

**DEVELOPMENT OF OPTIMIZED FRAMEWORK FOR  
COMPUTATIONAL OFFLOADING IN  
MOBILE CLOUD COMPUTING**

A Thesis

Submitted in partial fulfilment of the requirements for the  
award of the degree of

**DOCTOR OF PHILOSOPHY**

in

**COMPUTER SCIENCE & ENGINEERING**

By

**Robin Prakash Mathur  
41500183**

**Supervised By**

**Dr. Manmohan Sharma  
Professor**



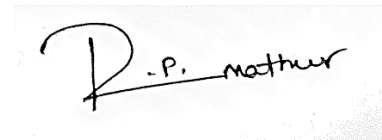
*Transforming Education Transforming India*

**LOVELY PROFESSIONAL UNIVERSITY  
PUNJAB  
2021**

---

## DECLARATION

I hereby declare that the thesis work entitled “Development of Optimized Framework for Computational Offloading in Mobile Cloud Computing” is an authentic record of my own work carried out as requirements for the award of the degree of Doctor of Philosophy in Computer Science and Engineering from Lovely Professional University, Phagwara, India under the guidance of Dr. Manmohan Sharma, during March 2016 to May 2021. All the information furnished in this thesis report is based on my intensive work, genuine and has not been submitted in whole or part for a degree in any university.

A handwritten signature in black ink on a light background. The signature reads "R. P. Mathur" in a cursive style. The 'R' is large and loops back, and the 'P' is also large and loops back. The name 'Mathur' is written in a smaller, more straightforward cursive.

---

Robin Prakash Mathur

July, 2021

---

## CERTIFICATE

This is to certify that the declaration statement made by the student is correct to the best of my knowledge and belief. He has completed the Ph.D. thesis **Development of Optimized Framework for Computational Offloading in Mobile Cloud Computing** under my guidance and supervision. The present work is the result of his original investigation, effort, and study. No part of the work has ever been submitted for any other degree at any University. The thesis work is fit for the submission and partial fulfillment of the conditions for the award of Doctor of Philosophy degree in Computer Science and Engineering from Lovely Professional University, Phagwara, India.



---

Dr. Manmohan Sharma

Professor

Lovely Professional University

Phagwara, Punjab, India

---

## ABSTRACT

In the recent past, mobile cloud computing (MCC) has arisen in the computing world, and many researchers and developers are working closely in this field. MCC is broadly a convergence of fields like cloud computing, wireless technology, and mobile computing. The rich computational resources available in the cloud are utilized to execute the mobile task using the concept of offloading. The compute-intensive part of the application is transferred to the cloud for execution. Upon the completion of task execution in the cloud, the results are sent back to the mobile device. In this way, the resource hunger applications are executed in a rich resource-intensive cloud. An appropriate Infrastructure is required to store the data and CPU processing off-site mobile device (not locally available) and execute on a remote cloud server. This technology is understood as a mobile cloud in recent times. Computation offloading solved various problems in a mobile cloud scenario. The task can be uploaded to the cloud or surrogate server to cope with the incapability of low-power smartphones. Computation like multimedia processing, image processing, audio processing, 3D rendering, security, gaming, and text processing needs a lot of energy to execute. A lot of energy is consumed in a compute-intensive task, and by offloading the task to the cloud, device energy can be saved.

Offloading in a better bandwidth environment of 4G and 5G connectivity will reduce the application's round trip time (RTT). Thus, performance will also improve in terms of running an application. The devices having low processing capabilities can take advantage of offloading and can enjoy the feel of high-end devices. An important question in the offloading scenario is which part of the application needs to be offloaded remotely. In order to identify that, the application needs to be partitioned. In this work, the graph partitioning approach is considered based on spectral graph partitioning with the Kernighan Lin algorithm. Experimental results show that the

proposed approach performs optimally in partitioning the application. The proposed technique gave better results than the existing techniques in terms of edge cut, which is less, concluding minimum communication cost among components and saving the mobile device's energy.

The decision to offload is a significant concern. The mobile device decides whether to execute the task locally or move on a remote server by assessing the storage, processing, and bandwidth parameters. A decision engine is a substantial component in the offloading framework, which helps decide when to offload the remote or cloud server. In the process of offloading decision, various profilers like network, device, and program profiler collect information related to network, application, battery level, and CPU cycle, which help the solver to the decision for the offloading. Energy and performance parameters are often evaluated during this phase. The decision engine's accuracy should be high for the flawless execution of the application during the offloading process. A technique has been proposed in this work by performing a stack ensemble approach on machine learning techniques like the Gaussian approach, multi-layer perceptron, k-nearest neighbors, and linear regression. It considers the various dynamics of the environment like task size, bandwidth, device battery, and device mobility. The proposed model performs better than other decision-making algorithms in terms of execution time and CPU utilization and achieves higher accuracy in making decisions while offloading the compute-intensive task to the remote server.

Mobility is a fundamental aspect of MCC where the mobile device gets connected to the cloud or edge server through the intermediate cellular network during roaming for the offloading process. The seamless connectivity with the network is required for the mobile device to remain connected to the cloud or edge server and completes the offloading effectively. Roaming is a significant procedure in mobility management; with the help of this, the customer automatically receives the calls, sends data, and travels outside the home network in Heterogeneous Access Networks (HAN) environments. The handoff procedure is a two-step process that transfers an active call from one cell to another, i.e., when a mobile node (MN) travels into a different cell

while a conversation is in progress, the MSC immediately switches the call to a new channel belonging to a new base station. Handoff dropped count depicts the scenario when user equipment does not get the required signal strength during handover and loses the connectivity with the base station of the cellular network. In this work, the seamless mobility scheme has been proposed based on the heterogeneous network of Wi-Fi and 4G networks. The proposed scheme is based on the fourth-order Markov model for mobility prediction and received signal strength (RSS) of the network nearest to the next predicted move of the device. The proposed scheme performed better as compared to SINR based handoff mechanism based on the number of handoffs and dropped count during device mobility in urban, semi-urban, and rural areas.

During offload, the job needs to be queued on the cloud servers and allocated to the virtual machines. Task scheduling is an important step where the mobile task is assigned to the servers and processed somehow. In the overall offloading process, energy conservation is a significant concern. The scheduling problem involves mapping the offloaded task to the cloud server while satisfying the energy and time constraints. When the task is offloaded from the mobile device to the cloud server, it reaches the cloud service provider's server. The cloud schedulers are a fully-managed entity in the cloud service providers. It minimizes the human intervention in scheduling the task and provides a reliable solution. The tasks are scheduled on various virtual machines available in the physical servers of the data centers. The cloud service provider manages all information about the task that approached it for processing. The Data center broker policy helps the cloudlets (task) to assign the virtual machines. The data center policy must be appropriate for the minimum execution time of the cloudlet. Similar to web applications, a mobile application consists of different tasks. The thesis presents a hybrid scheduling scheme based on particle swarm optimization (PSO) and bacterial foraging optimization (BFO). This scheme performs better when compared to other variants of PSO in terms of makespan and energy efficiency.

---

## ACKNOWLEDGMENT

I would take this opportunity to give sincere thanks to everyone who contribution led me to complete this thesis work. First and foremost, I would bow my head in front of God for giving strength and energy to carry my work with zeal and enthusiasm throughout this thesis work.

With immense pleasure and gratitude, I would like to express my sincere thanks to my supervisor Dr. Manmohan Sharma, Professor, School of Computer Application, Lovely Professional University. In my journey towards this degree, I have found a mentor who is knowledgeable and warm-hearted. He has provided his incredible support and guidance at all times and has given me worthy counselling, suggestions and recommendations in my pursuit for the knowledge.

I express my sincere thanks to Dr. Rajeev Sobti, Professor and Senior Dean, School of Computer Science and Engineering, Lovely Professional University for his kind support and encouragement at all times.

I would like to thanks the School of Computer Science and Engineering, Division of Research and Development and Central Library of the University for providing continues guidance and resources required for the conduct of my research work.

I am deeply thankful to my Parents, Wife Kriti Mathur, Dear Son Lavish Mathur and brother-in-law for their love, support, continues motivation and encouragement. Without them, this thesis would never have been written.

## LIST OF FIGURES

Figure 1-1MCC: Intersection of mobile computing and cloud computing [2] .....	2
Figure 1-2 Mobile cloud computing architecture [4].....	3
Figure 1-3 Computational offloading in mobile cloud computing .....	6
Figure 1-4 Decision to offload or not.....	10
Figure 1-5 Mobility Management in mobile cloud computing.....	11
Figure 1-6 Workflow of the thesis .....	19
Figure 2-1 Stages of the offloading process.....	20
Figure 2-2 Methods of computational offloading .....	21
Figure 2-3 Thinkair framework for offloading [18].....	23
Figure 2-4 Offloading mechanism in mobile cloud [24] .....	24
Figure 2-5 Jade framework [29] .....	25
Figure 2-6 Parameters deciding the behavior of code partitioning .....	28
Figure 2-7 LP-based code partitioning scheme in MCC [49].....	31
Figure 2-8 Mixed form of code partitioning scheme in MCC [49].....	32
Figure 2-9 A Graph-based code partitioning scheme in MCC [49] .....	33
Figure 2-10 Directed Acyclic Graph (DAG) representing a mobile application .....	34
Figure 2-11 Consumption Graph and Weighted Consumption Graph [40] .....	36
Figure 2-12 High-level view of MAUI's architecture [39]. .....	36
Figure 2-13 Evolution of Wireless communication [61].....	47
Figure 2-14 Frequency reuse in cellular cell.....	47
Figure 2-15 Basic structure of a cellular network [75] .....	48
Figure 2-16 Basic structure of a 3G network.....	49
Figure 2-17 Basic structure of a 4G network.....	50
Figure 2-18 Horizontal and vertical handoffs [76] .....	51
Figure 2-19 Desirable handoff features [76].....	52
Figure 2-20 Parameters used for making VHD decisions.....	53
Figure 2-21 Mobility model [61] .....	53
Figure 2-22 M <sup>2</sup> C <sup>2</sup> : A Mobility Management Scheme for MCC [77].....	54
Figure 2-23 Task scheduling in mobile cloud computing [86] .....	55
Figure 3-1 Partitioned component offloaded on cloud sever .....	62



Figure 3-2 Directed Acyclic Graph.....	64
Figure 3-3 Different types of graph topologies.....	65
Figure 3-4 Phases of graph partitioning.....	66
Figure 3-5 Scenario representing communication based on edge-cut .....	67
Figure 3-6 Flowchart representation of the proposed approach.....	68
Figure 3-7 Graph represents the edge cuts results of spectral ands KL algorithm.....	75
Figure 3-8 Graph represents the edge cuts results of the random .....	76
Figure 3-9 Graph represents the edge cuts results of the multi-level KL partitioning approach .....	77
Figure 3-10 Comparison of the edge cut results of spectral with and without KL approach, random partitioning with KL, and multi-level KL.....	78
Figure 4-1 Offloading decision process .....	84
Figure 4-2 Multilayer perceptron model.....	85
Figure 4-3 Logistic regression model .....	87
Figure 4-4 Stack ensemble approach used for predicting the offloading decision .....	88
Figure 4-5 ROC curve of the different algorithm and the proposed methodology .....	93
Figure 5-1 Computational offloading of task during mobility in MCC .....	101
Figure 5-2 N state Markov chain model for mobility prediction in MCC .....	101
Figure 5-3 Transition matrix .....	102
Figure 5-4 Handoff comparison between the two different strategies .....	106
Figure 5-5 Comparison of the number of handoffs dropped.....	108
Figure 6-1 Virtualization in cloud computing.....	110
Figure 6-2 System model for scheduling the offloaded task to the cloud server .....	111
Figure 6-3 Chemotaxis process of the bacteria [141] .....	115

## LIST OF TABLES

Table 1-1 Challenges of mobile user and solution by mobile cloud computing .....	4
Table 2-1 Code partitioning techniques in MCC .....	29
Table 2-2 Various decision making techniques in MCC .....	38
Table 2-3 Task scheduling schemes in MCC framework.....	58
Table 3-1 Comparison of graph partitioning techniques with proposed technique .....	74
Table 4-1 Features considered for offloading decision .....	84
Table 4-2 Performance of different algorithms and proposed methodology .....	92
Table 4-3 CPU utilization of Decision engine module (in percentage) .....	94
Table 4-4 Execution time of Decision engine module (in a sec) .....	95
Table 5-1 Base stations feature based on the cellular environment .....	104
Table 5-2 WLAN access point features .....	105
Table 5-3 Handoff results in a different environment.....	106
Table 5-4 Handoff dropped results in a different environment.....	107
Table 6-1 Parameters considered in the simulation .....	120
Table 6-2 Execution time of the task in different techniques.....	121
Table 6-3 Energy consumption of the task in different techniques.....	122

# CONTENTS

<b>Declaration.....</b>	<b>ii</b>
<b>Certificate.....</b>	<b>iii</b>
<b>Abstract.....</b>	<b>iv</b>
<b>Acknowledgment.....</b>	<b>vii</b>
<b>List of figures.....</b>	<b>viii</b>
<b>List of tables.....</b>	<b>x</b>
<b>1. Introduction.....</b>	<b>1</b>
1.1 Mobile cloud computing.....	1
1.2 Background.....	6
1.2.1 Computational offloading.....	6
1.2.2 Code Partitioning.....	8
1.2.3 Decision engines.....	8
1.2.4 Mobility Management.....	10
1.2.5 Task Scheduling in MCC.....	12
1.3 Motivation.....	12
1.4 Objectives of the study.....	13
1.5 Thesis Contributions.....	14
1.6 Structure of the thesis.....	16
<b>2. Review of Literature.....</b>	<b>20</b>
2.1 Computational offloading and frameworks.....	20
2.2 Code partitioning in MCC.....	27

2.3 Offloading decision making in MCC .....	36
2.4 Mobility mechanism in MCC .....	46
2.4.1 Wireless communication technologies.....	48
2.4.2 Handoff Management .....	50
2.5 Scheduling mechanism in MCC.....	55
2.6 Research Gaps and Challenges.....	59
2.7 Summary.....	61
<b>3. Code partitioning during computational offloading in MCC.....</b>	<b>62</b>
3.1 Introduction .....	62
3.2 Code partitioning using a graph model .....	64
3.3 Multi-level graph partitioning .....	65
3.4 Problem statement .....	67
3.5 Algorithm description.....	68
3.6 Performance Evaluation .....	73
3.6.1 Experimental Setup.....	73
3.6.2 Results and Discussion .....	74
3.7 Summary.....	79
<b>4. Offload decision making in computational offloading .....</b>	<b>80</b>
4.1 Introduction .....	80
4.2 Offload decision engine in MCC.....	81
4.3 Methodology.....	83
4.3.1 Offloading decision mechanism .....	83
4.4 Feature Selection .....	84
4.4.1 Multi-layer perceptron (MLP) .....	85

4.4.2 K-nearest neighbor (KNN) .....	85
4.4.3 Gaussian naïve Bayes method .....	86
4.4.4 Logistic Regression.....	87
4.5 Proposed stack ensemble approach .....	87
4.6 Performance Evaluation .....	90
4.6.1 Experimental Setup.....	90
4.7 Results and Discussion.....	92
4.8 Summary.....	96
<b>5. Mobility management scheme during computational offloading.....</b>	<b>97</b>
5.1 Introduction .....	97
5.2 Mobility management in MCC.....	98
5.2.1 Network ad Cloud Probing .....	99
5.3 Proposed mobility scheme during computational offloading.....	100
5.4 Proposed scheme for mobility management.....	103
5.5 Performance evaluation .....	104
5.5.1 Experiment Settings.....	104
5.5.2 Results and Discussion .....	106
5.6 Summary.....	108
<b>6. A multi-objective task scheduling scheme in mobile cloud computing .....</b>	<b>109</b>
6.1 Introduction .....	109
6.2 Related work.....	111
6.2.1 Task scheduling during computational offloading.....	111
6.2.2 Multi-objective approach of task scheduling.....	113
6.3 Proposed approach for task scheduling model .....	114
6.3.1 Bacteria Foraging Optimization.....	114

6.3.2 Particle swarm optimization (PSO) .....	116
6.4 Performance and Evaluation.....	119
6.4.1 Experimental setup .....	119
6.4.2 Results and Discussion .....	121
6.5 Summary.....	123
<b>7. Conclusion and future work .....</b>	<b>124</b>
7.1 Conclusion and Discussion.....	124
7.2 Future scope.....	126
<b>REFERENCES.....</b>	<b>127</b>
<b>LIST OF PUBLICATION .....</b>	<b>144</b>

---

# CHAPTER 1

## INTRODUCTION

---

### 1.1 Mobile cloud computing

As per the Ericsson mobility report 2020 [1], the expected number of smartphone users in 2025 will be around 8.9 billion. The demand for smartphones in developing nations like India and China is rising day by day. Android, BlackBerry, and Windows OS-based smartphones have their place in the commercial market. These devices provide a different type of features which enable the users to do various task related to location-based, image, networking and sufficient space to store the data. Users can run different smartphone applications like gaming, speech recognition, image and video-related editing, and navigation-based apps with these features. Such applications are resource-intensive applications that require a huge amount of CPU processing and battery, which are sometimes not answered by existing smartphones.

Emerging technologies like MCC, ubiquitous computing, IoT have adopted computational offloading to provide high-quality services (QOS) to the users. The *mobile cloud computing* concept has emerged in the recent past and attracts researchers and developers worldwide. It broadly means to run an application on the remote rich server. We can view our mobile phones as a client in this case, which runs applications like Google, Facebook, which are cloud-based applications. It is an infrastructure required to store the data and CPU processing offsite mobile devices (not locally available) and enable mobile tasks on a remote cloud server.

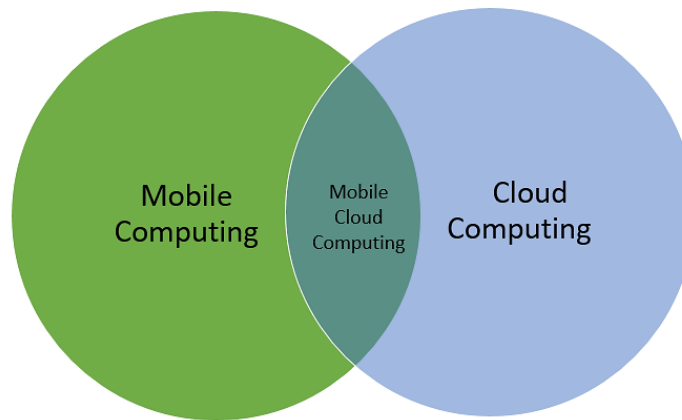


Figure 1-1 MCC: Intersection of mobile computing and cloud computing [2]

Fig. 1-1 depicts the mobile cloud computing which is an amalgamation of mobile computing and cloud computing. Mobile computing [2] is a field of wireless communication that empowers mobile devices to communicate during mobility. Handheld devices or personal digital assistants (PDA) can communicate using wireless mediums like WLAN or cellular networks. Rapid development in the field of telecommunication has been seen since the 1990s. People can communicate with each other during mobility. Several challenges still exist in mobility in mobile computing: coverage area, number of mobile users, limited bandwidth, and heterogeneous mobile networks. This factor affects the quality of services (QoS) to the user.

Cloud computing [3] is a distributed and parallel system where resources are available in a virtualized environment and used by the customers based on service level agreement (SLA). Cloud is a data center with nodes having resources available in the virtualized form provisioned in hypervisors. Users can process its execution on the cloud using web services like REST and SOAP. Users can access cloud services on pay-as-you prices and can increase or decrease their demand on an elasticity basis. Services used in the cloud are measured in metered form and transparent to the user. Cloud computing provides vital support to the failovers and keeps a replica of the data to achieve durability and reliability in the system.



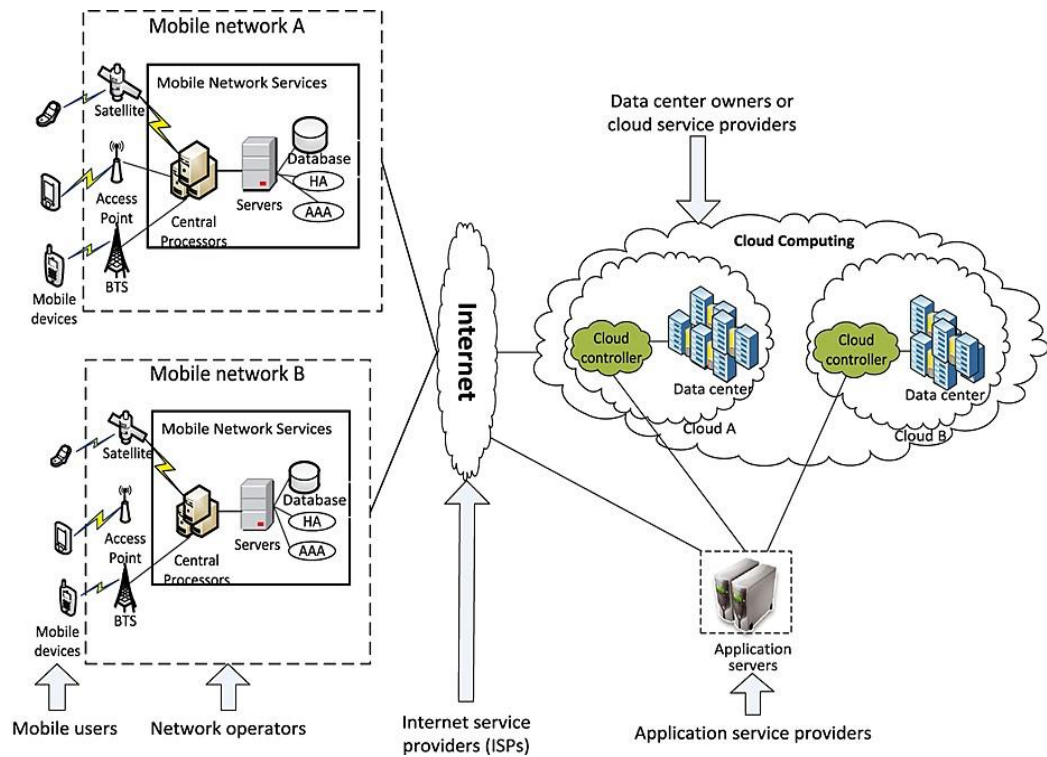


Figure 1-2 Mobile cloud computing architecture [4]

Fig. 1-2 represents the broad architecture of MCC. Smart devices like mobile phones, tablets are associated with the network operator (NO) through an access point or BTS. The cloud service request of the mobile user reaches the cloud service provider through the network operator and internet service provider (ISP). The mobile device is authenticated, authorized, and accounted (AAA) through the network service provider. ISP act as an intermediate between the NO and cloud service provider. The cloud service provider has data centers providing different services in the form of software, platform, and infrastructure through cloud controllers. It encompasses the power of virtualization and utility computing.

