud: Utility-Oriented Federation of Cloud Computing Environments for Scaling of Application Services. In *Algorithms and Architectures for Parallel Processing*, pages 13–31. Springer, 2010.
- [30] Rajkumar Buyya, Chee Shin Yeo, and Srikumar Venugopal. Market-Oriented Cloud Computing: Vision, Hype, and Reality for Delivering it Services as Computing Utilities. In *10th IEEE International Conference on High Performance Computing and Communications, (HPCC'08)*, pages 5–13. IEEE, 2008.
- [31] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic. 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, 2009.
- [32] Rodrigo N Calheiros, Rajiv Ranjan, Anton Beloglazov, César AF De Rose, and Rajkumar Buyya. 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, 2011.
- [33] B.G. Chun, S. Ihm, P. Maniatis, M. Naik, S. Ananth Narayan, S. Sharangi, A. Fedorova, A. Fattori, R. Pileari, L. Martignoni, et al. CloneCloud: Boosting Mobile Device Applications Through Cloud Clone Execution. *Arxiv preprint arXiv:1009.3088*, 2010.

- [34] Byung-Gon Chun and Petros Maniatis. Augmented Smartphone Applications Through Clone Cloud Execution. In *Proceedings of the 12th Workshop Hot Topics in Operating Systems (HotOS)*, volume 9, pages 8–11, 2009.
- [35] Byung-Gon Chun and Petros Maniatis. Dynamically Partitioning Applications between Weak Devices and Clouds. In *Proceedings of the 1st ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond*, page 7. ACM, 2010.
- [36] E. Cuervo, A. Balasubramanian, D. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl. Maui: Making Smartphones Last Longer with Code Offload. In *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*, pages 49–62. ACM, 2010.
- [37] K. Curran, E. Furey, T. Lunney, J. Santos, D. Woods, and A. McCaughey. An Evaluation of Indoor Location Determination Technologies. *Journal of Location Based Services*, 5(2):61–78, 2011.
- [38] M Darø Kristensen. Empowering Mobile Devices Through Cyber Foraging. In *Ph.D. Thesis*, 2010.
- [39] Gabriel Deak, Kevin Curran, and Joan Condell. A Survey of Active and Passive Indoor Localisation Systems. *Computer Communications*, 35(16):1939–1954, 2012.

- [40] Hoang T Dinh, Chonho Lee, Dusit Niyato, and Ping Wang. A Survey of Mobile Cloud Computing: Architecture, Applications, and Approaches. *Wireless Communications and Mobile Computing*, 13(18):1587–1611, 2013.
- [41] Jason Flinn, Dushyanth Narayanan, and Mahadev Satyanarayanan. Self-Tuned Remote Execution for Pervasive Computing. In *Proceedings of the Eighth Workshop on Hot Topics in Operating Systems*, pages 61–66. IEEE, 2001.
- [42] Jason Flinn, SoYoung Park, and Mahadev Satyanarayanan. Balancing Performance, Energy, and Quality in Pervasive Computing. In *Proceedings of the 22nd International Conference on Distributed Computing Systems*, pages 217–226. IEEE, 2002.
- [43] S. Goyal and J. Carter. A Lightweight Secure Cyber Foraging Infrastructure for Resource-Constrained Devices. In *Proceedings of the Sixth Workshop on Mobile Computing Systems and Applications, (WMCSA)*, pages 186–195. IEEE, 2005.
- [44] X. Gu, K. Nahrstedt, A. Messer, I. Greenberg, and D. Milojevic. Adaptive Offloading for Pervasive Computing. *Pervasive Computing*, 3(3):66–73, 2004.
- [45] Xiaohui Gu, Klara Nahrstedt, Alan Messer, Ira Greenberg, and Dejan Milojevic. Adaptive Offloading Inference for Delivering Applications in Pervasive Computing Environments. In *Proceedings of the 1st International Conference on Pervasive Computing and Communications (PerCom)*, pages 107–114. IEEE, 2003.