Table 1-1 Challenges of mobile user and solution by mobile cloud computing

<b>Challenges of the mobile users</b>	<b>Solutions delivered by Mobile Cloud Computing (MCC)</b>
Lack of Storage Capacity	MCC provides an extensive data storage facility and its access to the mobile user. Examples are Image Exchange, Flickr, and Amazon S3.
Users need reliable backup and security for their information	Reliability can be improved by the data access and running application on the cloud. Reliability is a critical factor of cloud services.
Energy is a significant challenge in mobile device	Computational offloading saves a good amount of energy by immigrating the compute-intensive task to the cloud from the smart mobile device.
Incapability to process an application when having a low-end hardware device	The mobile cloud provides solutions to mobile users which are having low-end hardware availability. They merely required optimal bandwidth for processing through the cloud server.

The various challenges of mobile users like lack of storage, reliable backup, energy consumption, and inability to process a compute-intensive application can be solved using cloud services. Various issues and solutions of mobile users are presented in Table 1-1.

There are various open challenges in the process of offloading in MCC are:

- a. *Uninterrupted internet connectivity and Bandwidth issue:* The smart device must have

good bandwidth connectivity, so offloading becomes possible for the device. Loss of signal may temper the offloading process, and again device need to execute the task locally in case of non-availability of mobile signals. The emergence of 5G technology, cognitive radio, and femtocells can help in providing seamless and quality services.

b. *Privacy and security*: Privacy and security must be seen as a crucial factor during the offloading process and helps in establishing and maintaining the trust of the mobile user. Data should be not being compromised in any sort of attacks, and privacy must be preserved of the user.

c. *Service Convergence*: Single cloud will not solve the computational problem and meet user expectations. Schemes need to formulate where users can exploit several clouds in a unified way. Different cloud vendors need to have a proper service level agreement as a migration of data is possible in the application processing. The pricing mechanism needs to ease out when the user utilizes the services of a mobile service provider and a cloud service provider.

d. *Offloading during mobility*: Handover in the perspective of MCC is less explored. Offloading becomes a difficult task during device mobility. Unstable network conditions and frequent handovers are a big challenge that needs to be addressed. Robust middleware and mobility management paradigm need to work upon for creating an effective offloading environment.

e. *Application partitioning*: *Granularity* of the application needs to be addressed correctly, seeing the network conditions, the data size of the application, battery consumption, and CPU cycles required. The mobile application consists of compute-intensive and graphics-intensive code. The compute-intensive workload needs to be carefully partitioned based upon the environment dynamics for energy saving. Still lot of scope is present where an application can be dynamically partitioned based on current mobile device conditions.

The mobile cloud cannot be only seen as a powerful machine for offloading purposes nor as an only pool of large virtual machines but can be explored with the great opportunity of clouds such as elasticity, parallelization with the help of map-reduce, and its utility

computing model. Budding technologies like cloudlets, Web 4.0, and Hypervisor virtual machines are boosting the popularity of MCC. Exploring the possibility of offloading in enterprise android applications can create a revolution in the mobile industry and improvise various business applications.

## 1.2 Background

In the era of highly configured mobile devices, sometimes users are under compulsion to act as thin clients, have energy constraints, storage issues, and process incapability. Some significant tasks in mobile devices like multimedia processing, image recognition, gaming, and text processing consume high resources that users need to rethink the task being performed. Computational offloading provides a platform to transfer the task from the user device to the server on the cloud to perform the necessary computation and serve the user. Several issues need to be addressed for computational offloading in MCC.

### 1.2.1 Computational offloading

A glimpse of the term '*computational offloading*' is found in the 1990s, where researchers have found that around 51% of the energy consumption of a portable computer battery can be saved through remote process execution [5]. Such inspiration is continued to date, and the methodology behind the computational offloading is used in recent technologies like mobile computing and IoT.

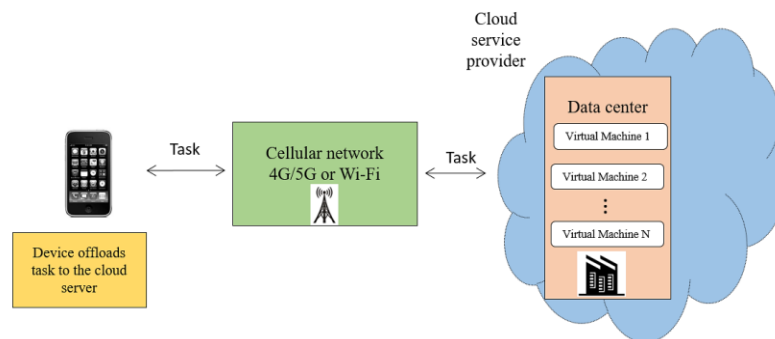


Figure 1-3 Computational offloading in mobile cloud computing

Fig. 1-3 depicts the computational offloading process which solves various problems in a mobile cloud scenario. In mobile cloud computation [6], the task can be offloaded to the cloud or surrogate server to cope with the incapability of low-power smartphones. Significant computation like multimedia processing, image recognition, gaming, text processing needs much energy to execute. The decision to offload is a major concern. By accessing the storage, processing, and bandwidth parameters, mobile devices decide whether to accomplish the task locally or move on a distant server.

Generally, mobile applications are resource-intensive and need a lot of energy and other processing requirements like CPU and memory. Based on various parameters like bandwidth, availability of cloud server, we can move our computational task over the cloud, process it, and get the outcome back on our device. Although it is not always possible to offload the task on the cloud, a decision engine is deployed on mobile to take a correct decision based on various parameters whether to offload or not [7]. Offloading can be defined based on two different categories.

**a) Partial Offloading:** In this type of energy-aware approach, the program is partitioned statically or dynamically on the client-side, and afterward, only a little required data is transmitted to the cloud server. Thus, by transmitting reduced data, energy is saved up to a large extent. In static partitioning developer [8] can annotate the methods or classes as `@remotable`, which needs to execute on the cloud server-side. In some of the cases, the developer needs to ensure not to mark `@remotable`. (i) code that implements user applications user interface (UI) (ii) code responsible for Input/output of the mobile device and restricted to it only.

**b) Full Offloading:** The complete program is transferred to the server in this scheme, and the programmer cannot amend the source code. The program is primarily performed within the client-side, and if it does not get executed within a specific time frame, the code is offloaded to the server-side. In this strategy, the short applications are executed on the client-side and extensive application is executed on the server as energy is the major constraint.

### **1.2.2 Code Partitioning**

Code needs to be partitioned before it is uploaded to the cloud. Code profiler partitions the code either statically or dynamically. In the static case, the code which needs to execute remotely is annotated with remote, which means that these methods or classes will be executed on a remote server. In dynamic profiling, the code is partitioned during the execution of code based upon the resource configuration available at the moment.

Static analysis and history trace strategies [7] are implanted by a different mechanism to estimate the portion of code is intensive or not. Automated techniques are preferred over static as they can quickly acclimate the code to be executed in different devices.

In a MCC environment, generally, two agents work for the process, one on the smartphone device and the other on the cloud. The code partition process is conducted before the offloading process based on parameters like application identifier, device identifier, and RTT between the smartphone and the cloud.

Once the partition plan is done, the smart device transfers the required state to enable the remote execution. A good partition plan can improve the performance, but the delay can be observed in the decision-making process to offload. So, the code partition algorithm must be accurate and fast to achieve optimized offloading performance.

Code partition algorithm should have some desired characteristics like Real-time adaptability and Partition efficiency. The algorithm must adapt to network and device changes. Code partition should be made dynamically based on the input of network conditions. How much code needs to be offloaded is another challenge. Fine-grained partitioning is always much value compared to coarse-grained partitioned. Thus, achieving partition efficacy is another critical parameter in computational offloading.

### **1.2.3 Decision engines**

Decision engines [7] decide when to offload based on various inputs make available by the system profiling. The decision engine applies logic based on stochastic methods, fuzzy logic, linear programming, machine learning-based, etc. The objective of offloading is to transfer the computation to the resourceful server, which may be a distant place to improve

the device's performance and save energy. It is not always expected to offload the task on a remote server but depending upon device conditions and bandwidth, and the decision to offload can be taken. If one part of the code is processed on the smart mobile device and the other is offloaded on the cloud/surrogate server, then it is partial offloading. The time  $T$  taken to execute the task locally [8] is

$$T = W/S_m \quad (2.1)$$

where  $W$  is the computation amount required for the second part and  $S_m$  is the processing speed of the mobile device.

The  $d_i$  quantity of data takes  $d_i/B$  seconds to send to a selected server if the second component of the compute-intensive task is offloaded to a cloud server with bandwidth  $B$ . The benefit of offloading the task on a cloud server is only when the computation of the task, including the communication, can be achieved faster at the cloud server than local execution.

$$\text{Total time} = \text{communication time to/from server} + \text{computational time} \quad (2.2)$$

The offloading decision is generally taken after the inputs taken from the code profiler and system profilers. Code profilers determine "what to offload" merely we can say the code partitioning task and system profilers gather the information about the crucial parameters like available bandwidth, data size to be migrated, and the most important energy required to execute the code. A decision engine is a thinker, a differentiator which decides "when to offload" to the cloud server. The concept of Lyapunov optimization, dynamic programming, linear programming, fuzzy logic, and Markov chain can be applied to build the optimal decision engine. In some of the references, the offloading policy is also based upon the two-level genetic algorithms. Fig. 1-4 depicts the decision-making process where the mobile device takes decision to offload the component of the application to the cloud using cellular or wireless network or run the application on the mobile locally.

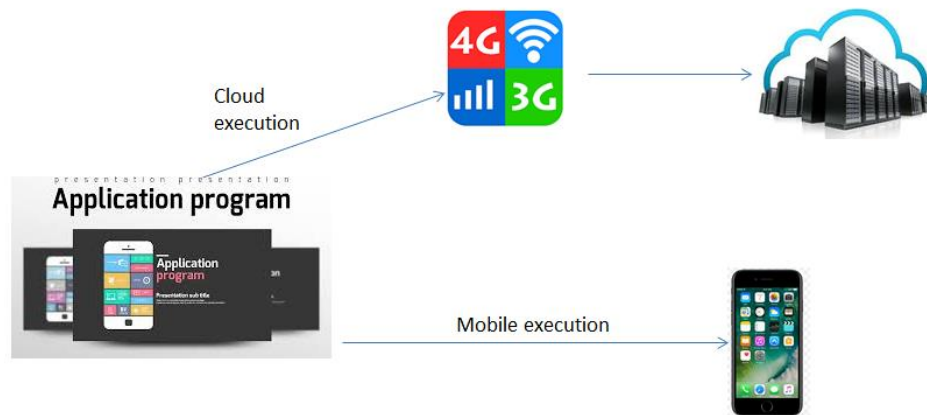


Figure 1-4 Decision to offload or not

### 1.2.4 Mobility Management

In the offloading process, the data that need to be processed is sent to the cloud or edge servers from the mobile device, get computed on the servers, and once the computation is done, the mobile device receives the computation results from the cloud or edge server. The offloading process in MCC may use heterogeneous types of wireless networks, which may include Wireless LAN (WLAN) and cellular services like 3G, 4G services, and even 5G services soon. Various issues get raised when the offloading application runs, like availability of connectivity, the energy level of mobile devices, and availability of the cloud or edge servers. The different types of mobile services [4] are available to the mobile device like Bluetooth, Wi-Fi, 2G/3G/4G services. The transitions among these services are getting possible by a concept of vertical handoff. The problems in MCC are similar to mobile computing, such as the issues [9] [10] related to handoffs, network delays, bandwidth, and limited battery energy. In the case of computational offloading, the mobile device or mobile nodes (MN) roams around different access networks like a mobile device may initially start some cloud services in the 4G network and commit offloading process in the Wi-Fi network due to its mobility.



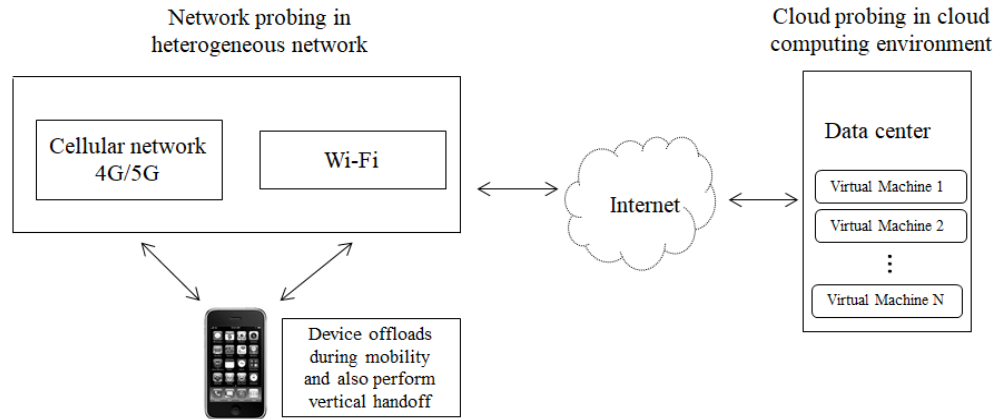


Figure 1-5 Mobility Management in mobile cloud computing

Handoff is a process when a mobile device changes its network from one to another. Fig. 1-5 discuss the mobility management in MCC framework. While in mobility, the mobile nodes initiate the process of handoffs for seamless connectivity. Handoff can introduce packet loss and long delays and hence can affect the cloud services further. Many applications in the modern computing environment are cloud-specific, like gaming applications, healthcare services, natural language processing (NLP) based applications, and computer vision. The mobile device can perform computations while roaming and may require cloud services for offloading purposes. The seamless transitions among networks can be either horizontal or vertical. When a device travels from one network to another without changing the network type, the process is called horizontal handoff, and if it changes the network, it is called vertical handoff. Heterogeneous networks (Hetnets) have various types of features like data rates, received signal strength (RSS), network capacity, bandwidth, and coverage span. The mobile device perceives these features and decides to select the best available network in its current location. The mobility of the device has a more substantial impact on the process of offloading. While the user is moving, the probability of changing the network is high. For the flawless offloading process, the transition among the cellular network must be smooth, and handoff must be minimized so that the mobile device remains attached to the cloud server.

### **1.2.5 Task Scheduling in MCC**

When the task is delegated to the cloud server from the mobile device, it reaches the cloud service provider's server. The cloud service provider manages all information about the task that approached it for processing. The Datacentre Broker policy [11] helps the cloudlets (task) to assign the virtual machines. The data center policy must be appropriate for the minimum execution time of the cloudlet. Similar to web applications, a mobile application consists of different tasks. These tasks can be represented as a directed acyclic graph(DAG). While the application's independent task can be executed simultaneously in multiple virtual machines, the dependent job needs to be synchronized as per their precedence order.

When speaking about task scheduling, achieving minimum makespan is considered an NP-hard problem. Most recent studies have focused on the cloud resources to the various cloudlets to optimize energy and execution time parameters. In this work, the particular task's execution time depends on the task size and the virtual machine's property. It is assumed in the work that the cloud service provider has a sufficient number of computational resources. The  $V$  number of virtual machines are deployed on the physical machines, and different virtual machines have a variety of processing units (CPU), random access memory (RAM), and networking capabilities. The data center brokers monitor all available resources and assign the machine to the task once approached. All jobs requiring the processing resources need to stand in a queue and based on the task scheduling scheme, tasks are planned to execute on the machine.

### **1.3 Motivation**

Computational offloading is an emergent field in the area of MCC. Smart mobile devices in the modern era are energy-hungry and required much energy for their computational processing. An application like multimedia processing, image recognition, gaming, and text processing consumes high resources to rethink the task being performed. A possible solution to the problem can be achieved by transferring our task to a resourceful cloud server for making optimal execution. The work will contribute to computational offloading

in MCC, which is challenging for mobile and cloud service providers in today's scenario. The topic is appropriate to the immediate environment as, at present, computational offloading can be possible due to the availability of 4G and, in the coming future, 5G technologies. It enables a mobile device to have the high bandwidth required for computation migration to distant cloud servers.

The applicational partitioning is a primitive task that is done in the offloading process. The compute-intensive portion of the application need lot of RAM and processing capability to process a task. Application partitioning identifies the task for offloading. Lower computational power and RAM capability motivate the developers to offload the task on the cloud servers, and thus application partitioning becomes a fascinating topic for research. Offloading decision is majorly affected by various factors like bandwidth, mobility of the device, size of the task, and battery level of the mobile device. Accessing accuracy and energy consumption during decision-making considering these parameters make it an interesting field to study. The mobility during the offloading process is a challenging task and motivates to work on mobility management. Handoff management and cloud probing is an essential and exciting topic that led the researchers to find the optimal solutions in terms of energy-saving and flawless connectivity to mobile users.

Lastly, the task scheduling on the cloud server is an open area. Many scheduling methods like machine learning and nature-inspired method attract researchers to find the optimal solutions. Computational offloading is a motivational topic in mobile cloud computing and will always have a scope of improvement in the coming future also.

#### **1.4 Objectives of the study**

The main objective of this thesis is to develop an optimized framework for computational offloading in MCC. There are broadly two types of offloading i.e., data offloading and computation offloading. In this work, compute-intensive offloading is considered for the study. The critical question which needs to answer in offloading is “what to offload.” The focus of the thesis is to develop a partitioning method for the code that will decide which portion of the application needs to be sent on the cloud server. The computational

capabilities of smart devices are limited in extent. Another question of “when to offload” is also challenging in offloading. A decision scheme needs to be developed for effective offloading on the cloud servers. Offloading depend on the various parameters of the device and mobile network. The mobility of the device needs to be addressed during the offloading process. A mobile device may roam and can change its position during offloading process. A mobility scheme is required to be built based on the mobile device's mobility pattern. After the data partitioning, offload decision making, and offloading during mobility, the task is scheduled on the cloud servers for processing. An optimal scheduling scheme is needed to place the task on the cloud servers.

Based on the study, the main objectives of this thesis would be:

1. To design an optimal algorithm for code partitioning in computational offloading.
2. To build a decision engine for computational offloading based on the dynamically changing environment
3. To develop a mobility-based offloading scheme where users can offload while moving from one location to other.
4. To develop a scheduling scheme that can prioritize the task on a virtual machine

## **1.5 Thesis Contributions**

1. In this research work, initially, a literature review regarding computational offloading in MCC has been pursued in chapter 2. The literature is studied from a different perspective like code partitioning, decision engines, computational frameworks, mobility management, and task scheduling. Further, the research issues have been identified in the computational offloading. The challenging part of the research is identifying *what, when, where, and how* to offload correctly. Studying the various types of existing architectures for offloading purposes is the prime step of this thesis.

2. The research contributes to identifying the compute-intensive part of the application. It

was found in most of the research papers that the UI part is locally executed while the computational task is offloaded on the cloud server. Code partitioning will divide the code into two parts- One to execute local and the other to execute remotely. Code partitioning algorithm is studied in-depth, and modification has been made in the dynamic code partitioning scheme where code is partitioned based on the current parameter like network, the bandwidth available. In order to identify that, the application needs to be partitioned. In this work, the graph partitioning approach is considered based upon the spectral graph partitioning with the Kernighan Lin algorithm. Experimental results show that the proposed approach performs optimally in partitioning the application. The proposed technique gave better results than the existing techniques in terms of edge cut, which is less, concluding minimum communication cost among components and saving energy of the mobile device.

3. The third contribution of the research is to build the decision engine. A decision engine is a reasoner that infers the "When to offload" question. To judge when to offload is very important since various parameters like bandwidth, device battery play a crucial part in this decision. A technique has been proposed by performing a stack ensemble approach on machine learning techniques like the Gaussian approach, multi-layer perceptron, k-nearest neighbors, and linear regression. It considers the various dynamics of the environment like task size, bandwidth, device battery, and device mobility. The proposed model performs better than other decision-making algorithms in terms of execution time and CPU utilization and achieves higher accuracy in making decisions while offloading the compute-intensive task to the remote server.

4. A fourth contribution of the research will make the mobile cloud computing user device offload its computational task while moving from one location to another. The mobility pattern of the user can be recorded, and based on the location, the offloading scheme can be developed where the user device can offload the task to the nearest cloud server as per its convenience. The seamless mobility scheme has been proposed based on the heterogeneous network of Wi-Fi and 4G networks. The proposed scheme is based on the fourth-order Markov model for mobility prediction and received signal strength (RSS) of the network nearest to the next predicted move of the device. The proposed scheme

performed better as compared to SINR based handoff mechanism based on the number of handoffs and dropped count during device mobility in urban, semi-urban, and rural areas.

5. The fifth contribution of the research is the task scheduling scheme in a mobile cloud environment. In the end, the task is offloaded to the cloud server where virtual machines (VMs) exist. A single physical cloud server contains  $N$  virtual machines where the tasks are forked into the primary and secondary tasks. Allocating the same VM to the main task is required since the primary and secondary tasks sometimes communicate. It is necessary to schedule the offloaded task to execute without any deadlock or starvation-like problem. This work offers a hybrid scheduling scheme based on bacterial foraging optimization and Gaussian-based multi-objective particle swarm optimization (GMOPSO). When compared to other PSO variations, this method outperforms them in terms of makespan and energy efficiency.