- [46] Lizheng Guo, Shuguang Zhao, Shigen Shen, and Changyuan Jiang. Task Scheduling Optimization in Cloud Computing based on Heuristic Algorithm. *Journal of Networks*, 7(3):547–553, 2012.
- [47] Yao Guo, Lin Zhang, Junjun Kong, Jian Sun, Tao Feng, and Xiangqun Chen. Jupiter: Transparent Augmentation of Smartphone Capabilities through Cloud Computing. In *Proceedings of the 3rd ACM SOSP Workshop on Networking, Systems, and Applications on Mobile Handhelds*, page 6. ACM, 2011.
- [48] Selim Gurun, Chandra Krintz, and Rich Wolski. NWSLite: A Light-Weight Prediction Utility for Mobile Devices. In *Proceedings of the 2nd International Conference on Mobile Systems, Applications, and Services*, pages 2–11. ACM, 2004.
- [49] Karen Henriksen, Jadwiga Indulska, and Andry Rakotonirainy. Infrastructure for Pervasive Computing: Challenges. In *GI Jahrestagung (1)*, pages 214–222. Citeseer, 2001.
- [50] Dijiang Huang, Xinwen Zhang, Myong Kang, and Jim Luo. MobiCloud: Building Secure Cloud Framework for Mobile Computing and Communication. In *5th International Symposium on Service Oriented System Engineering (SOSE)*, pages 27–34. IEEE, 2010.
- [51] Gonzalo Huerta-Canepa and Dongman Lee. An Adaptable Application Offloading Scheme based on Application Behavior. In *22nd International Conference*

- on Advanced Information Networking and Applications-Workshops, (AINAW)*, pages 387–392. IEEE, 2008.
- [52] Gonzalo Huerta-Canepa and Dongman Lee. A Virtual Cloud Computing Provider for Mobile Devices. In *Proceedings of the 1st ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond*, page 6. ACM, 2010.
- [53] G.C. Hunt and M.L. Scott. The Coign Automatic Distributed Partitioning System. *Operating Systems Review*, 33:187–200, 1998.
- [54] Anthony D Joseph, Alan F de Lespinasse, Joshua A Tauber, David K Gifford, and M Frans Kaashoek. *Rover: A Toolkit for Mobile Information Access*, volume 29. ACM, 1995.
- [55] S. Kafaie, O. Kashefi, and M. Sharifi. A Low-Energy Fast Cyber Foraging Mechanism for Mobile Devices. In *Arxiv Pre-print, arXiv:1111.4499*, 2011.
- [56] Swaroop Kalasapur and Mohan Kumar. Resource Adaptive Hierarchical Organization in Pervasive Environments. In *First International Communication Systems and Networks and Workshops (COMSNETS)*, pages 1–8. IEEE, 2009.
- [57] R. Kemp, N. Palmer, T. Kielmann, and H. Bal. The Smartphone and the Cloud: Power to the User. In *International Workshop on Mobile Computing and Clouds (MobiCloud)*, 2010.

- [58] R. Kemp, N. Palmer, T. Kielmann, F. Seinstra, N. Drost, J. Maassen, and H. Bal. eyeDentify: Multimedia Cyber Foraging from a Smartphone. In *Proceedings of the 11th International Symposium on Multimedia*, pages 392–399. IEEE, 2009.
- [59] Roelof Kemp, Nicholas Palmer, Thilo Kielmann, and Henri Bal. Cuckoo: A Computation Offloading Framework for Smartphones. In *Mobile Computing, Applications, and Services*, pages 59–79. Springer, 2012.
- [60] Roelof Kemp, Nicholas Palmer, Thilo Kielmann, and Henri Bal. The Smartphone and the Cloud: Power to the User. In *International Workshop on Mobile Computing, Applications, and Services*, pages 342–348. Springer, 2012.
- [61] Sokol Kosta, Andrius Aucinas, Pan Hui, Richard Mortier, and Xinwen Zhang. Thinkair: Dynamic Resource Allocation and Parallel Execution in the Cloud for Mobile Code Offloading. In *Proceedings of the 31st Annual International Conference on Computer Communications (INFOCOM)*, pages 945–953. IEEE, 2012.
- [62] D. Kotz and K. Essien. Analysis of a Campus-Wide Wireless Network. *Wireless Networks*, 11(1-2):115–133, 2005.
- [63] David Kotz, Robert Gray, Saurab Nog, Daniela Rus, Sumit Chawla, and George Cybenko. Agent TCL: Targeting the Needs of Mobile Computers. *Internet Computing*, 1(4):58–67, 1997.