## **1.6 Structure of the thesis**

Chapter 1 presents the introduction to the concepts of MCC. It provides the details of the areas which are presented in the work. The benefits of MCC have been presented in the chapter. The concept of computational offloading has been presented, transferring the data from mobile to cloud server to overcome the resource limitations present in the mobile. It includes the discussion on code partitioning, where code needs to be partitioned before uploading to the cloud. It also discussed decision engines that decide when to offload based on various inputs provided by the system profiling. Mobility management is also discussed, which defines that the offloading process in MCC may use heterogeneous wireless networks, including Wireless LAN (WLAN) and cellular services like 3G, 4G services, and even 5G services soon. When the task is delegated to the cloud server from the mobile device, it reaches the cloud service provider's server. The concept of task scheduling is presented where the cloud service provider manages all information about the task that approached it for processing and scheduled it for execution.

Chapter 2 presents the literature review of the various task partitioning schemes that a special program structure usually implements in computation offloading or a design pattern

that allows a code to run locally or remotely and handles interactions between the local and remote code without affecting the functionality's integrity. It also provides a literature review on decision engines. Decision engines decide when to offload based on various inputs provided by the system profiling. The decision engine applies logic based on stochastic methods, fuzzy logic, linear programming, machine learning-based, etc. Further, it provides literature on mobility management in mobile cloud computing. A mobile device is connected to a distant cloud, and continuous mobility causes a disconnection problem. Suppose a device is connected through a 4G/ 3G network or in the future 5G with the cloud. If the device moves to such a place where the mobile network is not available, the connection with the cloud will be broken. The last section of the chapter provides the work is done so far in the field of scheduling in MCC and cloud computing. Once the task has been offloaded to a virtual machine, its execution plan or schedule is another challenge. The scheduling algorithm must be optimally designed so that the task's timely execution can be achieved and starvation or deadlock-like conditions can be avoided.

Chapter 3 presents the efficient partitioning technique considering an application as a graph. Two different tasks are primarily done during offloading, first partitioning an application and second, moving the created partition to the cloud or server. In this chapter, the focus is on the first task, and the graph partitioning approach is considered, which is based upon the spectral graph partitioning along with the Kernighan Lin algorithm. The proposed approach reduces the communication cost between the different application components in terms of edge cuts. Minimizing the communication cost between the components leads to saving the energy of the mobile device.

Chapter 4 presents an approach of an offloading engine that is placed in the mobile device. It must be light weighted and also provide highly accurate offloading decisions based on the statistics provided to it by the context analyzer. The chapter presents a proposed technique for the offloading decision that aims to achieve higher accuracy. It is based on the stacked ensemble approach considering various mobile device parameters. The proposed techniques aim to reduce the mobile device's processing time and CPU utilization while taking the offloading decision.

Chapter 5 addressed the issues of seamless connectivity and proposed a technique for reducing the number of handoffs and the dropped rate. It is based on the fourth-order Markov model for location prediction and an RSS-based scheme for handoff decisions. The mobile device perceives these features like data rates, received signal strength (RSS), network capacity, bandwidth, and coverage span and decides to select the best available network in its current location. The mobility of the device has a more significant impact on the process of offloading. While the user is moving, the probability of changing the network is high. For the flawless offloading process, the transition among the cellular network must be smooth, and handoff must be minimized so that the mobile device remains connected with the cloud server.

Chapter 6 proposed a hybrid scheduling technique based on Gaussian-based multi-objective particle swarm optimization (GMOPSO) and Bacterial foraging optimization (BFO). The GMOPSO provides us the global best solution, whereas using the BFO, the local best solution is improvised. The contribution can be summarized as follows. a) Minimize the energy consumption and makespan of the scheduling process. b) Simulation and performance evaluations of the proposed algorithm with existing approaches.

Chapter 7 presents the conclusion of the overall work, limitations and the future directions. The workflow of the thesis is discussed in the fig. 1-6.



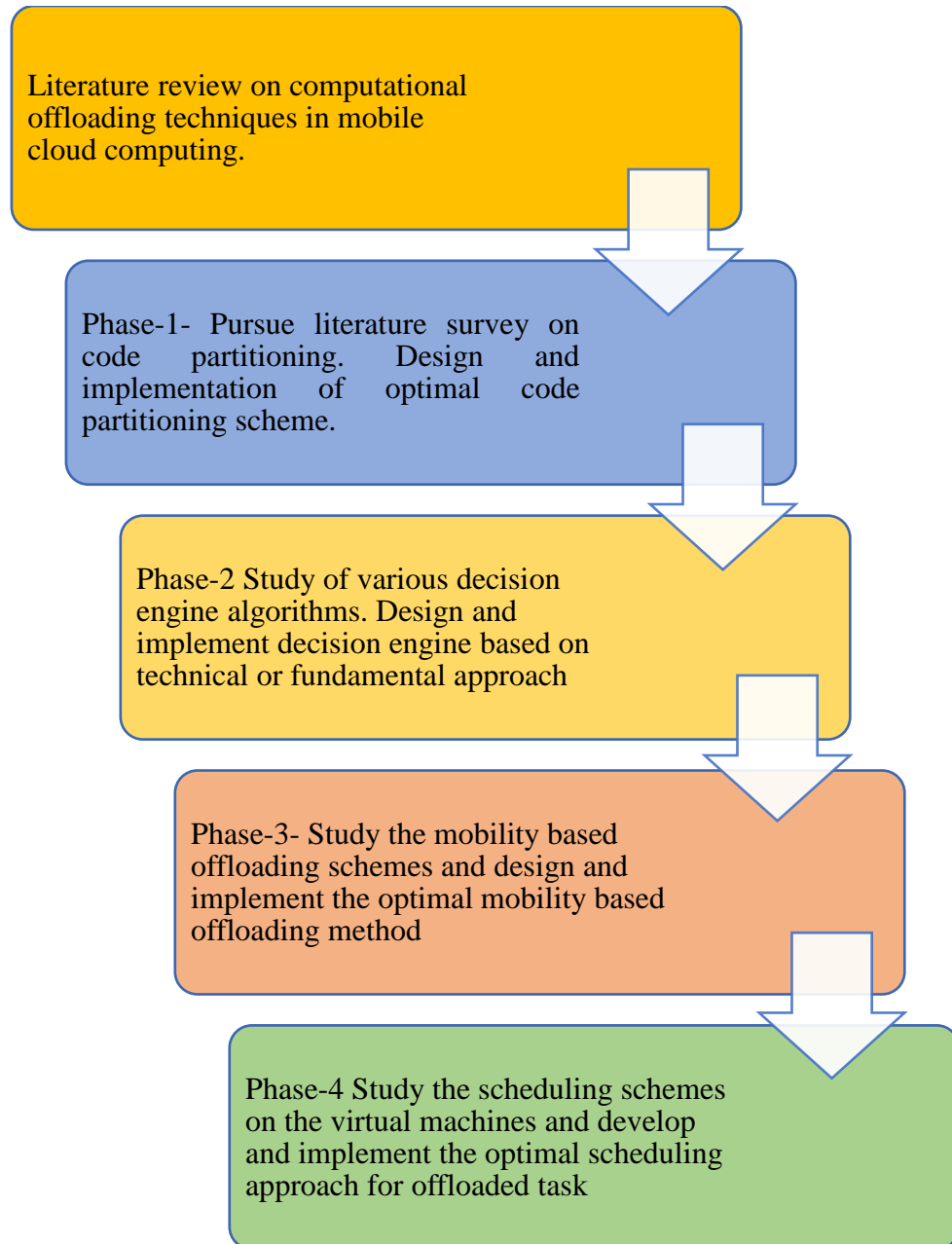


Figure 1-6 Workflow of the thesis

---

## CHAPTER 2

### REVIEW OF LITERATURE

---

In this chapter, the literature review has been presented. Computational offloading is an essential paradigm in MCC. It has been discussed along with some of the state of art frameworks that exist in this field. Further, this chapter emphasizes four significant sections of the thesis a) Application partitioning, b) decision making in MCC, c) mobility management in MCC, d) Task scheduling in the MCC environment.

#### 2.1 Computational offloading and frameworks

Offloading is a complex task that is performed in a step-by-step process. Fig. 2-1 present the major steps taken during the offloading process are partitioning an application, preparation for offloading, and decision to offload or not.

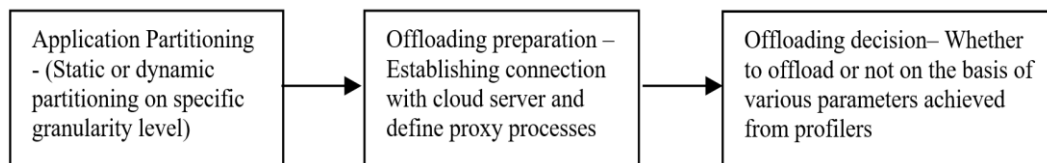


Figure 2-1 Stages of the offloading process

Deciding what to be offloaded is typically done during application partitioning. The different granularities of the application can be considered for offloading like object level, method level, class level, etc. Applications are comprised of both compute-intensive tasks and GUI-related tasks. The task which is responsible for the GUI cannot be offloaded. So, the compute-intensive part is partitioned either statically or dynamically. Annotation used by application developers is a popular style of static partitioning where the developer writes

local or remote with a code for partitioning purposes. Dynamic partitioning incurs an extra cost as, during the execution of an application, it is taking extra effort to identify the code for local or remote execution. Once the partitions are ready, the next step is establishing a connection with a cloud server, defining the proxy process on both the smart mobile device and a remote cloud server. The device should be robust enough to handle failure if a connection breaks with the cloud server. It must act intelligently by running a computation on the local device itself and provide results to the user. Since program states are transferred, re-executing a portion of the computation will not affect the correctness of the program. The next major step is whether to offload or not, i.e., offloading decision. Various profilers like network, device and program profiler collect information related to network, application, battery level, and CPU cycle, which help the solver decide for the offloading. Energy and performance parameters are often evaluated during this phase.

### Major approaches of computational offloading

The Fig. 2-2 deliberates offloading approaches in three main directions; Client-Server Communication methods, virtualization, and mobile agents.

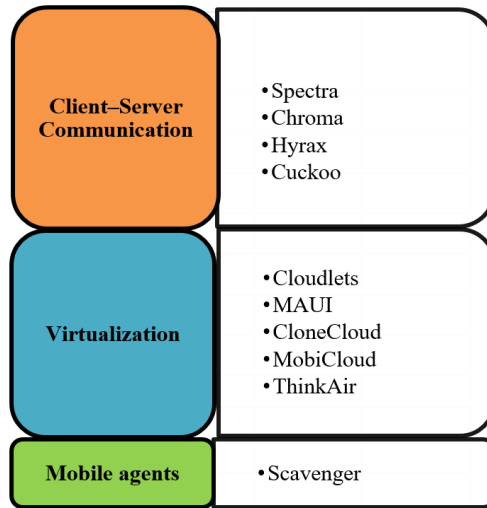


Figure 2-2 Methods of computational offloading

In the Client-Server Communication process communication, the offloading process utilizes the APIs of RPC, sockets, and RMI to offload the task communicate with the cloud

server. The advantage is this method is the stability that is offered to the mobile application developers. The disadvantage is that the application services need to be pre-installed on both ends, and it also not performs optimally due to the unavailability of the services during mobility of the device. Existing work like Hyrax [12], Spectra [13], Chroma [14], and [6] has developed techniques based on client-server communication.

A majority of frameworks use virtualization like MAUI [15], Clonecloud [16], Cloudlets [17], Thinkair [18], and Mobicloud [19]. Virtualization has reduced the work of programmers as rewriting of complete applications is not required in this method. However, the virtual machine synthesis takes time, and the compatibility issue also arises due to a dynamic mobile environment.

In the mobile agent method, the mobile code is partitioned and distributed on one or more surrogate servers. The cost assessment is done based on the speed of the server. There are issues related to security and agent management in this method. Scavenger [20] is a method which uses a mobile agent-based method for offloading purpose.

Various literature has been reviewed which have done considerable work in the field of computational offloading in MCC.

M. Satyanarayanan et al. [21] have proposed the concept of cyber foraging in which migrating the task from mobile device to surrogate server is discussed. The author has presented some real challenges like discovering the surrogate servers, trust formation between client and surrogate, load balancing, which are widely addressed in the recent past. RKK et al. [22] proposed the stack-on-demand (SOD) concept in which migration of light-weighted threads is done using the JVM environment. The SOD. supports the partial migration of thread data onto the server and thus optimizes the performance of the MCC process. The SOD. model is implemented into a Java distributed runtime named the SOD. S. Kosta et al. [18] proposed the framework for offloading work "Thinkair" over the cloud with its application server concept and broadly discussed profiling. Fig. 2-3 presents the Thinkair framework which have exploited the mobile device virtualization over the cloud and provide method-level computational offloading. Parallel execution of tasks over multiple virtual machines has been proposed in the concept. Thinkair framework can be

seen in fig. 2-3, which is implementing the offloading concept using the client-server model.

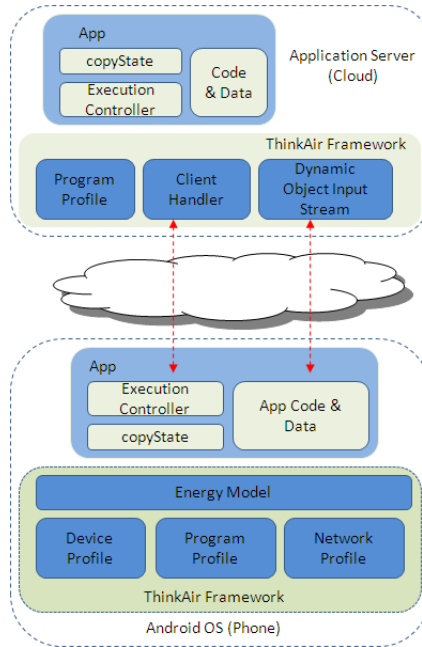


Figure 2-3 Thinkair framework for offloading [18]

R. Kemp et al. [6] proposed the primitive android-based framework CUCKOO for computational offloading. The framework offers a simple programming model using remote method invocation and IPC mechanisms for local and remote job execution. It provides a dynamic runtime system that decides whether a code will execute locally or remotely at runtime.

Yang Ge et al. [8] proposed an algorithm that is based upon a game-theoretic approach. The client resembles the player, and its strategy is to select one server which provides him energy-efficient offloading scheme. The researcher can achieve Nash equilibrium in polynomial time, which means it is an optimum solution where no player can find a better policy if he deviates from the current policy unilaterally.

Ejaz Ahmed et al. [23] provide an extensive survey for seamless application execution in MCC. They have focused on the study of state-of-the-art cloud-based mobile application execution frameworks (CMAEFs). Different frameworks are compared based on some

significant parameters and, in the end, have suggested open challenges in the field.

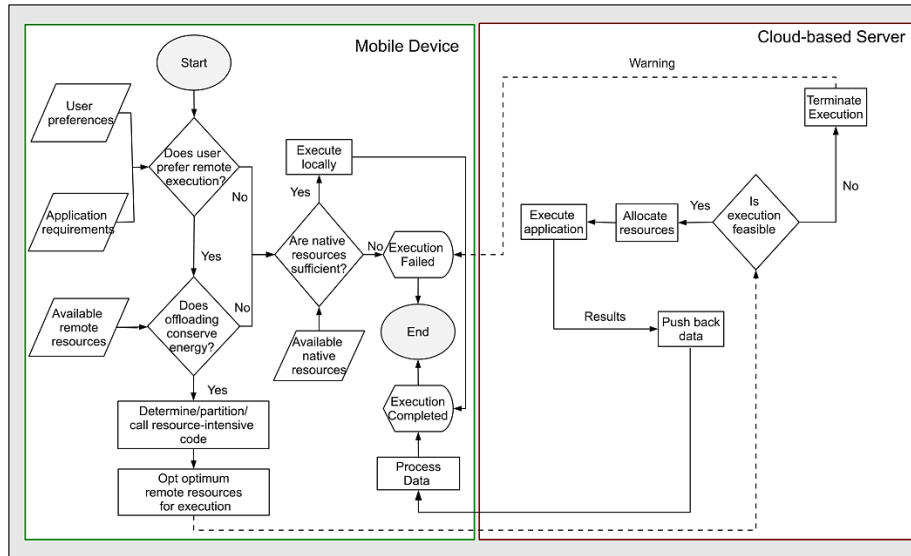


Figure 2-4 Offloading mechanism in mobile cloud [24]

Dejan Kovachev et al. [24] present Mobile Augmentation Cloud Services (MACS) middleware, enabling the adaptive extension of Android application execution from a mobile client into the cloud. Fig. 2-4 presents a middleware which performs the application partitioning, resource monitoring, and computation offloading. The application is partially divided dynamically into two parts: running locally and running on a cloud server. The middleware supports android-based offloading and achieved parallelization of the offloading services.

S. H. Hung et al. [25] proposed the profile-based policy manager where agent programs and integrated VPA tools are used for dynamic profiling. They have utilized cloud-based services for getting better energy and performance factor.

S. Yang et al. [26] proposed a two-phased portioning mechanism MACO in which, after code extraction, the user interface is executed locally, and the computational part is offloaded on a remote server. They have proposed code partitioning to present the application as two components- UI components and computational segment. Sending UI information is inefficient over the network. Thus, the proposed method divides applications

so that UI runs locally and code computational is transferred on a remote server.

P. Balakrishnan et al. [27] proposed an energy-efficient algorithm for offloading in which applications are partitioned into several interconnected partitions like task interaction graphs (TIG) and scheduled for resources with minimum slack time. They have used the technique dynamic voltage and frequency scaling (DVFS) for better power consumption modeling.

A. Mtibaa et al. [28] proposed the offloading scheme where offloading is done on the set of mobile device which they named as mobile device cloud(MDC) rather directly on a distant cloud server. The task will be offloaded on the stable and durable mobile nodes which are identified based on some social and contact history information. The proposed algorithms show 80-90 % of energy-saving than offload on a distant cloud server.

H. Qian et al. [29] proposed the android based framework Jade which dynamically changes its offloading strategy for energy-efficient offloading. Fig. 2-5 depicts the Jade framework, which is using the RPC mechanism for client-server communication.

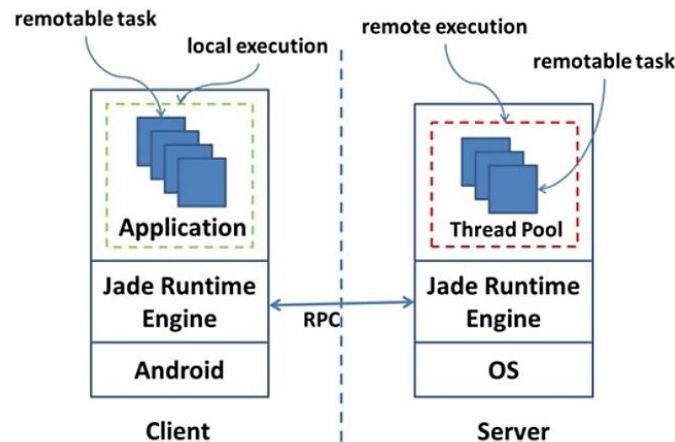


Figure 2-5 Jade framework [29]

M. P. Anastasopoulos et al. [30] have proposed the traffic-based computational based upon a multi-objective non-linear programming scheme to optimize the network performance, computational infrastructure, and battery lifetime in the worst case of delay condition. The researchers have used the concept of network calculus for the theory.

Mati B et al. [31] have proposed the multi-site offloading policy based upon the Markov decision process. The idea distinguishes between data and computation-intensive components of an application and performs data and process-centric multi-site offloading. To represent fading wireless mobile channels, they used the discrete-time Markov chain. The energy-efficient Multi-site Offloading Policy (EMOP) algorithm has been developed as an efficient solution to the multi-site partitioning problem, based on a value iteration algorithm (VIA).

Byung-Gon Chun et al. [16] have implemented CloneCloud, a flexible application partitioner that enables application-level VMs in mobile applications onto a device clone operating in a computational cloud. They used a dynamic profiler and optimization solver to migrate the method from mobile devices to the cloud. CloudClone migration works at the granularity of thread-level, making the whole process lightweight and energy-efficient.

Xinwen Zhang et al. [32] have proposed offloading schemes where the single elastic application is partitioned into platform-dependent or multiple independent weblets, which can execute locally or migrated on the cloud server. They have also discussed the cost model of an elastic application.

Y. Zhang et al. [33] have suggested that for a mobile user with intermittent connections while on the move, an efficient offloading technique is needed. The task can be offloaded to the servers based on the mobile device's mobility pattern, local cloud, and cloudlet availability. For the best development of the algorithm, they used the mathematical model Markov decision process. The MDP model is used to determine whether a program should be run locally or remotely.

Min Chen et al. [34] proposed mobility-aware-based caching and computational offloading in a 5G ultra-dense network. The authors have presented the different caching schemes and developed a hybrid offloading mechanism to achieve the tradeoffs among MBS, SBS, and D2D computational offloading.

M. V. Barbera et al. [35] have proposed an architecture where each mobile device is associated with a software clone on the cloud. They consider two clones (i) an Off-clone responsible for computational offloading and (ii) a back-clone used to restore user's data,



and the app is required. This concept helps to save energy and bandwidth both at the same time.

Mark S. Gordon et al. [36] have proposed an offloading scheme built over Dalvik Virtual Machine where threads can migrate freely over multiple virtual machines depending upon the workload. The researcher also proposed T-scheduler, which schedules the threads between the endpoints to optimize the throughput. Threads communications, VM synchronization, and thread migration have inculcated the scheme.

Marinelli et al. [12] have proposed the platform Hyrax, which is derived from Hadoop that cloud computing over android devices. It enables smart mobile devices to utilize the network for the various resources required for its task execution. After doing specific customization, the Hyrax can be used as Hadoop over a mobile device. Using the concept of MapReduce and HDFS, Hyrax performs offloading in mobile devices. Task partitioning can be done as per the MapReduce philosophy.

Hao Qian et al. [37] proposed a system of offloading to classify the local and remote workflow by annotation. Those marked as @local will execute locally, and @remote will execute on a distant cloud server. The scientific workflow is defined by the XAML file, where the node represents each step. The hierarchical structure of XAML makes it easy to analyze the relationship of the steps.

Diogo Lima et al. [38] have proposed the cyber foraging technique where they have adopted the programmer-driven partitioning model where the developer wrote annotations to partition the code from its bytecode to remotely executable code. Annotation includes 1 and 0 for local and remote execution, which influences the decision to offload. A decision will be affected by the network bandwidth, execution load, and available services provided by the cloud server.