- [64] Mads Darø Kristensen. Enabling Cyber Foraging for Mobile Devices. In *Proceedings of the 5th MiNEMA Workshop: Middleware for Network Eccentric and Mobile Applications*, pages 32–36. Citeseer, 2007.
- [65] Mads Darø Kristensen and Niels Olof Bouvin. Developing Cyber Foraging Applications for Portable Devices. In *2nd IEEE International Interdisciplinary Conference on Portable Information Devices, and the 7th IEEE Conference on Polymers and Adhesives in Microelectronics and Photonics*, pages 1–6. IEEE, 2008.
- [66] M.D. Kristensen. Execution Plans for Cyber Foraging. In *Proceedings of the 1st Workshop on Mobile Middleware: Embracing the Personal Communication Device*, page 2. ACM, 2008.
- [67] M.D. Kristensen. Scavenger: Transparent Development of Efficient Cyber Foraging Applications. In *Proceedings of the International Conference on Pervasive Computing and Communications (PerCom)*, pages 217–226. IEEE, 2010.
- [68] M.D. Kristensen and N.O. Bouvin. Scheduling and Development Support in the Scavenger Cyber Foraging System. In *Pervasive and Mobile Computing*, volume 6, pages 677–692. Elsevier, 2010.
- [69] Karthik Kumar and Yung-Hsiang Lu. Cloud Computing for Mobile Users: Can Offloading Computation Save Energy? *Computer*, 43(4):51–56, 2010.

- [70] Eemil Lagerspetz and Sasu Tarkoma. Mobile Search and the Cloud: The Benefits of Offloading. In *International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*, pages 117–122. IEEE, 2011.
- [71] Haofan Liang and Hanan Lutfiyya. A Cyberforaging Infrastructure Based on Web Services. In *Third International Conference on Autonomic and Autonomous Systems, (ICAS07)*, pages 59–59. IEEE, 2007.
- [72] H. Liu, H. Darabi, P. Banerjee, and J. Liu. Survey of Wireless Indoor Positioning Techniques and Systems. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 37(6):1067–1080, 2007.
- [73] P. Mell and T. Grance. The NIST Definition of Cloud Computing. *National Institute of Standards and Technology*, 53(6), 2009.
- [74] Alan Messer, Ira Greenberg, Philippe Bernadat, Dejan Milojevic, Deqing Chen, Thomas J Giuli, and Xiaohui Gu. Towards a Distributed Platform for Resource-Constrained Devices. In *22nd International Conference on Distributed Computing Systems*, pages 43–51. IEEE, 2002.
- [75] A.P. Miettinen and J.K. Nurminen. Energy Efficiency of Mobile Clients in Cloud Computing. In *Proceedings of the 2nd Conference on Hot Topics in Cloud Computing*, page 4. USENIX Association, 2010.

- [76] Alin Florindor Murarasu and Thomas Magedanz. Mobile Middleware Solution for Automatic Reconfiguration of Applications. In *Sixth International Conference on Information Technology: New Generations (ITNG'09)*, pages 1049–1055. IEEE, 2009.
- [77] Gil Neiger, Amy Santoni, Felix Leung, Dion Rodgers, and Rich Uhlig. Intel Virtualization Technology: Hardware Support for Efficient Processor Virtualization. *Intel Technology Journal*, 10(3), 2006.
- [78] Manjinder Nir and Ashraf Matrawy. Centralized Management of Scalable Cyber Foraging Systems. In *Proceedings of the 4th International Conference on Emerging Ubiquitous Systems and Pervasive Networks (EUSPN)*, volume 21, pages 265–273. Elsevier, 2013.
- [79] Manjinder Nir, Ashraf Matrawy, and Marc St-Hilaire. An Energy Optimizing Scheduler for Mobile Cloud Computing Environments. In *Proceedings of the 33rd Annual International Conference on Computer Communications - Workshop on Mobile Cloud Computing (INFOCOM WKSHPS)*, pages 404–409. IEEE, 2014.
- [80] Manjinder Nir, Ashraf Matrawy, and Marc St-Hilaire. Optimizing Energy Consumption in Broker-Assisted Cyber Foraging Systems. In *28th International Conference on Advanced Information Networking and Applications (AINA)*, pages 576–583. IEEE, 2014.