## **2.2 Code partitioning in MCC**

Computational offloading in MCC utilizes the code partitioning approach to discrete the compute-intensive portion of the application for running it in distributed or cloud environment. The applications which can participate in the runtime partitioning scheme are known as elastic applications. These elastic applications get executed transparently and

seamlessly on the remote servers. Applications can be partitioned by using various partitioning schemes into separate components with different granularity

Code partitioning splits the application into two parts- compute-intensive and mobile device-specific like user-interface ensuring that semantics of the application can be preserved. Code partitioning is performed as the pre-processing step in the offloading process, deciding which portion of the application will run locally or remotely on the device. Application or code partitioning in mobile cloud offloading can be performed using different strategies to partition the application. Fig. 2-6 defines the parameters of the application where code partitioning can be performed.

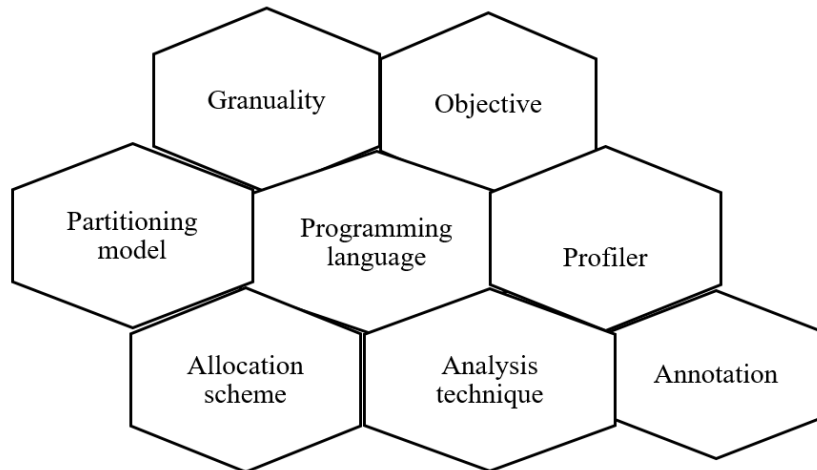


Figure 2-6 Parameters deciding the behaviour of code partitioning

Code partitioning in the application can be done on different granularities like module level, method level, object level, thread level, class level, task level, allocation site-level partitioning, and hybrid level partitioning. The objective of partitioning can be improving the performance, reducing memory constraints, reducing network overload, reducing programmer burden, and saving energy. Different partitioning models can be used to partition the application for offloading purposes like Graph-based, Linear programming based and Hybrid applications. Partitioning can support a single-level or multilevel programming approach. Various types of profilers like hardware, software, network

profilers can be used by application partitioning scheme. The decision to offload to remote or execute locally is decided by the allocation decision attribute, where the decision can be taken online, offline, and in a hybrid format. The analysis technique is used to identify the dependency of the component on each other. It can be done statically at the bytecode level or dynamically involving runtime profiling. Annotation is a type of metadata that talks about which component will be executed for partitioning. A programmer does manual annotation during the code development, stating that a particular piece of code will be executed remotely or locally. Automatic annotation decides in runtime about code availability for execution. Table 2-1 discussed the code partitioning scheme in MCC along with techniques of partitioning and its limitations.

Table 2-1 Code partitioning techniques in MCC

<b>Paper</b>	<b>Technique used for partitioning</b>	<b>Limitations</b>
ThinkAir [18]	ThinkAir provides a basic library that, when combined with compiler support, simplifies the job of the programmer: every procedure that should be considered for offloading is annotated with @Remote.	Requires developers to annotate source code methods
Cuckoo [6]	Use the current activity/service architecture in Android, which uses an interface specified by the developer in an interface specification language to separate compute-intensive elements (services) from interactive sections of the	Used android based activity model but still requires an optimal code to be offloaded

	programme (activities) (AIDL)	
MAUI [39]	Developers must use the "Remotable" annotation to indicate the can-be-offloaded methods of a .Net mobile application.	Requires developers to annotate source code methods
Clonecloud [16]	Modifies the Android Dalvik VM to include an application partitioner and execution runtime, allowing apps to offload chores to a cloned VM hosted by a Cloud server.	Static partitioning technique requires to use of a modified JVM
Jade [29]	A remotable class is a class that implements the Remotable Task interface. A remotable object is an instance (object) of remotable classes.	Static partitioning technique requires developers to annotate source code methods
Automatic scientific workflow on the local cluster and cloud [37]	Annotation for remote as @remote and local as @local	Static partitioning technique requires developers to annotate source code methods
Towards a new model for cyber foraging[38]	Propose to use annotations written by the developer to partition an application from its bytecode into remotely executable methods based on network conditions, execution workload, and no. of server available	Dynamic partitioning technique requires developers to annotate source code methods
An Optimal Offloading Partitioning Algorithm in Mobile Cloud Computing [40]	Based on the Consumption Graph (CG) and Weighted Consumption Graph (WCG), which is a dynamic based concept	Dynamic partitioning method but communication cost b/w local and remote server is a challenge

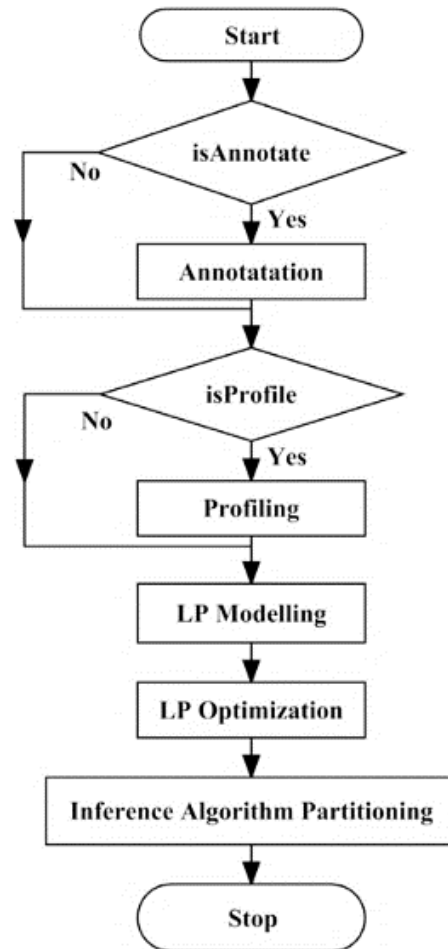


Figure 2-7 LP-based code partitioning scheme in MCC [49]

Linear programming (LP) is a mathematical representation of an code partitioning approach that uses an objective function to identify the best solution [41] [42]. The objective function is a linear type and helps in achieving a solution in a worst-case scenario. LP models help in formulating optimization equations in mobile applications while considering various variables as an integer value. In case of unavailability of the profilers and annotation also in mobile application, the linear programming helps to decide the partitioning module as seen in fig. 2-7.

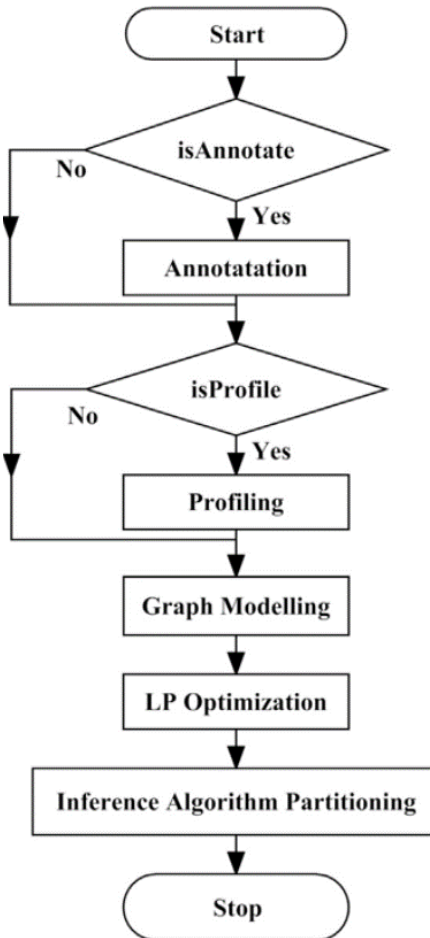


Figure 2-8 Mixed form of code partitioning scheme in MCC [49]

In order to increase the efficiency of APAs, the hybrid application partitioning algorithm combines elements from both graph-based and LP-based application partitioning algorithms by extracting relevant aspects which is discussed in fig. 2-8. Mobile Assistance Using Infrastructure (MAUI) [39] performs hybrid application partitioning by considering the application as a graph and performing linear programming also for optimization of the partition results. In similar manner, other framework like cloudclone [16] and [43] [44] have used the hybrid application partitioning.

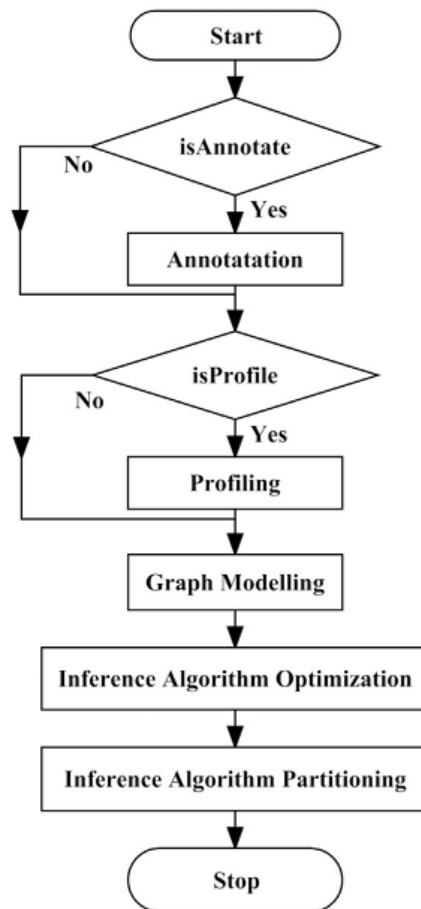


Figure 2-9 A Graph-based code partitioning scheme in MCC [49]

Applications are difficult to comprehend and represent; applications are modelled using a Directed Acyclic Graph (DAG). Fig. 2-9 discussed the graph -based code partitioning scheme. Vertices and edges are the two components of a graph, and they represent the various parameters of an application. [5]. The vertex represents the computational cost, while the edge represents the communication cost. The partitioning strategy aims to partition the code in different segments where minimum possible communication holds between the nodes. In MCC, during offloading, the application is partitioned either statically or dynamically [45] [46] [47] [48].

Graph-based application partitioning models the mobile applications in the form of a Directed Acyclic Graph (DAG) as shown in the Fig. 2-10. The different elements of the graph, namely vertices and edges, represent the various parameter of the mobile application, such as computation cost, memory cost, granularity, data size, communication cost.

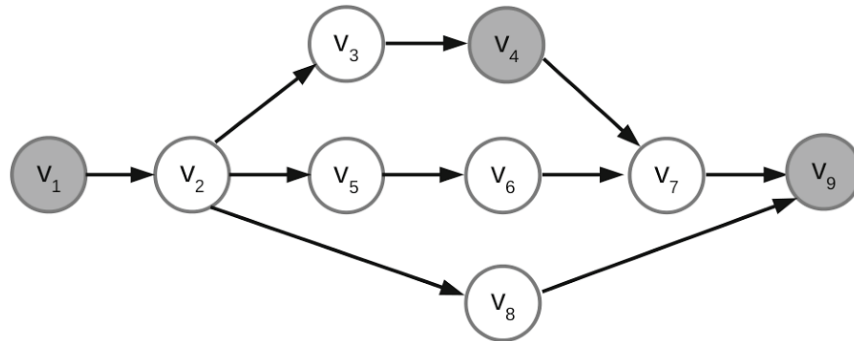


Figure 2-10 Directed Acyclic Graph (DAG) representing a mobile application

Graph-based techniques use a different coarsening algorithm to maximize code partitioning. In mobile cloud computing, a step-by-step process for graph partition. The first step is to determine whether the annotation is required. If the annotation is not present, the developer must manually add annotations to the code [49].

If the application has annotations, it will continue to check the profiler's output. The profiler will collect the information that the application requires. After completing all of the preceding phases, the execution moves on to the graph modelling phase. Graph modelling can benefit from programmer annotation and profiling results.

Following that, one or more algorithms are used to improve the graph model. Finally, inference methods such as solver are given the optimization result in order to decide and perform partitioning. Since our work is focused upon the code partitioning algorithm, the annotations and profiler output are assumed to be ideal.

Many real-life applications have N number of solutions in the solution space. If an application requires multiple solutions, optimization can be achieved by selecting the best alternative solution based on certain factors. The graph partitioning problem (GPP) is



concerned with partitioning the vertices in such a way that the edge cut value is reduced. When considering the GPP in the mobile cloud computing applications, the different vertices can be assumed at different granularity levels like classes, methods, threads, objects, or the application itself as a vertex.

In our problem, we are assuming vertex as method-level granularity. There are various techniques like local search optimization techniques like Simulated Annealing, Genetic Algorithm, Tabu Search, Random Walk, Neighbourhood Search, Swarm intelligence-based Ant Colony Optimization, and Particle Swarm Optimization which can be used for the optimization method. These optimization strategies are distinguished by the recursive application of the local search approach to the problem's solution. Graph partitioning is an NP-hard problem that aims to divide the graph's nodes such that there is a minimal inter-partition relationship that means minimum communication cost and execution cost on both sides of the client or cloud side. The multilevel graph partitioning method has emerged as highly effective graph partitioning in recent scientific studies. George Karypis et al. [50] have done proposed multilevel graph partitioning schemes which are effective in their manner. A multilevel approach can be used to divide Graph  $G$ . A multilevel algorithm's basic flow is simple to comprehend. The graph  $G$  is coarsened down to a few hundred vertices, a partition of this considerably smaller graph is produced, and the partition is then projected back towards the original.

Huaming Wu et al. [40] suggested the min-cost offloading partitioning (MCOP) algorithm, which partitions the code for local and remote execution. The partition model differentiates the offloadable and unaffordable tasks based on the consumption graph as presented in the fig. 2-11. The cost model is also proposed for calculating the overall cost of execution of the task. The MCOP algorithm provides a stably quadratic runtime complexity to decide the task execution locally or remotely.

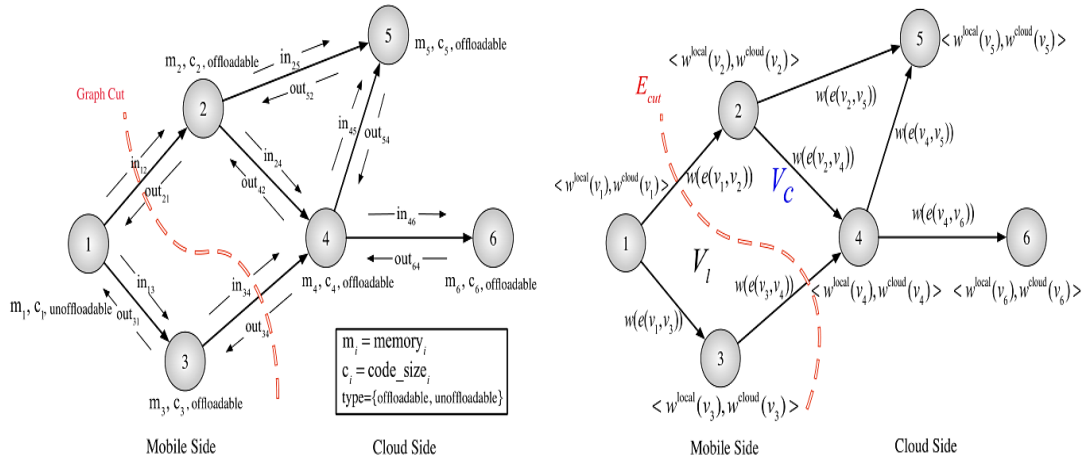


Figure 2-11 Consumption Graph and Weighted Consumption Graph [40]

### 2.3 Offloading decision making in MCC

Decision engines are considered a key component in the offloading framework [2], which decides to offload the task to the remote server based on the profiling process's available parameters.

Eduardo Cuervo et al. [39] proposed an energy-aware offloading mechanism “MAUI” and used code partitioning and profiler concept for offloading the task. It has used the optimization engine for decision making and provides an energy-efficient strategy at runtime. MAUI architecture is presented in fig. 2-12.

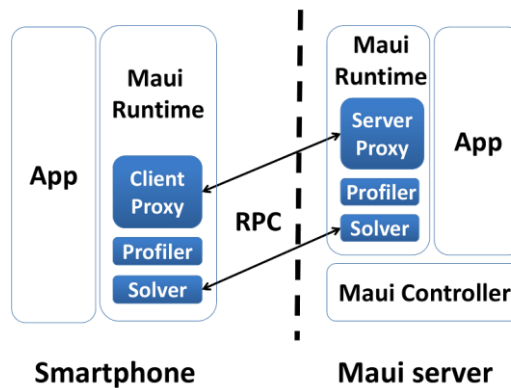


Figure 2-12 High-level view of MAUI's architecture [39]

Mohammed A. Hassan et al. [51] proposed the scheme POMAC for dynamic decision-making, which is transparent to the developer and compares the existing decision classifier for their approach.

B.Gao et al. [52] proposed an algorithm that takes total energy consumption and schedule length as an essential parameter for offloading. The authors have proposed two strategies- One for uploading the task from the mobile device to the server and the second on the server's side workflow engine. They have focused on autonomous decision-making ability, offload authorization, and task clustering.

B. Zhou et al. [53] suggested the "mCloud" offloading framework, which consists of a mobile device, proximate cloudlets, and a public cloud service, with the goal of improving the MCC service's availability and performance. Context-aware offloading decision provides decision at the runtime where to offload code and in which wireless medium. The framework also provides the cost estimation method for accessing time and energy.

M. Amoretti et al. [54] proposed a modelling and simulation framework MCC design and analysis, including energy efficiency, storage capacities, processing power, and data security. The proposed discrete event simulator is a useful tool for evaluating the parallel task's execution. The model is based on the Queueing Network (QN), which consists of two sub-networks, one for each type of queue. They have also proposed the offloading decision algorithm based upon the energy level. The offloading policy is formulated on the offloading probability parameter.

Mahbub E. Khoda et al. [55] have presented an intelligent computation offloading system for offloading code from a mobile device to a cloud server over a 5G network. They suggested a decision engine based on the Lagrange Multiplier, a non-linear optimization solver that increases application reaction time and reduces mobile device energy consumption.

Mendoza et al. [3] suggested a Python-based system in which offloading decisions are made based on cloudlet execution time, client device execution time, and mobile device to cloud transfer time.

Kosta et al. [4] proposed the framework for offloading work "Thinkair" over the cloud with

its application server concept and broadly discussed profiling. They have exploited the mobile device virtualization over the cloud and provide method-level computational offloading.

Kemp et al. [5] proposed the primitive android-based framework CUCKOO for computational offloading. The framework offers a simple programming model using remote method invocation and IPC mechanism for the job's local and remote execution. It has a robust runtime algorithm that determines whether code should be executed locally or remotely at runtime. Table 2-2 presents the comparative study of the offloading related work on the basis of energy (E) and performance (P) parameter and type of profiling, decision engine and application considered.

Table 2-2 Various decision-making techniques in MCC

Paper	Year	Contribution	E	P	Type of Profiling done	Decision engine addressed	Offloading application
Kosta et al. [18]	2012	Proposed a framework based upon three-component, i.e., application server, execution environment, and profiler	√	√	Hardware, Software, Network	Execution controller based upon time, energy, and cost	N-Queens problem, Virus scanning application, Image combiner, face detection application
Rudenkc et al. [56]	2012	CUCKOO-Worked for android based application. Presented intelligent offloading mechanism based upon IPC	√	√	Heuristic approach considered	Yes	Eyedentify (image-based app) and Photo-shoot (augment reality game-based app)
Cuervo et al. [39]	2010	Primarily worked on energy management. Provides high-level programming	√	√	Device, Program, Network	Yes	Face recognition

		architecture for remote execution based on RMI.					
Ma et al. [22]	2011	Propose and implemented computation migration technique stack-on-demand (SOD) into Java distributed runtime that migrates the light-weighted threads. It optimizes offloading by moving more specific data over the cloud.		✓	Data migration through JVM Tool Interface (JVMTI)	No	iPhone based application
Kovachev et al. [24]	2012	Proposed the middleware named mobile augmentation cloud services (MACS), which enables applications of android to offload the task from the mobile device to the cloud server	✓	✓	Profiling is done by MACS middleware	MACS form an optimization problem that decides for local or remote execution	Generally, android based application Solved the N-Queens problem, and face detection and face recognition
Gao et al. [52]	2012	Present a heuristic algorithm and provide a dynamic offloading solution that saves time and energy	✓	✓	Energy profile	Decision algorithm based upon time and energy	Type of workload not defined clearly
Hung et al. [25]	2012	Proposed a framework that profiles the data and decides dynamically to offload or not		✓	Framework integrated along with VPA tool	profile-based policy manager and profiling service	object recognition (OR)

						helps in decision making	
Yang et al. [26]	2012	Proposed a two-phase partitioning mechanism called Manageable Application Code Offloading (MACO) which divides the problem in UI based and computation based and executes them separately, one locally and the other remotely.	✓	✓	Network-based and done by automatic partitioning mechanism	Used the concept of decision-maker	web pages from commercial and online products
Xia et al. [57]	2013	The phone2cloud architecture, which includes a bandwidth and resource monitor as well as an execution time predictor, was proposed and built. For the framework, an offloading decision engine was also proposed.	✓	✓	Bandwidth and Execution time	Used offloading decision algorithm based on execution time and power consumption	Sort, Pathfinder (shortest path), and word count
Eom et al. [58]	2013	Presents an adaptive machine learning algorithm based on the founding from four different workloads and 19 distinct machine learning algorithms and four workloads and worked over the Android-	✓	✓	computation work, data size, bandwidth	Use machine learning-approach to decide whether to offload or local	Android-based applications

		based application for final testing of the algorithm					
Mtibaa et al. [28]	2013	Proposed environment mobile device cloud (MDC) in offloading happens in the cloud created by a group of the mobile device.	✓		Energy and time	Implemented own MDCloud to decide to offload on MDC, Cloud, or cloudlet	Android-based applications
Anastasopoulos et al. [30]	2014	Proposed multi-objective service provisioning scheme for energy-aware offloading with delay consideration	✓		Decision-based on energy consumption	Use the concept of network calculus	Not defined
Qian et al. [29]	2014	Proposed a framework that saves energy during computation offloading in Android apps	✓	✓	Execution time, Battery level, CPU and wireless connectivity, data size	Use the concept of Jade optimizer for making a decision	Image processing, Navigation application
Truong-Huu al [59]	2014	Dynamic opportunistic offloading algorithm has been proposed, which is based upon the Markov decision process		✓	Energy	Use MDP model for opportunistic offloading	Size reduction of photographs
Hyytiä et al. [60]	2015	Proposed a stochastic model for studying the dynamic offloading in mobile cloud computing	✓	✓	Energy, monetary cost, delay	Decision is based on a multi-queue model which captures the required features	Not available

Terefe et al. [31]	2015	Application can be executed on multi-site which uses discrete-time Markov chain (DTMC) to prototype fading wireless mobile channels	✓		Energy	Combination of static analysis and dynamic profiling leads to the formation of a mathematical model	Code intensive and data-intensive
Lee et al. [61]	2015	Proposed a mobility aware based offloading decision and use a second-order Markov model as a mobility model	✓	✓	Device, Program, Network	propose a probability-based prediction engine for taking an offloading decision	Considered dummy dataset
Flores et al. [7]	2015	Framework is proposed, which is offloading the application considering granularity at the method level. Java reflection is used along with the client-server model	✓	✓	memory, CPU, network bandwidth, cloud server capacity, and also size of the application	Information defined in a JSON schema is used to create the automated mechanism which profiles the code	NQueens problem
Zhang et al. [33]	2015	Proposed an MDP based dynamic offloading algorithm	✓	✓	Computation power and execution cost	Used stochastic modelling and dynamic optimization. Designed a dynamic algorithm for decision making	Face recognition
Wang et al. [62]	2015	Proposed adaptive application offloading model	✓	✓	Response time and energy	Dynamic application offloading policy	Real-time applications



		in the paper. Lyapunov optimization theory is used to propose the offloading decision algorithm				based on stochastic network optimization	
Neto et al. [63]	2016	Proposed and implemented a location-aware decision engine that is modelled upon execution time, energy, and bandwidth prediction	✓	✓	CPU and Bandwidth	Used location awareness and spline interpolation for modelling the decision engine	Face Detection Application
Khoda et al. [55]	2016	Over the 5G network, proposed an intelligent computation offloading system that takes decisions for code offloading from a mobile device to a cloud server.	✓	✓	Bandwidth, data size, cloud speed factor, server load condition	Used regression model, Lagrange multiplier for decision making	N-Queens application
Wu et al. [40]	2016	The min-cost offloading partitioning (MCOP) technique was proposed, which divides the code into sections for local and remote execution. Based on the consumption graph, the partition model distinguishes between offloadable and non-offloadable processes.	✓	✓	Program, network, and energy profiler	Weighted call graph is used for offloading decision	Face recognition application

Chen et al. [34]	2016	Proposed the concept of mobility aware-based caching and computational offloading in the 5G ultra-dense network.	✓		Channel bandwidth	Based on the mobility of the user, offloading strategy is decided	Random mobility nodes considered
Chen et al. [64]	2017	A framework is offered based on an application's design pattern, and an estimating model is offered that determines which cloud resource to offload.	✓	✓	Focused upon the network aspect considered bandwidth, round trip time, and server (cloud resource) response time	Based on the profiling information, a decision algorithm will select the optimal service for offloading	Gobang game (Interactive chess game) and face finder application
Rego et al. [65]	2017	Proposed an offloading decision scheme in which decision tree algorithm C4.5 is considered as a significant theme.	✓		Mainly considered the network conditions like data rate and round-trip time (RTT)	Decision to offload is taken based on decision tree	Matrix operation
Ko et al. [66]	2018	Decision-making is done using the formulation of the Markov decision process (MDP). It is formed considering the various types of wireless networks and device spatial and temporal conditions.	✓		Network conditions are considered for profiling purposes like device position in Wi-Fi AP, edge, and cloud	Value Iteration Algorithm is used for making an optimal decision in MDP formation	Considered dummy task of different complexities level

Ravi et al. [67]	2018	DBSCAN, along with KL divergence, has been used for identifying the offloading task. Further, the offloading algorithm is used for decision-making purposes.	✓		CPU cycles	Designed decision algorithm which considered CPU cycles and execution deadline based on probabilistic measures.	Chess game and Video editing application
Zhou et al. [68]	2019	Optimal solution for offloading on multi-edge servers has been explored using Markov approx. approach. Performed state transition over Markov chain considering different configurations	✓	✓	CPU cycles, uplink, and downlink data rates	Used Markov based approximation approach for decision making	Considered dummy task with different load scenarios
Qi et al. [69]	2019	Decision algorithm has been developed using the deep RL approach. Mobility models are also developed for decision-making in smart vehicles.	✓		Mobility factor, bandwidth, CPU cycle	Used A3C algorithm for decision making	Created own task for testing purpose
Misra et al. [70]	2019	Proposed and implemented three-tier architecture for offloading in the cloud, cloudlets, and nearby devices based upon auction theory.	✓	✓	Bandwidth, Response time, energy	Used the auction theory principle for decision making	Merge Sort, Knapsack problem, Matrix multiplication

Zhou et al. [53] suggested the "mCloud" offloading structure, which consists of mobile devices, cloudlets near the device, and a public cloud service, with the goal of improving the MCC service's availability and performance. The context-aware offloading decision provides a decision at the runtime where to offload code and wireless medium.

Manukumar et al. [71] proposed an enhanced particle swarm optimization algorithm approach for decision making and aimed to reduce the makespan of the offloading process and power consumption. A decision-making approach based on online machine learning and genetic algorithms was proposed by Xiaomin et al. [72]. It was also designed to conserve energy and time when using a mobile device.

Elhosuieny et al. [73] proposed a methodology based on non-linear polynomial regression, which helps in building the time-predicting model. It decided to offload based on bandwidth and predicted time to execute on the mobile device.

Shahidinejad et al. [74] proposed a decision-making scheme based on learning automata. It has improved the decision engine's execution time using the probability of the mobile device's events. Much work has been done on energy and performance during the offloading process, but the work on the accuracy of the decision engine classifier has not been addressed in most research.

## **2.4 Mobility mechanism in MCC**

Mobility in wireless networks refers to a node, Mobile Node (MN), changing its attachment point to the network while its communication to the network remains uninterrupted. The current requirement in wireless communication is to provide services to the consumer on a real-time basis. Various applications related to multimedia applications, healthcare, and gaming require ubiquitous services maintaining the high data rate and providing roaming services. The evolution of wireless communication can be seen in fig. 2-3, where it can be seen that the data rate gets increased in the past years, and mobility has taken its essential place in the communication system.

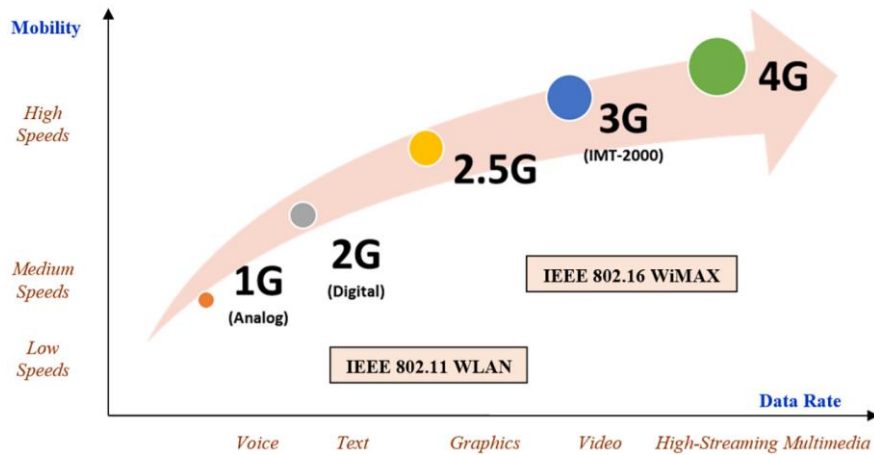


Figure 2-13 Evolution of Wireless communication [61]

Evolution of wireless communication is discussed in fig. 2-13. In the early 1980s, the foundation of 1G was laid down, which provided seamless connectivity in mobile voice services. The data bandwidth was around 2kbps and used the FDMA multiplexing. The base stations were deployed for the mobile users in the geographical area called a cell. Ideally, the hexagonal shape seems perfect for frequency reuse in the cellular network, as seen in fig. 2-14 but these cells overlap in reality.

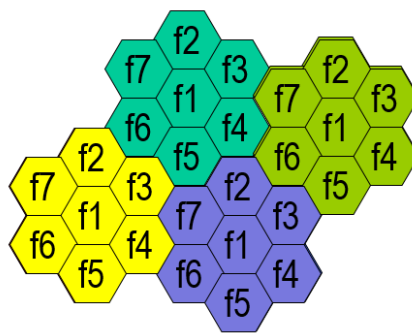


Figure 2-14 Frequency reuse in cellular cell

A base station is a communication system used to establish communication with the mobile device using radio waves. Each base station has a coverage area, and receivers and transmitters are mounted on the base station. Two radio channels are used for the

communication between mobile and base station a) control channel used for call setup, call initialization, and other control purpose b) forward channel used for transmitting information from base station to the mobile device.

### 2.4.1 Wireless communication technologies

The 1G mobile technology [75] has used analog signal that works amazing for voice communication but has challenges like limited capacity in terms of spectrum and limited scalability. 2G was introduced in the 1990s, providing more voice capacity using modulation techniques like TDMA and working on digital signals. Technologies like D-AMPS and GSM were based on the TDMA. The digital transmissions enable voice compression and very scalable technology. Still, an issue of signal interference was there in the 2G technology. The call drop was potentially high in the 2G network. It had started using CDMA technologies for voice communication. 2G does the communication by using both circuit and packet switching modes. In all these cellular networks, there are base station controller (BSC) and base transceiver stations (BTS). Each cell in the network has one BTS, and multiple BTS are controlled by one BSC. These BSC are connected to the mobile switching centre (MSC), which is linked to the PSTN network. The cellular network's basic design and its many components are depicted in fig. 2-15.

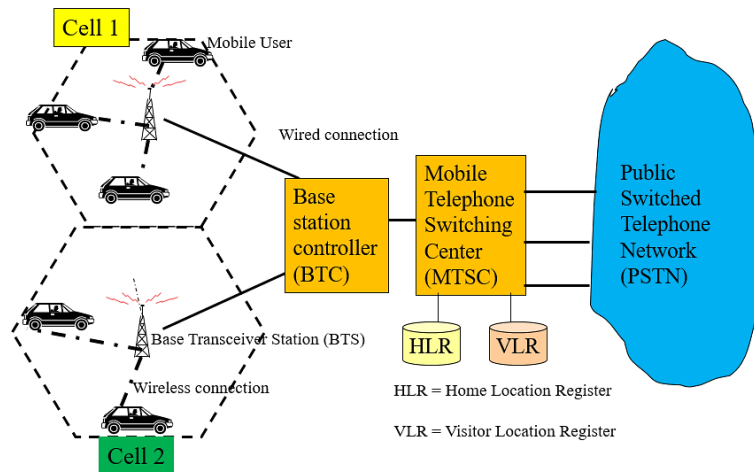


Figure 2-15 Basic structure of a cellular network [75]

In the early 2000s, the 3G technology, also known as the universal mobile telecommunication system (UMTS), was introduced with a high data rate of 2Mbps and a packet network method. It delivered high data rates, more capacity, and enhanced the experience of mobile broadband. It provided high-quality audio and video services using WCDMA technologies and supports both circuit and packet-switched for call and internet. UTRAN acts like the brain of the network. The nodeB is acting similar to the BTS of the 2G systems. Radio network controller (RNC) can be connected to many nodeB through the Iub interface. RNC is further connected to the PSTN and PDN network in the core network of the 3G systems. The 3G system works parallel with the conventional cellular system where the voice network works as the previous network setup, and the data network operates in parallel by serving the GPRS support node (SGSN) and gateway GPRS support node (GGSN) as defined in fig. 2-16.

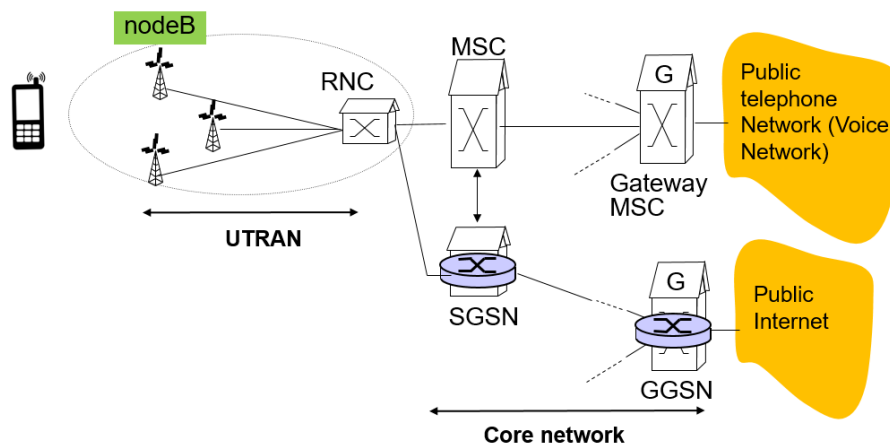


Figure 2-16 Basic structure of a 3G network

After the evolution of the 3G system, around the year 2010, the 4G technology was introduced with a data speed of 200 Mbps. It is a unified IP and seamless mixture of broadband technologies of LAN, WAN, and WLAN. It uses CDMA and internet technologies for core communication. The enodeB is similar to BTS and nodeB, whereas E-UTRAN acts as an interface for all coordination of user devices and backend core network. The EPC has two planes a) core plane which is used for controlling, managing, and monitoring the communication, and b) the user plane

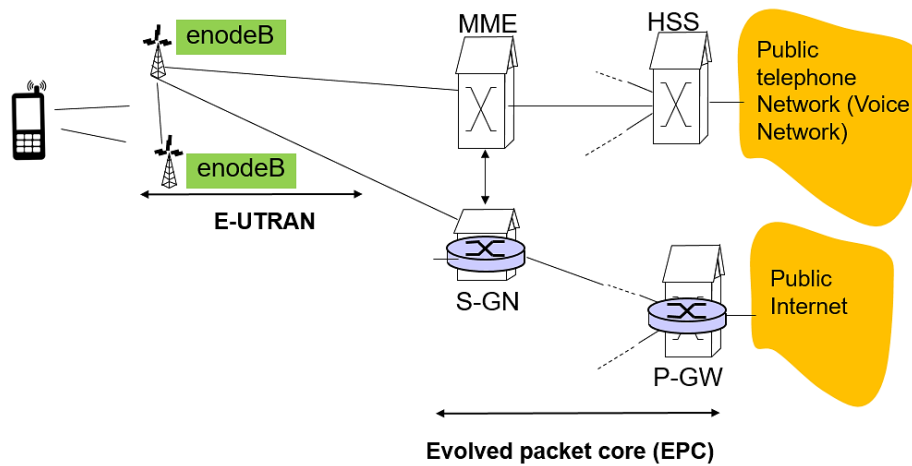


Figure 2-17 Basic structure of a 4G network

which is used for managing data communication, management, and evaluation purposes. The MME is a mobility management entity that takes care of mobility issues and authentication, and HSS is a home subscriber system that keeps the information about user records.

### 2.4.2 Handoff Management

Disconnection becomes a vital problem because of continuous mobility. Assume a device is connected to the cloud over a 4G/ 3G network or, in the future, 5G. Now, if the device goes to a location where the mobile network is unavailable, the cloud connection will be lost. A handoff occurs when a mobile device switches from one network to another. The seamless transitions among networks can be either horizontal or vertical.

#### Vertical and horizontal handoff

When a device moves from one network to another without changing the network type, then the process is called horizontal handoff [76], and if it changes the network, then it is called vertical handoff. Heterogeneous networks (Hetnets) have various types of features like data rates, received signal strength (RSS), network capacity, bandwidth, and coverage span. Fig. 2-18 depicts the horizontal and vertical handoff style.



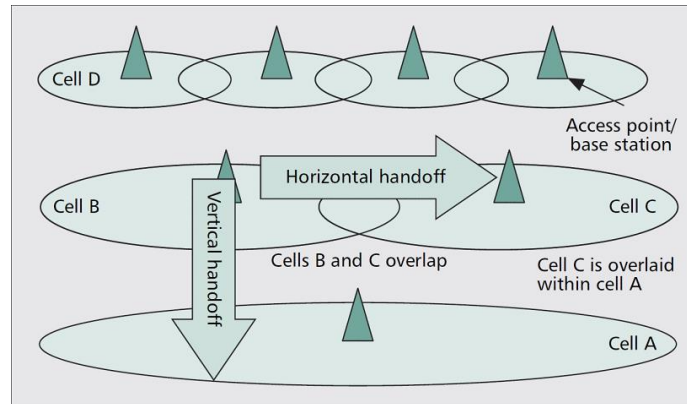


Figure 2-18 Horizontal and vertical handoffs [76]

The mobile device perceives these features and decides to select the best available network in its current location. Various reasons can lead to the process of handoff, like avoiding the call drops situation, when cell capacity to hold new calls is exhausted, the existence of channel interference, or when there is a change in user behaviour in mobility and speed.

### **Hard and Soft Handoff**

Hard handoff follows the "break before make" policy, and soft handoff follows the "make-before-break" connection. Hard handoff requires the user device connected to get a break before connecting to another station. It is implemented in FDMA and TDMA based devices. It is a cheaper strategy to implement, but the delay is mostly experienced while implementing this method. On the other hand, in soft handoff, the mobile device gets connected to more than one BTS during the same time. It achieves higher quality and low delay but is only supported by CDMA/ WCDMA mobile phones.

### **Network-controlled, Mobile-controlled and Mobile-assisted Handoff**

When the handoff decision is taken by the network after measuring the number of mobile stations in the cell, it is known as network-controlled handoff. It was taken initially in AMPS (advanced mobile phone system) and TACS (total access communication system). In a case where a mobile station measures the network parameters and handoff decision is

taken by the network, it is called mobile-assisted handoff. In a different scenario, when a mobile device measures the signal strength and interference level and takes the decision to handoff, then it is known as mobile-controlled handoff. The reaction time is much smaller than 0.1 seconds.

### Desirable handoff features

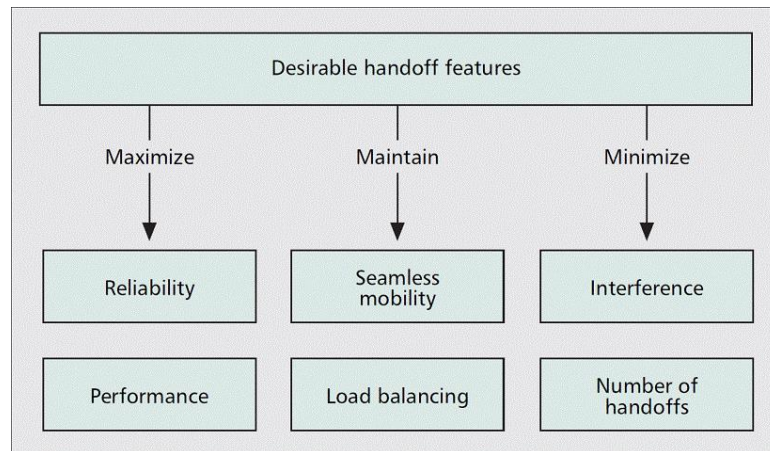


Figure 2-19 Desirable handoff features [76]

The handoff method aims to maximize reliability and performance. The methods must provide a base station having high signal quality and signal to noise ratio (SNR) and receive signal strength (RSS). The handoff technique must provide seamless mobility, uninterrupted services, and load balancing. It must aim to minimize the channel interference and number of handoffs as seen in the fig. 2-19.

### Vertical Handoff Criteria and Metrics

There are number of parameters that affect the handoff decision like received signal strength, network connection time, handoff latency, network load, power consumption, and velocity. These parameters depicted in the fig. 2-20 are often analysed while moving from one network to another network.

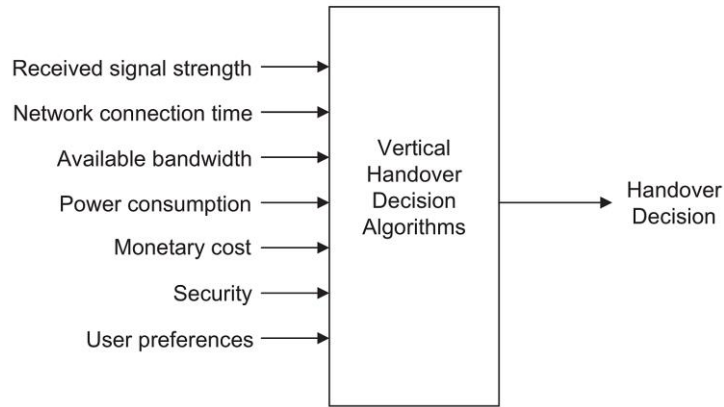


Figure 2-20 Parameters used for making VHD decisions

Kilho Lee et al. [61] have created a mobility model as seen in fig. 2-21 and the decision to offload will be made based on the pattern seen by the model. Based on user location data (Wi-Fi), the 2nd order Markov model is framed and trained with the mobility pattern of specific users. In the future, the offloading decision will be taken based on this model. A prediction engine is also proposed to decide whether to offload or not. However, the researcher leaves the future scope of the moving speed factor as it also affects the offloading performance.