- [81] B.D. Noble, M. Satyanarayanan, D. Narayanan, J.E. Tilton, J. Flinn, and K.R. Walker. Agile Application-Aware Adaptation for Mobility. In *ACM SIGOPS Operating Systems Review*, volume 31, pages 276–287. ACM, 1997.
- [82] Daniel Nurmi, Richard Wolski, Chris Grzegorzczak, Graziano Obertelli, Sunil Soman, Lamia Youseff, and Dmitrii Zagorodnov. The Eucalyptus Open-Source Cloud-Computing System. In *9th International Symposium on Cluster Computing and the Grid (CCGRID'09)*, pages 124–131. IEEE, 2009.
- [83] Jukka K Nurminen. Parallel Connections and their Effect on the Battery Consumption of a Mobile Phone. In *7th Consumer Communications and Networking Conference (CCNC)*, pages 1–5. IEEE, 2010.
- [84] Jehwan Oh, Seunghwa Lee, and Eunseok Lee. An Adaptive Mobile System using Mobile Grid Computing in Wireless Network. In *Computational Science and Its Applications-ICCSA*, pages 49–57. Springer, 2006.
- [85] S. Ou, K. Yang, and Q. Zhang. An Efficient Runtime Offloading Approach for Pervasive Services. In *Wireless Communications and Networking Conference (WCNC)*, volume 4, pages 2229–2234. IEEE, 2006.
- [86] Shumao. Ou, Kun. Yang, and Jie. Zhang. An Effective Offloading Middleware for Pervasive Services on Mobile Devices. *Pervasive and Mobile Computing*, 3(4):362–385, 2007.

- [87] J. Porras, O. Riva, and M. Darø Kristensen. Dynamic Resource Management and Cyber Foraging. *Middleware for Network Eccentric and Mobile Applications*, pages 349–368, 2009.
- [88] Peng Rong and Massoud Pedram. Extending the Lifetime of a Network of Battery-Powered Mobile Devices by Remote Processing: A Markovian Decision-Based Approach. In *Proceedings of the 40th Annual Design Automation Conference*, pages 906–911. ACM, 2003.
- [89] Mendel Rosenblum and Tal Garfinkel. Virtual Machine Monitors: Current Technology and Future Trends. *Computer*, 38(5):39–47, 2005.
- [90] Alexey Rudenko, Peter Reiher, Gerald J Popek, and Geoffrey H Kuenning. The Remote Processing Framework for Portable Computer Power Saving. In *Proceedings of the ACM Symposium on Applied Computing*, pages 365–372. ACM, 1999.
- [91] Zohreh Sanaei, Saeid Abolfazli, Abdullah Gani, and Rajkumar Buyya. Heterogeneity in Mobile Cloud Computing: Taxonomy and Open Challenges. *IEEE Communications Surveys & Tutorials*, 16(1):369–392, 2014.
- [92] M. Satyanarayanan. Pervasive Computing: Vision and Challenges. *IEEE Personal Communications*, 8(4):10–17, 2001.
- [93] Mahadev Satyanarayanan. Avoiding Dead Batteries. *IEEE Pervasive Computing*, 4(1):0002–3, 2005.

- [94] Mahadev Satyanarayanan. Mobile Computing: The Next Decade. *ACM SIG-MOBILE Mobile Computing and Communications Review*, 15(2):2–10, 2011.
- [95] Mahadev Satyanarayanan, Paramvir Bahl, Ramón Caceres, and Nigel Davies. The Case for VM-Based Cloudlets in Mobile Computing. *IEEE Pervasive Computing*, 8(4):14–23, 2009.
- [96] Ya-Yunn Su and Jason Flinn. Slingshot: Deploying Stateful Services in Wireless Hotspots. In *Proceedings of the 3rd International Conference on Mobile Systems, Applications, and Services*, pages 79–92. ACM, 2005.
- [97] G. Sun, J. Chen, W. Guo, and K.J.R. Liu. Signal Processing Techniques in Network-Aided Positioning: A Survey of State-of-the-Art Positioning Designs. *IEEE Signal Processing Magazine*, 22(4):12–23, 2005.
- [98] Ruben Van den Bossche, Kurt Vanmechelen, and Jan Broeckhove. Cost-Optimal Scheduling in Hybrid IaaS Clouds for Deadline Constrained Workloads. In *Proceedings of the 3rd International Conference on Cloud Computing (CLOUD)*, pages 228–235. IEEE, 2010.
- [99] Luis M Vaquero, Luis Roderó-Merino, Juan Caceres, and Maik Lindner. A Break in the Clouds: Towards a Cloud Definition. *ACM SIGCOMM Computer Communication Review*, 39(1):50–55, 2008.