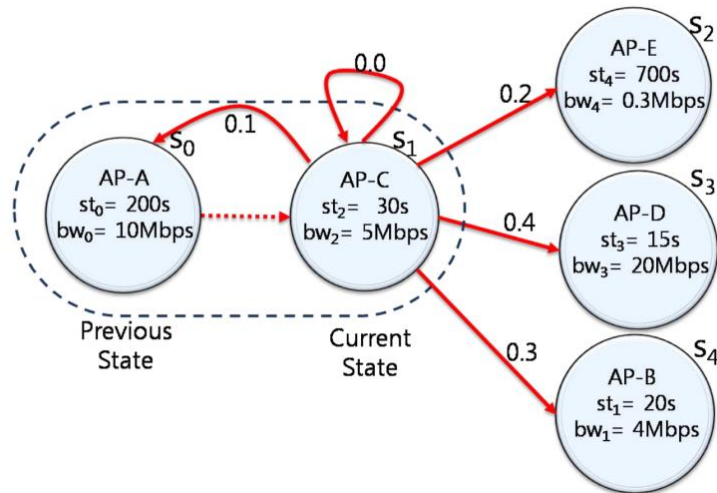


Figure 2-21 Mobility model [61]

Different solutions have been proposed by researchers that address mobility during computational offloading.  $M^2C^2$ [77] has proposed a multihoming mechanism as in fig. 2-22 where device probes for network and cloud network during mobility. The best cellular network and cloud network are selected based on the RSSI and cloud ranking process in this work.

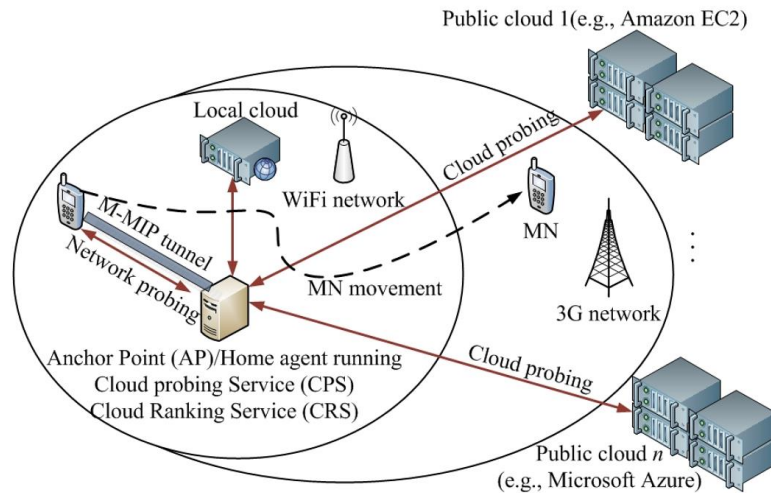


Figure 2-22  $M^2C^2$ : A Mobility Management Scheme for MCC [77]

Clonecloud [16], MAUI [39], ThinkAir [18], and Cuckoo [6] have proposed an effective and standard solution for computational offloading in mobile cloud computing but have not considered mobility as a factor in their work.

A handoff scheme in mobile cloud computing has been proposed by Q. Qi et al. [78], which focuses on saving the device power, but while offloading, the bandwidth must be above 4MB to avoid the handoff delay.

A. S. Alnezari et al. [79] presented the handoff mechanism and offloading strategy in mobile cloud computing using the fuzzy logic model approach and worked on 3G and WLAN environments. Q. Bani Hani et al. [80] presented a robust five-layer service-oriented architecture that can perform seamless handoff in WiMAX and helps in reducing the bandwidth and power consumption.

T. Ali et al. [81] and A. Sgora et al. [82] have presented the fuzzy approach based on multi-

criteria and multi-attribute handover decisions.

Z. Sanaei et al. [83] has discussed the challenges and issues faced in MCC when heterogeneity in the network is introduced. The complexity gets increases when the mobile device roams in 3G, Wi-Fi, and WiMAX networks.

D. Bhattacharjee et al. [84] and Tong Liu et. al. [85] discussed the user device mobility and the prediction of the location in the next movement. The mobility in mobile cloud computing is still an open area where much research is still required, and hence, mobility is incorporated in the computational offloading in this work.

## 2.5 Scheduling mechanism in MCC

This section provides the work done so far on the topic of MCC scheduling. As seen in fig. 2-23, once the task has been offloaded to a cloud server, its execution plan or schedule is another challenge on a virtual machine. Task scheduling is ordering a task and assigning a module to the server that can optimally manage the task. The scheduling algorithm must be optimally designed so that the task's timely execution can be achieved and starvation or deadlock-like conditions can be avoided.

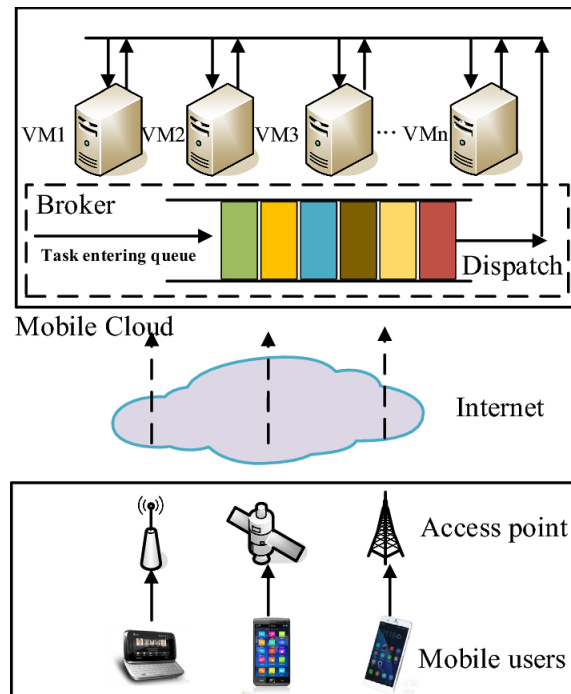


Figure 2-23 Task scheduling in mobile cloud computing [86]

A good scheduler can decrease the server's operational cost, improve resource utilization, and reduce the waiting time in the queue. Some of the current work done on the cloud task scheduling is presented below:

The mobile user offloads [86] the task to the cloud server using the wireless access point or cellular network. The required bandwidth by the device is provided by the access point and supports like signal strength so that mobile remains connected to it. When the offloaded task reaches the cloud server, they are arranged in a queue by the broker entity. Broker controls the task admission, check resources availability of CPU, storage, and memory in the form of virtual machines. After that, it allocates the task to the VMs for execution, as seen in figure 2-23. This is the basic architecture of task scheduling in mobile cloud computing.

Hsu Mon Kyi et al. [87] have proposed an algorithm on scheduling and resource allocation of virtual resources and virtual machines named Efficient Virtual Machines Scheduling Algorithm (MSA). The performance of the scheduling algorithm is evaluated using the stochastic Markov model. To present the concept, Eucalyptus architecture is introduced as a system model. The resource allocation decision model is based upon the continuous Markov chain model.

K. Jagannathan et al. [88] have presented the mathematical model on the buffer overflow in parallel queues. The study shows that the longest queue first scheduling policy has a superior queue overflow performance than queue blind policies. Several lemmas are presented in support of the theory presented in the paper. The study assumed that the system contains  $N$  parallel queues that are served by a single server. Time is allotted, and the server only handles one queue at a time.

Jilan Chen et al. [89] presented the weighted round-robin scheduling algorithm for task scheduling in a Hadoop framework. Since some tasks are light weighted, and some tasks is heavy weighted, the researcher proposed the algorithm to optimize the

H. Khojasteh et al. [90] have proposed a resource allocation mechanism over the cloud server using prioritization. The forked task has given top priority over the newly arrived task in the task queue, and in another case, the threshold is defined to control the priority.

The markovian multi-server queueing system analyzes the performance of both mechanisms. The impact of task arrival rate, service time, and the quantity of offloaded jobs on the performance indicators for both priority systems was also examined.

X. Nan et al. [91] have investigated the QoS and resource cost of the multimedia service provider by the proposed queueing model and optimization methods. The researchers have analyzed various types of scenarios like single server scenarios and multi-server scenarios. Using the window azure platform, various simulations are made in the study.

H. Eom et al. [58] focused on offloading scheduling and using machine learning-based techniques to improve the process. Their research looked at 19 distinct machine learning methods and four different workloads.

Yuan Zhang et al. [92] has proposed the joint resource scheduling and code partitioning for effectively allocating cloudlet to multiple cloud users. They have proposed a code partitioning algorithm based on the call tree. Hsu Mon Kyi et al.

Efficient Virtual Machines Scheduling Technique is a suggested algorithm [87] for scheduling and resource allocation of virtual resources and virtual machines (EVMSA). The performance of the scheduling algorithm is evaluated using the stochastic Markov model. Eucalyptus architecture is introduced as a system model. The resource allocation decision model is based upon the continuous Markov chain model.

X. Wei et al. [93] have proposed the extended cloudlet approach for supporting local mobile cloud. They have presented a hybrid PSO approach and optimized the profit and energy consumption during scheduling.

M. Nir et al. [94] have presented a task scheduler model that optimizes mobile cloud computing's energy function.

X. Lin et al. [95] proposed a scheduling scheme based on dynamic voltage and frequency scaling and has optimized the application makespan and reduce energy consumption.

Table 2-3 presents the various task scheduling schemes, specifically in the mobile cloud computing framework.

Table 2-3 Task scheduling schemes in MCC framework

Techniques and Work Done	Year	Type of Problem	Objective function	Framework	Environment
HACAS [93]	2013	Application scheduling	Profit and Energy consumption	MCC	Simulation
TSPCCE [94]	2014	Task scheduling	Energy	MCC	IBM's linear programming solver
MCC task scheduling algorithm[95]	2014	Task scheduling with DVFS	Energy and Time	MCC	MATLAB
LARAC algorithm [96]	2015	Task scheduling with DVFS	Energy and Time Deadline	MCC	Simulation
eDors [97]	2016	Dynamic scheduling and energy-efficient offloading	Energy and completion time	MCC	Simulation
MCF-DF [98]	2016	Task admission and scheduling	Admission rate and execution cost	MEC	Python
HCOA[99]	2017	Task offloading and scheduling	Energy	MCC	Simulation
CMSACO [100]	2017	Multi-Task offloading	Profit and completion time	MCC	Simulation
TSRA[101]	2017	Resource allocation and scheduling	Delay	MEC	Simulation



COPE [102]	2017	Task scheduling	Energy, Price of Cloud service provider, Delay	MCC	Thinkair based simulation
DAA [103]	2018	Task scheduling	Makespan	MEC	Simulation
GABTS [104]	2018	Task offloading and scheduling	Energy, response time, deadline, and cost	MCC	C++
OAOA [105]	2019	Stochastic approach for task scheduling	Energy and QoS	MCC	Simulation
Application-aware [106]	2019	Task Scheduling	Latency	MEC	iFogSim
MWSM [107]	2019	Workflow scheduling	Latency, Energy, and Cost	MCC	Simulation
RCTSP0 [108]	2020	Task scheduling	Makespan, Reliability, and Load	MEC	Cloudsim
EBCO-TS [109]	2020	Task scheduling	Makespan and energy	MCC	Cloudsim
ADO-MTS [110]	2020	Task scheduling	Makespan, Resource utilization, and Energy	MCC	Cloudsim

## 2.6 Research Gaps and Challenges

This section presents the challenges in mobile cloud computing and research gaps that are found during literature review. When speaking about the partitioning, the issue of synchronization is still a problem. The compute-intensive part is required to be executed on a cloud server, and it becomes essential that the mobile device remains synchronized

with the cloud for offloaded tasks. If a deadlock occurs on the cloud server, it can hinder a mobile application's working. The granularity of the application needs to be decided appropriately as offloading objects, classes, or methods creates a different type of overheads. Other observations in code partitioning were:

- There has been various research about application partitioning, but very few researchers have considered dynamic partitioning during offloading in mobile cloud computing.
- It was found that schemes were not adaptive in terms of network bandwidth, energy consumption, and task size.
- It was also observed that minimum communication should exist between the local and remote components of the application, but very less work had been done in this area and the edge-cut concept was not explored upon.

Research Gaps observed on the basis of literature review in the decision engine were:

- Offloading faces significant challenges on the front of energy consumption and performance, and there is still a scope of improvement that needs to be addressed.
- Lot of research has been done on energy and performance, but limited work was found on the accuracy of the decision engine of the offloading process.

Research gaps observed in mobility management during offloading are:

- Based upon the literature review, it was found very few researchers had worked upon mobility during offloading. Mobile device normally roams with the users. It has not been considered as factor during the mobility scheme.
- Handoff of the mobile device was not explored much in the research of mobile cloud computing which can affect the device connectivity with the cloud server.

The mobile device may roam from one position to another. The mobile applications running on devices use cloud services and move in the heterogeneous cellular network. The handoff mechanism must be smooth enough so that connectivity cannot be lost with the cloud server. The techniques must be developed to reduce the mobile device's energy conservation while roaming from one base station to another.

In the mobile cloud context, task scheduling still has a lot of room for improvement. The virtual machine must schedule the mobile task that was offloaded to the cloud server. Traditional methods focused on the task's execution time on the cloud server.

- Energy consumption is a big challenge on the cloud servers also as a number of the task are growing tremendously.
- The scheduling scheme must focus on the bandwidth, CPU utilization, and memory of the physical machines, where tasks are allocated to the virtual machines by the brokers.

## **2.7 Summary**

This chapter presents the literature review of the various task partitioning schemes utilized in designing the code partitioning scheme for mobile cloud computing. Literature has been explored, and limitations of the various approaches are found in the study. It also provides a literature review on decision engines. Decision engines decide when to offload based on various inputs provided by the system profiling. The detailed tabular form of the comparison is presented where the decision engines are presented specifying the energy and performance criteria of the techniques. Mobility feature is also explored, and focus on various cellular technologies has been placed. Handoff mechanisms are discussed with respect to mobility management. The scheduling schemes in mobile cloud computing have been presented in the last phase of the chapter, where different optimization parameters are also presented. The various research gap and challenges are discussed at last of the chapter.

---

## CHAPTER 3

# CODE PARTITIONING DURING COMPUTATIONAL OFFLOADING IN MCC

---

### 3.1 Introduction

Computational offloading is emerging as a popular field in mobile cloud computing (MCC). Modern applications are power and compute-intensive, leading to energy, storage, and processing issues in mobile devices. Using the offloading concept, a mobile device can offload its computation to the cloud servers and receive back the device's results.

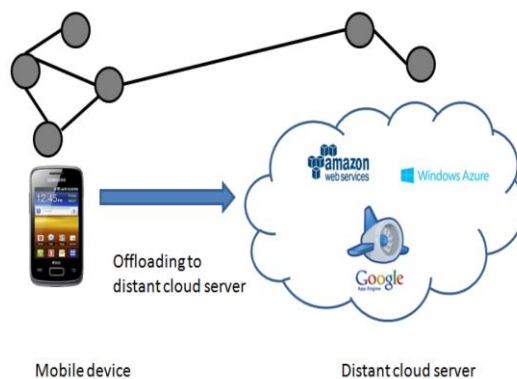


Figure 3-1 Partitioned component offloaded on cloud sever

An important question that arises in the offloading scenario is which part of the application needs to be offloaded remotely. In order to identify that, the application needs to be partitioned. In this work, the graph partitioning approach is considered based upon the spectral graph partitioning with the Kernighan Lin algorithm. Experimental results show that the proposed approach performs optimally in partitioning the application. The proposed technique gave better results than the existing techniques in terms of edge cut,

which is less, concluding minimum communication cost among components and saving energy of the mobile device.

Graphs are usually used as simplification by researchers while displaying application problems. One of the significant operations in graph theory is graph cutting. Other fundamental operations like traversal, flows, trees, and path are used in many scientific problems. Graph partitioning [111] is used to solve the complex problem as assuming an application as a graph reduces the complexity of manifolds.

Scientific problems lie in VLSI design, social network analysis, image analysis, and DNA mapping are solved using graph partitioning. Graph partitioning aims to divide the vertices into a certain number of groups where the nodes in one group are strongly connected while having a minimum connection with the other group. The two types of graph partitioning are constrained and unconstrained partitioning. The partitions of the same size are known as a constrained partition, while the partitions of unconstrained are of different sizes. Graph partitioning is used to minimize the computational load on either side of the partition and reduce the communication cost in various scientific simulations.

A Graphs  $G = (V, E)$  is the data structures of non-linear type and consist of vertices or nodes  $V$  and collection of edges  $e = \{x, y\}$  between pairs of vertices. The number of vertices  $n$  in the graph is represented as  $|V(G)|$  and the number of edges is represented as  $|E(G)|$ .

A graph  $H = (W, F)$ , where  $W \subseteq V$  and  $F \subseteq E$ , is a subgraph of a graph  $G = (V, E)$ .

The simple graph with vertex set  $V_1 \cup V_2$  and edge set  $E_1 \cup E_2$  defined by  $G_1 \cup G_2$  is the union of two simple graphs,  $G_1 = (V_1, E_1)$  and  $G_2 = (V_2, E_2)$ .

Matching  $M$  is a subset of the set  $E$  of edges of the graph  $G = (V, E)$  in which no two edges are incident with the same vertex. The vertex of the endpoint is matching is said to be matched; otherwise, it is known as unmatched.

Code partitioning is the foremost important task in computation offloading. It aims to define the application's components, which can be offloaded to the server or run locally on a mobile device. Specific components like GUI-based code and codes that need to be secured from different attacks are intentionally made to run locally to run graphics smoothly and mitigate the risk of attacks. The developer can annotate the component

@remote and @local to classify the code for offloading.

The nodes represent the computation points, and the edges depict the communication between two nodes. The graph partitioning problem is an NP-hard problem which makes it more appealing for many scientific problems.

### 3.2 Code partitioning using a graph model

Applications are complex to understand and simplify; applications are modelled using its call graph, a Directed Acyclic Graph (DAG) which can be seen in fig. 3-2.

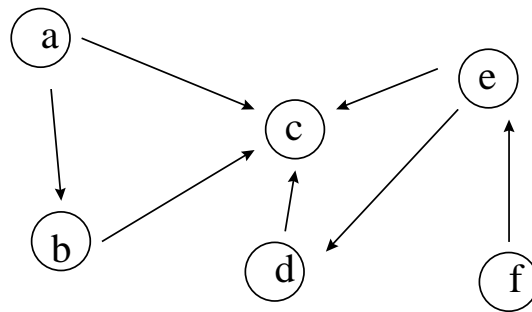


Figure 3-2 Directed Acyclic Graph

The two components of the graph are vertices and edges representing the different parameters of an application [55]. The vertex represents the computational cost, while the edge represents the communication cost. The partitioning strategy aims to partition the code in different segments where minimum possible communication holds between the nodes. In MCC, during offloading, the application is partitioned either statically or dynamically. The set of nodes is offloaded on the cloud for computation.



Figure 3-3 Different types of graph topologies

Fig. 3-3 represents various type of graph topologies where a) Node topology b) Linear topology c) Tree or Mesh topology are represented respectively. There can be a different level of granularity of the application, i.e., thread-level, method level, class level, or application level.

The graph can be represented with different topologies, i.e., either a complete graph as a node, a linear chain, tree, or a mesh. The mobile application can be signified by an array of fine-grained tasks in a linear chain, where the task can be executed either on the mobile device or offloaded to the cloud for execution.

### 3.3 Multi-level graph partitioning

Graph Partitioning is an NP-hard problem [48], and to achieve the optimal solutions of the problem, heuristic-based methods have been formulated. The goal of each heuristic method is to achieve the smallest possible cut for the two sides. In the scenario, the objective of the partitioning is to divide the compute-intensive part in the client and cloud or server-side. A multi-level technique is a series of smaller graphs that are combined to form a larger graph. The smallest of these graphs is used for the initial bisection. Finally, the graph is uncoarse, and each of the coarse graphs undergoes partition refinement. Vertices are given a weight that is proportional to their function. Weights are assigned to edges based on the amount of data that must be transmitted.

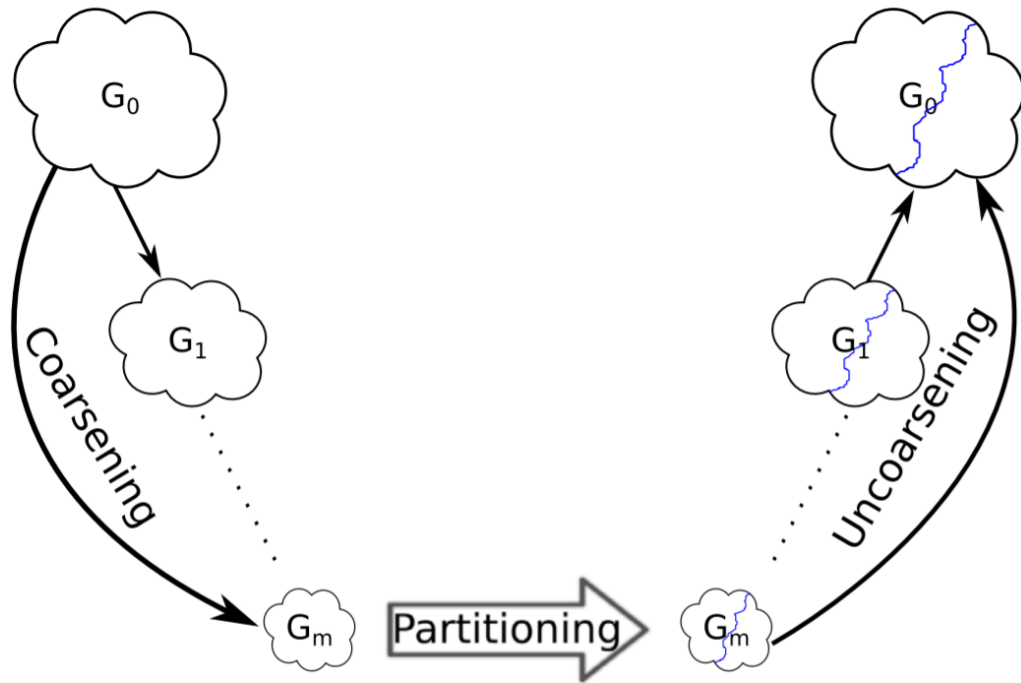


Figure 3-4 Phases of graph partitioning

Fig. 3-4 presents the three phases of multi-level graph partitioning [50][112][113] in which a graph is partitioned. These phases are coarsening phase, partitioning phase, and the last is refinement phase. The first phase is defined as coarsening phase, where the graph is converted into a sequence of smaller graphs using the concept of matching. The coarsening phase reduces the complexity of the graph up to a large extent. The coarsening leads to the edge contradiction where two connected vertices joined with an edge are merged into one vertex. The weight of the two vertexes is added into one, whereas the weight of the edges remains intact as earlier one. The various matching techniques used in the graph partitioning are random matching, heavy edge matching, heaviest edge matching, and zero-edge matching. After the coarsening phase, the next task is partitioning phase where the target is to have minimum edge-cut bisection that will divide the graph into two parts. Dividing the coarsened graph will lead the original graph into two partitions with less complexity. The last phase is the refinement phase which make the partitions more stable and improved version of the results achieved in the previous phase.