- [100] Tim Verbelen, Pieter Simoens, Filip De Turck, and Bart Dhoedt. AIOLOS: Middleware for Improving Mobile Application Performance through Cyber Foraging. *Journal of Systems and Software*, 85(11):2629–2639, 2012.
- [101] M. Weiser. The Computer for the 21st Century. *Scientific American*, 265(3):94–104, 1991.
- [102] Yonggang Wen, Weiwen Zhang, and Haiyun Luo. Energy-Optimal Mobile Application Execution - Taming Resource-Poor Mobile Devices with Cloud Clones. In *Proceedings of the 31st Annual International Conference on Computer Communications (INFOCOM)*, pages 2716–2720. IEEE, 2012.
- [103] Richard Wolski, Selim Gurun, Chandra Krintz, and Daniel Nurmi. Using Bandwidth Data to Make Computation Offloading Decisions. In *International Symposium on Parallel and Distributed Processing (IPDPS)*, pages 1–8. IEEE, 2008.
- [104] David Wong, Noemi Paciorek, Tom Walsh, Joe DiCelie, Mike Young, and Bill Peet. Concordia: An Infrastructure for Collaborating Mobile Agents. In *Mobile Agents*, pages 86–97. Springer, 1997.
- [105] He Wu, Sidharth Nabar, and Radha Poovendran. An Energy Framework for the Network Simulator 3 (ns-3). In *Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques*, pages 222–230, 2011.
- [106] Huaming Wu, Qiushi Wang, and Katinka Wolter. Tradeoff Between Performance Improvement and Energy Saving in Mobile Cloud Offloading Systems.

- In *Proceedings of the Internatioanl Conference on Communications: Ist International workshop on Mobile Cloud Networking Services (MCN)*, pages 738–742. IEEE, 2013.
- [107] Feng Xia, Fangwei Ding, Jie Li, Xiangjie Kong, Laurence T Yang, and Jianhua Ma. Phone2Cloud: Exploiting Computation Offloading for Energy Saving on Smartphones in Mobile Cloud Computing. In *Information System Frontiers*, pages 1–17. Springer, 2013.
- [108] Changjiu Xian, Yung-Hsiang Lu, and Zhiyuan Li. Adaptive Computation Offloading for Energy Conservation on Battery-Powered Systems. In *International Conference on Parallel and Distributed Systems*, volume 2, pages 1–8. IEEE, 2009.
- [109] K. Yang, S. Ou, and H.H. Chen. On Effective Offloading Services for Resource-Constrained Mobile Devices Running Heavier Mobile Internet Applications. *Communications Magazine*, 46(1):56–63, 2008.
- [110] Xinwen Zhang, Sangoh Jeong, Anugeetha Kunjithapatham, and Simon Gibbs. Towards an Elastic Application Model for Augmenting Computing Capabilities of Mobile Platforms. In *3rd International ICST Conference on Mobile Wireless Middleware, Operating Systems, and Applications*, pages 161–174. Springer, 2010.

- [111] Xinwen Zhang, Joshua Schiffman, Simon Gibbs, Anugeetha Kunjithapatham, and Sangoh Jeong. Securing Elastic Applications on Mobile Devices for Cloud Computing. In *Proceedings of the Workshop on Cloud Computing Security*, pages 127–134. ACM, 2009.
- [112] F. Zhu, M. Mutka, and L. Ni. Classification of Service Discovery in Pervasive Computing Environments. *Michigan State University, East Lansing, available at <http://www.cse.msu.edu/~zhufeng/ServiceDiscoverySurvey.pdf>, MSU-CSE-02-24*, 2002.
- [113] F. Zhu, M.W. Mutka, and L.M. Ni. Service Discovery in Pervasive Computing Environments. *Pervasive Computing*, 4(4):81–90, 2005.