### 3.4 Problem statement

The code partitioning in mobile cloud computing aims to partition the code for offloading purposes, ensuring that there should be minimum interaction among the client and server (cloud). Resource constraint device often needs a solution to reduce the computational complexity of its application; the developed approach should focus on the least communication among the partitions of the application. The computation is divided among the local mobile device and remote cloud server to ease the mobile device's processing.

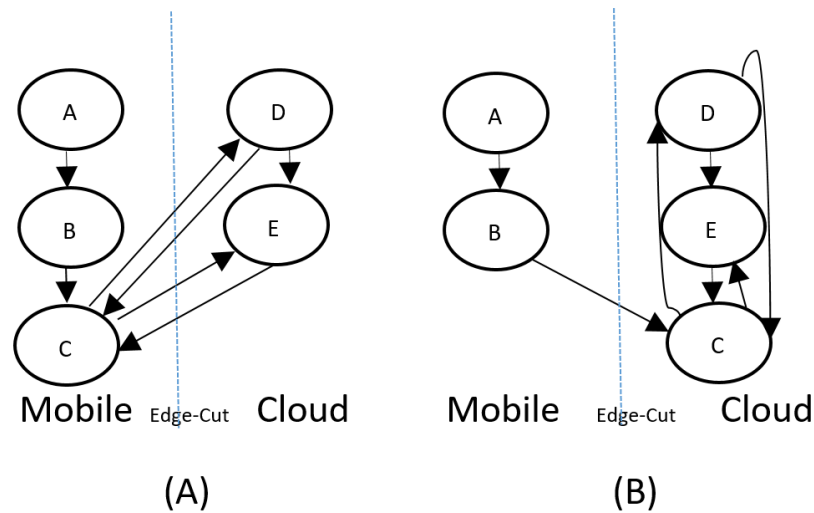


Figure 3-5 Scenario representing communication-based on edge-cut

Fig. 3-5 depicts two scenarios a) Communication between mobile and cloud before stable partition b) Communication between mobile and cloud after stable partition Data dependencies inside a computation are commonly described using graphs. The graph  $G = (V, E)$  consists of vertices  $V = \{V_1, V_2, \dots, V_n\}$  and edges so that it is partitioned into smaller components with specific computation tasks. The vertices and edges are weighted, i.e., the computation and communication costs are included in the problem. The partition of a graph is a process of dividing the task into subtask where  $P = P_1 \cup P_2 \cup \dots \cup P_k$  such that the application seems to be balanced partitioned. For example, the graph is coarsened with heavy edge matching, partitioned with spectral partitioning and finally refined by

kernighan lin algorithm. Assuming a node F along with node D and E, the initial partition of components created by spectral partitioning is fig. 3-5 (A). The refinement can be done by kernighan lin algorithm as seen in fig. 3-5 (B) where node C is swapped with node F to other side reducing the number of edge-cuts. Thus, final swapping of F and C node by KL algorithm will reduce cost and edge-cuts.

### 3.5 Algorithm description

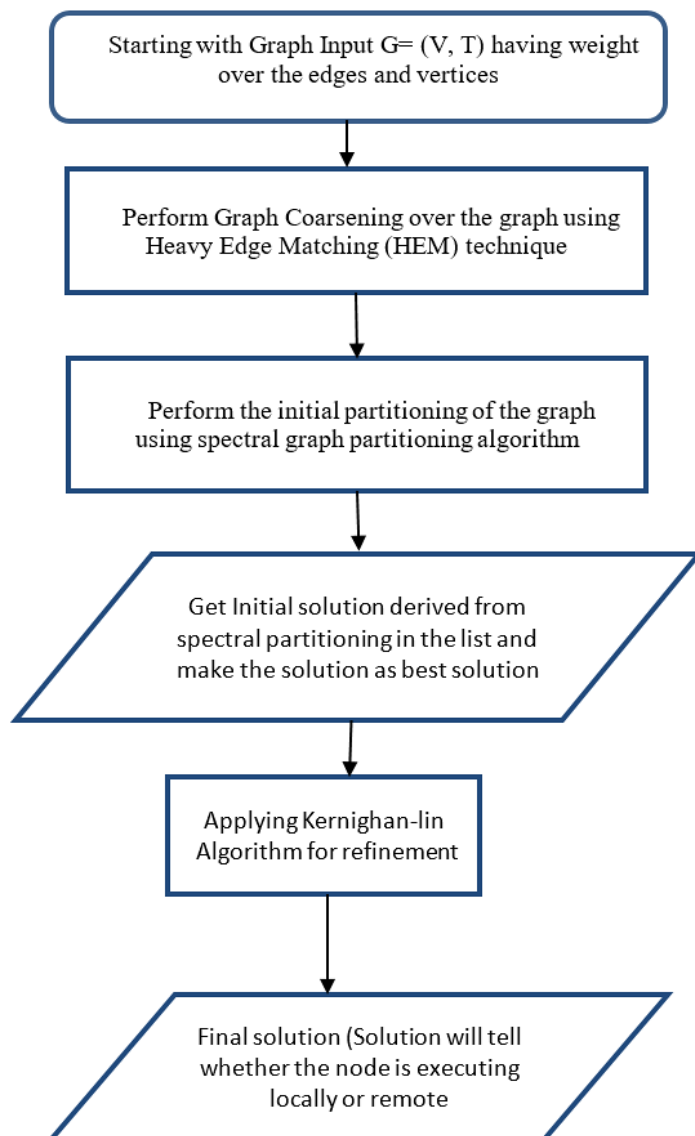


Figure 3-6 Flowchart representation of the proposed approach

For balanced graph partitioning, a multi-level hybrid technique is proposed in this thesis work. Although perfect partitioning is a challenging task, it can be achieved by optimizing the partitions at different levels. In fig.3-6, a graph is initially taken as an input where each node represents some class, method, or object based on the granularity. In this scenario, method-level granularity is considered. The graph contains many nodes at the initial level. To increase the effectiveness of the partitioning, the graph is coarsened using a matching concept. Heavy edge matching is performed on the graph initially, and after coarsening, the initial partitioning is applied by using spectral graph partitioning. During the initial partitioning, it is desired that a minimum edge cut among nodes must be achieved along with the optimal partitions of the component. The spectral graph partitioning algorithm creates balanced partitions, but the Kernighan Lin algorithm is further applied during the refinement phase for achieving minimum edge cut. The proposed approach has considered three different algorithms- Heavy edge matching for coarsening purpose, Spectral graph partitioning for initial partition, and Kernighan Lin algorithm to refine the partition results set.

The step by step phases of the proposed approach are discussed below:

### **Phase1: Coarsening Phase:**

The original graph  $G_0$  is condensed into a series of smaller graphs  $G_1, G_2, \dots, G_n$  such that the number of a vertex in the initial graph is reduced to a small number of vertexes  $|V_0| > |V_1| > |V_2| > \dots > |V_n|$ . The coarsening process results in the formation of graphs with a reduced number of vertices and edges [40]. By collapsing the edge, the weight of the two vertices gets summed up, which are connected by the collapsed edge. The process of coarsening is achieved by the matching process. Various matching techniques are used, like random matching, maximal matching, and heavy edge matching (HEM).

---

**Algorithm 3.1** Heavy edge matching

---

*HEM aims to achieve the minimized cut by finding maximal matching*

*//Input:  $G$ : the graph at state  $j$  with corresponding Edges  $E$  and Vertices  $V$ , i.e.*

*$G_i(V_i, E_i)$*

*$w$ : Edge weight and Vertex weight in the graph*

*$x, y$ : vertices which will be merged after coarsening*

*//Output: new graph  $G = (V, E)$  after merging nodes.*

1. *For all nodes  $v \in V$*
  2. *If  $v$  does not belong to  $\{x, y\}$       // a random vertex is selected*
    - a. *Then*
    - b.  $w\{e(x \cup y, v)\} = w\{e(x, v)\} + w\{e(y, v)\}$   
*// addition of weight (heavy edge matching)*
  3.  $E_i \leftarrow E \setminus \{e(x \cup y, v)\}$
  4. *End if*
  5.  $E \leftarrow E_i - \{e(a, v), e(b, v)\}$       // removing all edges from  $E$
  6. *End*
  7. *For  $V \leftarrow V_i - v(x, y)$*
  8. *Return  $G(V, E)$*
- 

**Phase 2: Partitioning Phase:**

After coarsening, the next phase is to apply the partitioning strategy over the coarsened graph [114]. It aims to partition the graphs into a bisectional graph or more based on the problem's requirement. In reference to mobile cloud computing, the partition can be bisectional, i.e., the node running locally or remotely.

---

**Algorithm 3.2** Spectral approach for partitioning

---

*//Input: a weighted connected graph  $G = (V, E)$*

*//Output: a partitioned graphs  $G1 = (V1, E1), G2 = (V2, E2)$*

1. *Construct Laplacian matrix  $LM$  and then compute the eigenvector  $ev$ .*
  2. *Explore the median of  $ev$*
  3. *LOOP process*
  4. *For each graph node  $ni \in G$*
  5. *if  $ev(ni) \leq \text{median}$*
  6. *move node  $ni$  in  $P1$*
  7. *else*
  8. *move node  $ni$  in  $P2$*
  9. *If  $|V1| - |V2| > 1$  transfer some nodes from  $P1$  to  $P2$  having equal median so that to equate the difference among two vertices count.*
  10. *Let  $P_s$  represent the collection of vertices that are adjacent to  $P2$  in  $P1$ .*
  11. *Let  $P_t$  represents the collection of vertices that are adjacent to  $P1$  in  $P2$ .*
  12. *Place edge separator  $E_s$  which is the set of edges of  $G$  with one point in  $P_s$  and the second in  $P_t$ .*
  13. *Let  $E1$  represents the collection of edges whose both end vertices lies in  $P1$ .*
  14. *Let  $E2$  represents the collection of edges whose both end vertices lies in  $P2$ . Build up the graphs  $G1 = (V1, E1), G2 = (V2, E2)$*
  15. *End*
-

---

**Algorithm 3.3** Refinement phase

---

*//Input: a graph  $G1 = (V1, E1)$ ,  $G2 = (V2, E2)$*

*//Output: improved and refined graphs  $G(V, E)$  define a refined nodes partition into sets  $A$  and  $B$*

1. *Best partition  $\leftarrow$  Current partition*
  2. *Do* *//computing initial gain*
  3.  *$\forall$  nodes  $a \in A$  and  $\forall$  nodes  $b \in B$ , compute the  $D$  value*
  4. *Let  $EL1$ ,  $EL2$ , and  $EL3$  be the empty lists*
  5. *LOOP process*
  6. *For each node  $n1$  of  $G$  to  $|V|/2$*
  7. *Examine  $a$  from set  $A$  and  $b$  from set  $B$  in the following way:*
  8.  *$g = D[a] + D[b] - 2*c(a, b)$  is maximal*
  9. *In this pass, discard  $a$  and  $b$ .*
  10. *Append  $g$  to  $EL1$ ,  $a$  to  $EL2$ , and  $b$  to  $EL3$*
  11. *Compute updated value of  $D$  for elements  $A = A \setminus a$  and  $B = B \setminus b$*
  12. *End for*
  13. *Calculate the value of  $k$  that maximises  $g_{max}$ , the sum of  $g_{max}$ .*
  14.  *$EL[1], \dots, EL[k]$*
  15. *If ( $g_{max} > 0$ ) then*
    - a. *Exchange  $av[1], av[2], \dots, av[k]$  with  $bv[1], bv[2], \dots, bv[k]$*
    - b. *Until ( $g_{max} \leq 0$ )*
  16. *Return  $G(V, E)$*
- 

The partitioned graph  $G1 = (V1, E1)$ ,  $G2 = (V2, E2)$  is gradually refined further with the Kernighan Lin partitioning strategy to improve the quality of partition.

## 3.6 Performance Evaluation

### 3.6.1 Experimental Setup

The approach has been implemented as a partitioning strategy in the Chaco simulator [115], which is primarily written in the C language. The work has been performed on a device having configuration Intel (R) core i3 CPU M330 @ 2.12 GHz processor, 4 GB of RAM, and Ubuntu 14 Operating System. The performance of the partitioning method is evaluated with respect to specific parameters like the execution time, data transferred, and energy consumption which is embedded in the input graph as weighted vertex and weighted edges. The evaluation has been conducted to investigate the effectiveness of the partitioning method in terms of edge cuts. A lower value of edge cuts reflects the minimum communication between the partition P1 and P2. The behavior of the proposed solution is also compared with the random partitioning and multi-level KL algorithm.

For each approach, the execution time and energy consumption are chosen at random from 100ms to 500ms and 1J to 20J, respectively, using the uniform distribution. These execution time limits are obtained from the range defined in the actual android application's trace log file [116] and the energy model for offloading framework in run time [117] [118]. For each method, these assumptions are reasonable as there is no correlation between energy consumption and execution time. The size of data that moves during offloading is assumed to be in the range of 50KB -500KB [29]. During offloading, the data travels between the various methods for complete execution of the application.

For the evaluation of the proposed technique, different node size graphs are generated. The weight of the nodes [119] are assumed to be the execution time of the node or method, and the weight of the edges are calculated based on the given formula

$$T = \text{Data Size} / \text{Bandwidth} \quad (3.1)$$

To reduce the complexity of the evaluation, the different dependencies between the software components are assumed to be the weighted nodes and edges. In a real scenario,

the call graph will depend upon the actual software design of the application and may be drawn with different topologies. In this work, the linear graph has been assumed for the evaluation of the proposed approach.

### 3.6.2 Results and Discussion

The comparative performance of the spectral approach with Kernighan Lin, Random partitioning, and Multi-level KL are discussed in Table 3-1.

Table 3-1 Comparison of graph partitioning techniques with proposed technique

Node count	Multilevel KL	Random		Spectral	
		Without KL	With KL	Without KL	With KL
<b>10</b>	1.5	15.42	1.5	1.5	<b>1.5</b>
<b>20</b>	1.5	26.04	1.5	4.5	<b>1.5</b>
<b>30</b>	1.72	37.21	1.72	6.6	<b>1.72</b>
<b>40</b>	1.35	40.51	3.1	8.45	<b>1.35</b>
<b>50</b>	4.58	52.16	3.39	13.12	<b>4.58</b>
<b>60</b>	2.41	64.7	3.03	3.02	<b>2.21</b>
<b>70</b>	3.33	99.43	3.88	9.34	<b>2.06</b>
<b>80</b>	2.91	78.12	5.32	7.26	<b>2.5</b>
<b>90</b>	4.81	108.19	4.91	8.06	<b>4.74</b>
<b>100</b>	10.12	110.12	11.15	10.5	<b>9.14</b>

The experiment was conducted on different node counts ranging from 10 to 100. The experimental results are achieved with a coarsening percentage of 50%. The graph results are compared with different approaches, i.e., spectral approach with and without KL, Random partitioning with KL and without KL, and proposed multi-level graph partitioning.



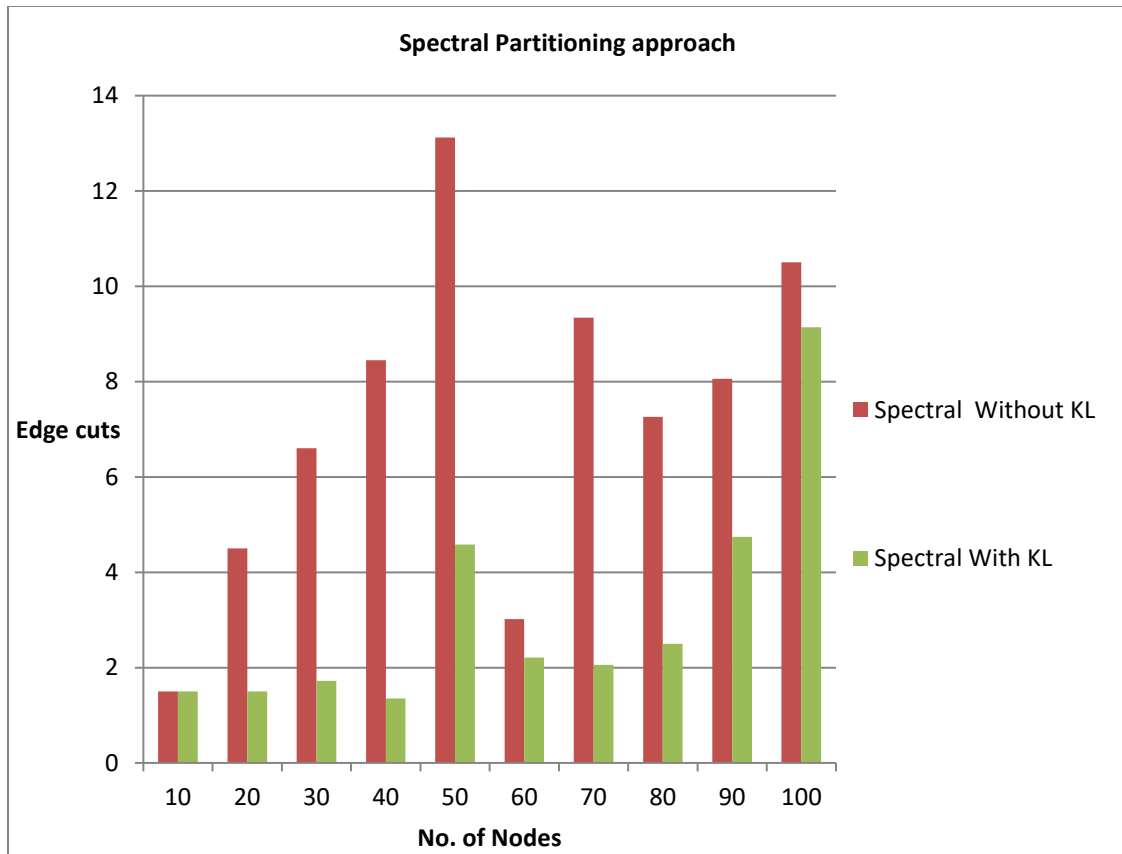


Figure 3-7 Graph represents the edge cuts results of spectral ands KL algorithm

The graph of fig. 3-7 represents the results of the spectral approach on the different graphs having a varying number of nodes from 10 to 100. The edge cut results of the spectral approach are plotted with a combination of the Kernighan Lin (KL) approach and without its combination. The results conclude that spectral partitioning performs better when combined with the Kernighan Lin approach. The number of edge cuts is significantly reduced in this combination. The Kernighan Lin improves the results during the refinement process.

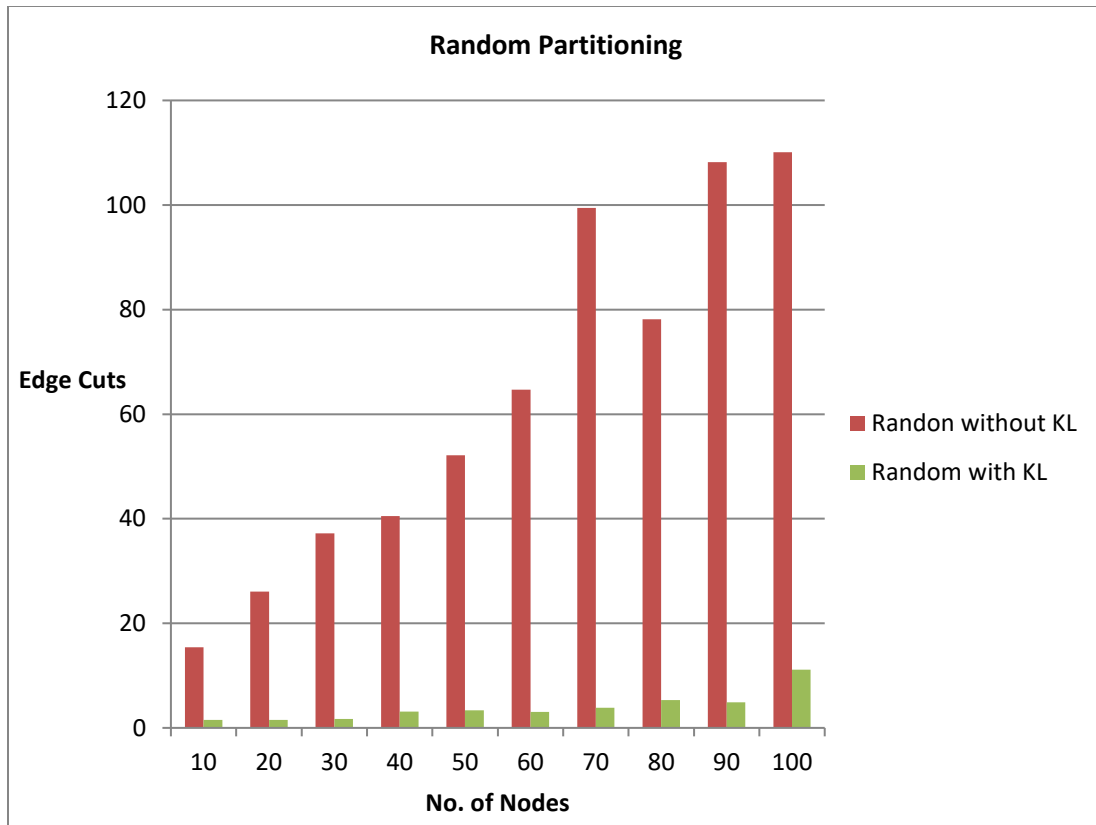


Figure 3-8 Graph represents the edge cuts results of the random

The graph of fig. 3-8 represents the results of a random approach on the different graphs having a varying number of nodes from 10 to 100. The edge cut results of the random approach are drawn with a combination of the Kernighan Lin (KL) approach and without its combination.

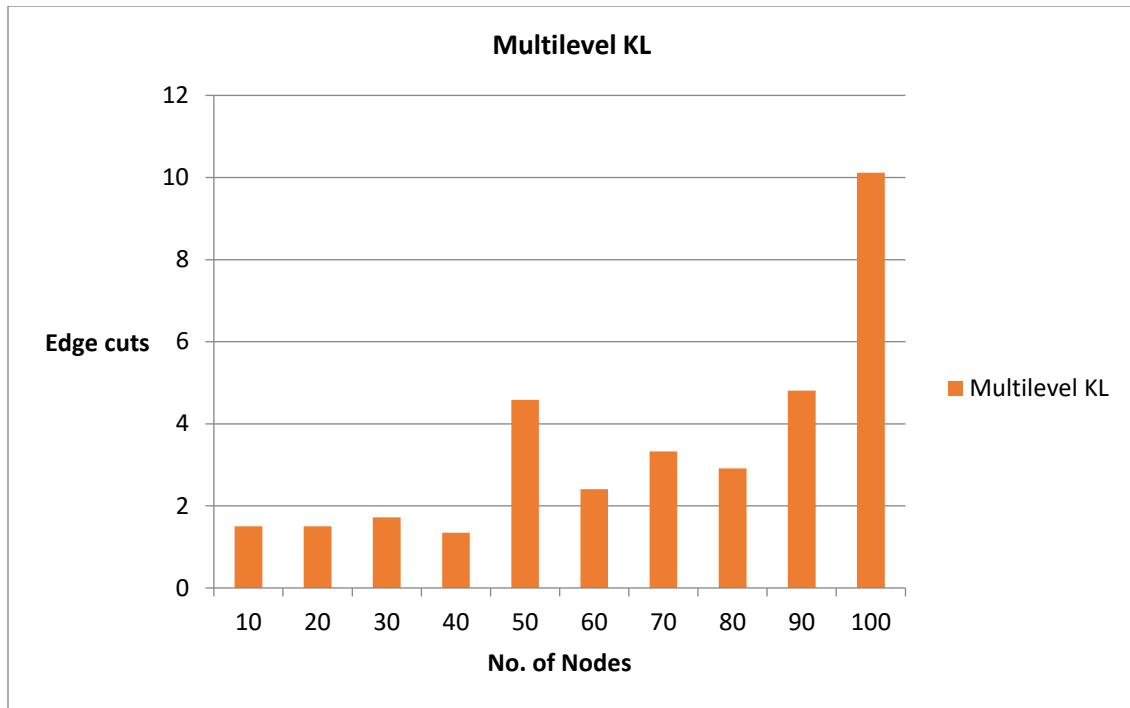


Figure 3-9 Graph represents the edge cuts results of the multi-level KL partitioning approach

The graph of fig. 3-9 represents results of the multi-level KL approach on the size of the different graphs varying from 10 to 100. The number of the graph cut are gradually increasing with the number of the nodes.

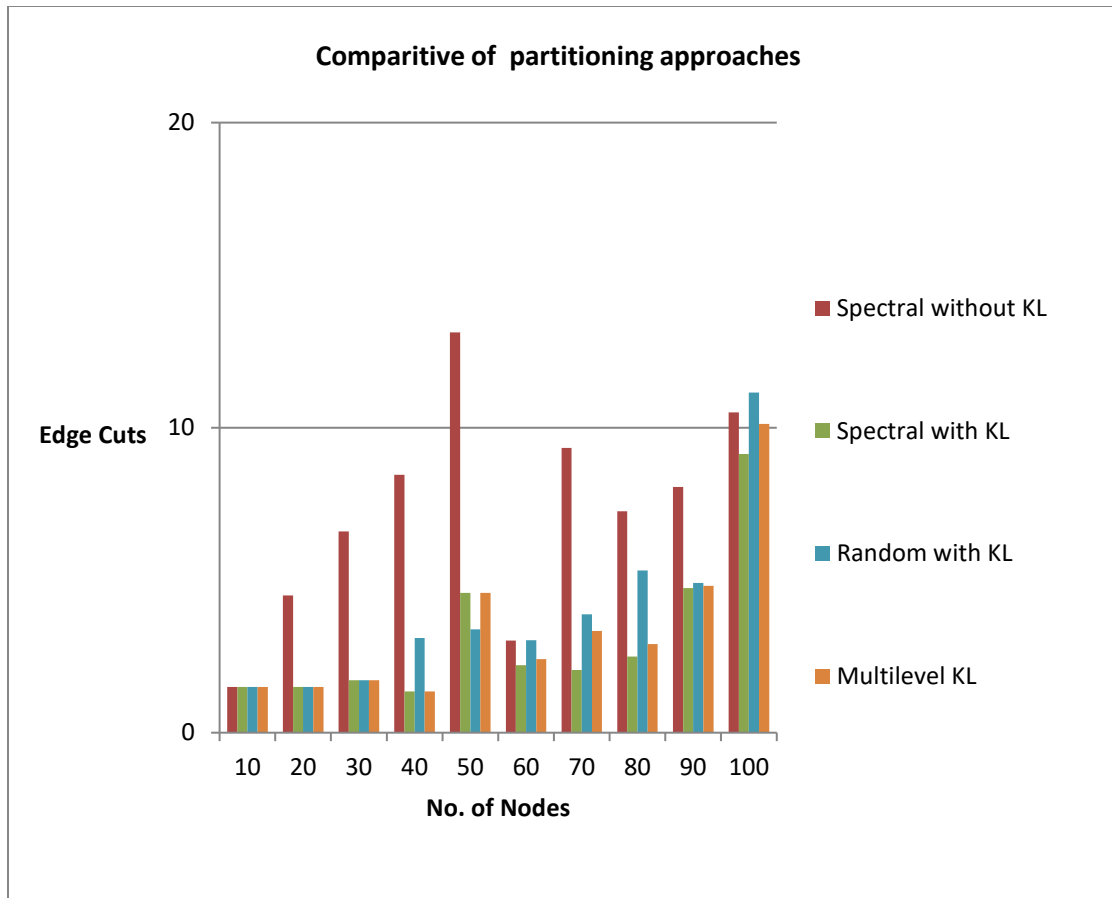


Figure 3-10 Comparison of the edge cut results of spectral with and without KL approach, random partitioning with KL, and multi-level KL.

The results indicate that considering the combination of spectral approach with the Kernighan Lin algorithm performs optimally compared to random and multi-level partitioning in a mobile cloud scenario. The minimum edge cut describes the minimum communication between the mobile device, i.e., client and cloud side. The spectral approach with KL is slightly better than the other approaches concerning the number of edge cuts

### **3.7 Summary**

This chapter presents a heuristic approach developed based on a multi-level hybrid approach for balanced graph partitioning. Although perfect partitioning in the mobile cloud computing offloading process is challenging, it can be achieved by optimizing the partitions at different levels. Heavy edge matching is performed on the graph initially for coarsening, and then spectral graph partitioning is applied for the initial partitioning of the graph. In the last stage, the Kernighan Lin algorithm is used for the refinement of the partitioned graph. The spectral partitioning and the Kernighan Lin algorithm have performed optimally compared to the existing approach of random partitioning and multi-level KL approach in terms of edge cuts. The spectral method and KL help to increase the edge cut size, which is important for communication between the client device and the cloud. In the future, the approach will be implemented in the real software design in the mobile application, and further investigation will be carried out considering different parameters in the account. Partitioning of an application during offloading in mobile cloud computing is a critical problem. Different heuristics can be developed considering various parameters during the study. In this work, the mobility of the device is not affecting the partitioning results. It can be an open area where researchers can work and develop a better model based on mobility.

---

## **CHAPTER 4**

# **OFFLOAD DECISION MAKING IN COMPUTATIONAL OFFLOADING**

---

### **4.1 Introduction**

With the development of emerging communication networks like 4G, 5G, and even 6G, mobile users have rapidly increased. This also increases the number of fascinating mobile applications like image processing, healthcare applications, gaming, etc. The primary concern in these mobile applications is their energy requirements. These applications consume a lot of energy, which drains the mobile battery faster. Mobile cloud computing provides computation resources like processing and storage to the needed devices in the cloud framework. The various reasons that intent the present developers to make cloud-based mobile applications are massive storage and processing facilities in the cloud. The emergence of communication networks improves the data transfer rates extensively. Technology like cloudlets, hypervisor virtual machines designed for a mobile device also enables mobile cloud background processing. Mobile cloud computing encompasses a computational offloading framework that helps deploy the compute-intensive task to the remote server to save energy and increase the mobile device's performance. Once the computation is completed on the remote server, the computed results are directed back to the mobile device. "When to offload." is a challenging question in computational offloading that always needs a solution for the mobile device's optimal performance during mobile offloading to the cloud servers.

## 4.2 Offload decision engine in MCC

A decision engine is a significant component in the offloading framework, which helps decide when to offload the task to the remote or cloud server. The decision engine's accuracy should be high for the flawless execution of the application during the offloading process. There are various subtasks in the offloading process, like identifying the offloadable task using partitioning, profiling, and offloading decisions. The application can be alienated into the compute-intensive portion and the graphical user interface portion (GUI). The graphics-related partition cannot be offloaded as it will refrain the application from performing, so the only compute-intensive section is offloaded to the remote server. Profiling is a process of gathering the various devices, networks, and application-related information required for the offloading operation. The process of offloading is entirely opportunistic, which relies on external features to offload the task from a mobile device. It is a non-trivial process that requires various parameters for decision-making. The mobile battery status, CPU cycles, global positioning system (GPS) for gauging the device's mobility, signal strength, bandwidth availability, and size of the application task are the various features that need to be extracted offloading process. This dynamic information is gathered on a timely basis by the profiler for the decision engine's offloading decision. A decision engine is a substantial component in the offloading framework, which helps decide when to offload to the remote or cloud server. Several contexts [120], like application specifications, mobile specifications, and network specifications, are utilized to make accurate decision-making. Different algorithms of machine learning like logistic regression [39], decision tree [121], naïve Bayes [122], fuzzy logic [123], and SVM [124] have been used in recent research for offloading decision making. Since the offloading engine is placed in the mobile device, it must be light weighted and also provide highly accurate offloading decisions based on the statistics provided to it by the context analyzer.

The work contributes to the following points in the computational offloading process:

- a) A technique is proposed for the offloading decision that aims to achieve higher accuracy. It is based on the stacked ensemble approach considering various mobile device parameters.

b) The proposed techniques aim to reduce the processing time and CPU utilization of the mobile device while taking the offloading decision.

A technique has been proposed by performing a stack ensemble approach on machine learning techniques like the Gaussian approach, multi-layer perceptron, k-nearest neighbors, and linear regression. It considers the various dynamics of the environment like task size, bandwidth, device battery, and device mobility. The proposed model performs better than other decision-making algorithms in terms of execution time and CPU utilization and achieves higher accuracy in making decisions while offloading the compute-intensive task to the remote server.

In the process of offloading decision, various profilers like network, device, and program profiler collect information related to network, application, battery level, and CPU cycle, which help the solver to the decision for the offloading. Energy and performance parameters are often evaluated during this phase. It is not always expected to offload the task on the remote server but depending upon device conditions and bandwidth; the decision to offload can be taken. The time  $T$  taken to execute the task locally is

$$\text{Local Execution Time} = W/S_m \quad (4.1)$$

Where  $W$  is the computation amount required for the second part and  $S_m$  is the processing speed of the mobile device. If the second part of the computation is offloaded to the cloud server having bandwidth  $B$ , the  $d_i$  amount of data takes  $d_i/B$  seconds to transfer data to the designated server. The benefit of offloading the task on a cloud server is only when the computation of the task, including the communication, can be achieved faster at the cloud server than executing locally.

$$\text{Execution Time (Remote)} = (\text{communication time to/from server}) + (\text{computational time on server}) \quad (4.2)$$

Key Decision:

$$\text{If Execution Time (Local)} < \text{Execution Time (Remote)} \text{ then Go Local} \quad (4.3)$$

$$\text{Else Execution Time (Local)} > \text{Execution Time (Remote)} \text{ then Go Remote} \quad (4.4)$$



### **4.3 Methodology**

The mobile application is considered as a graph in work. The graph nodes represent the interrelated tasks. During offloading, the application is partitioned to identify the job to be offloaded or executed locally. In this work, the nodes which are required to be offloaded are considered for the decision-making phase. Application partitioning is a primitive task and done before decision-making. It is assumed that nodes coming to the decision engine are offloadable tasks. During the context analysis phase, different metrics are collected using the profiling tool for offloadable tasks. The context analyzer works with the device profiler to collect the battery status of the device. Other profilers like network profilers give data related to GPS and bandwidth. GPS details come with the availability of satellites [125], but actual power consumption is based upon the active or inactive state of the GPS component.

#### **4.3.1 Offloading decision mechanism**

The offloading decision is considered as the concluding step of the computational offloading process. A decision engine helps to decide when to offload to the remote or cloud server. The decision engine accuracy should be optimal for taking the correct decision to offload in the offloading process. Since the offloading engine is placed in the mobile device, it must be light weighted and also provide highly accurate offloading decisions centered on the statistics provided to it by the context analyzer. The offload decision process is represented in Fig. 4-1, in which the decision engines direct the computation task towards remote or local execution based on the decision logic upon which it is designed. The various profilers collect information about the mobile device and provide the decision engine to take offload decisions. Based on the decision, the device either offloads the task to the cloud or edge server and performs remote execution or executes the task locally.

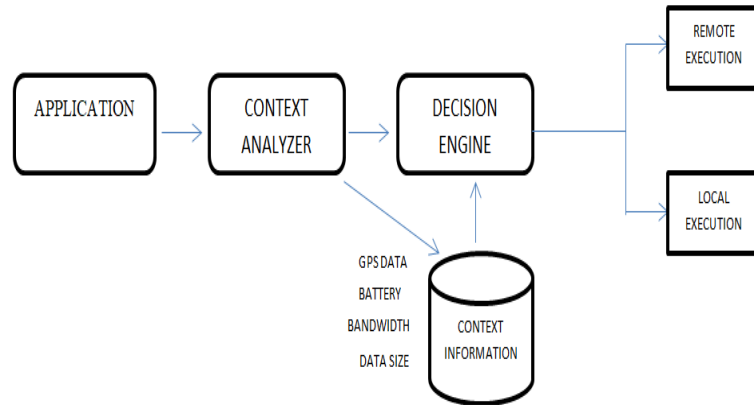


Figure 4-1 Offloading decision process

#### 4.4 Feature Selection

The metrics considered in the work are battery status, bandwidth, global positioning system (GPS) data, and the application's size. In most of the work related to offloading, the mobility of the device is not considered. As mobility is an important aspect, it is regarded as a parameter for decision-making. Mobility of the device is accessed with the GPS of the device. If the device GPS is in an ON state, it is considered that the user is moving, and if GPS is in an OFF state, the user is in a stable position. Table 1 represents the various features and classes used in work. Once the offloading decision is taken, the task will be executed locally represented as  $E_{LOCAL}$  or executed remotely represented as  $E_{REMOTE}$ .

Table 4-1 Features considered for offloading decision

Features	Information
$E_{LOCAL}$	Local execution class of task on the mobile device
$E_{REMOTE}$	Remote execute class of task on the cloud server
$D_{SIZE}$	Size of a task that is compute-intensive
$B_{CHANNEL}$	Bandwidth available during the offloading process
GPS	Global positioning system of the mobile device (on/off)
$BT_{STATUS}$	Battery available during the offloading process

#### 4.4.1 Multi-layer perceptron (MLP)

An artificial neural network (ANN) [126] resembles the human biological brain in artificial intelligence. The three different processes of human neurons are simulated in ANN. Fig. 4-2 represents the multi-layer perceptron model where the first layer is the receiving and evaluating the input signal; dendrites do the same in a human neuron. The second layer is the processing of input information by the node or neuron. The third layer of the process is used to generate the processed data's output, similar to the biological neuron's axon.

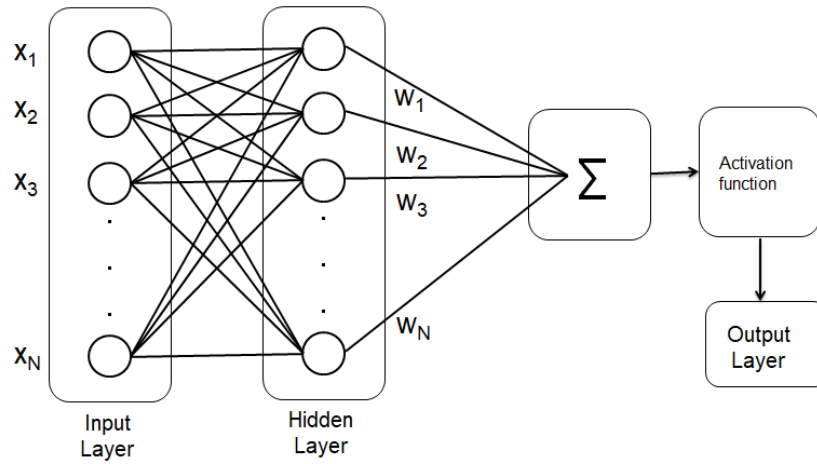


Figure 4-2 Multilayer perceptron model

The input layer of the model has  $n$  nodes defined as  $\mathbf{x} = [x_1 x_2 \dots x_n]$ . The model has  $n$  features to be given as input to the perceptron model and can have one or more hidden layers based on the computation complexity. There are weights  $\mathbf{w} = [w_1 w_2 \dots w_n]$  associated with the input features. The activation function determines the output of the perceptron model and is represented by

$$A = \sum_{i=1}^n x_i w_i \quad (4.5)$$

#### 4.4.2 K-nearest neighbor (KNN)

It is a supervised machine learning [127] based on an instance-based classification method where training records are stored and used to predict the class of the unseen record case. In

this scheme, the K parameter value is determined, which describes the number of nearest neighbors. Different kinds of distance metrics are used to classify the unseen case in the respected class. For example, the distances can be the Euclidean distance as the following equation:

$$d(u, v) = \sqrt{(|x_{u1} - x_{v1}|^2 + |x_{u2} - x_{v2}|^2 + \dots + |x_{um} - x_{qn}|^2)} \quad (4.6)$$

Also, the distance can be Manhattan distance as

$$d(i, j) = |x_{i1} - x_{j1}| + |x_{i2} - x_{j2}| + \dots + |x_{ip} - x_{jp}| \quad (4.7)$$

After calculating the distance of unseen case with the training set, the majority vote of class labels is taken among the k-nearest neighbors:

$$Y = \operatorname{argmax}_{v \in D_z} I(v = y_i) \quad (4.8)$$

Selecting K's value is a crucial task as a compelling too-small value leads to sensitivity to the noise points, and a larger value includes the data points of other classes.

#### 4.4.3 Gaussian naïve Bayes method

It is a statistical classifier [128] that enables the class's prediction based on probabilities, primarily based on the Bayes probability theory. The prediction principle of the naïve Bayes model is based on the following equation:

$$P(H | Q) = \frac{P(Q | H)P(H)}{P(Q)} = P(Q | H) \times P(H) / P(Q) \quad (4.9)$$

In the above equation, H assumes that Q has its place in the class label C. Let Q be a data sample where the class label is unidentified. The model's task is to work out the posterior probability P (Q|X), which is that the probability that assumption H embraces in observed data X. The P (H) is that the prior probability, P(Q) is the probability that the trial data is

monitored.  $P(Q|H)$  is the likelihood probability of observing the sample  $X$ , as long as the assumption holds.

#### 4.4.4 Logistic Regression

In machine learning, logistic regression [129] is one of the favored classification algorithms based on supervised learning. It is a particular case of linear regression where the target variable is categorical. In fig. 4-3 represents the logistic regression model where the data points are fitted in a logit model or sigmoid function, and the probability of the target variable is predicted after that.

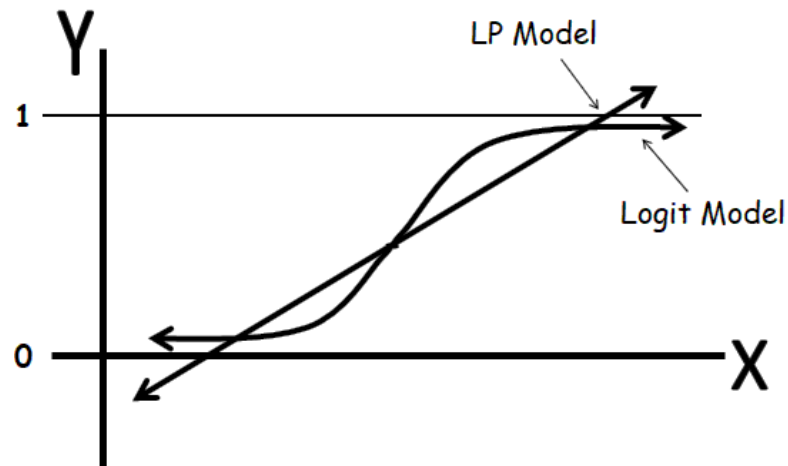


Figure 4-3 Logistic regression model

During the computation, a categorical value is predicted on a given independent input variable. The logistic regression model is based upon the sigmoid function, which is seen as

$$\text{sigmoid}(z) = 1/(1 + e^{(-z)}) \quad (4.10)$$

#### 4.5 PROPOSED STACK ENSEMBLE APPROACH

The stack ensemble approach is considered a powerful method in achieving the classification task [130]. In this technique, multiple machine learning techniques are combined to improve and boost prediction accuracy. The modern learning approach

follows the ensemble technique by combining the diversified set of machine learning algorithms to overcome the particular algorithm's weakness and build a robust model. Hence, each algorithm puts a substantial contribution where the strength of another algorithm counters its weakness. In the simplest form of the ensemble, all models are considered, and the unweighted average of the prediction of each model is utilized. The unweighted average will be calculated by dividing the sum of the models' predicted values in the library. In the current scenario, the concept of model stacking is used where an automatic assignment of balanced weights is done by using another level of the learning algorithm. The balanced weight is used to avoid the class imbalance in the scenario. Stacking is a competent ensemble method in which groups are laid down. Certain machine learning algorithms are placed at a specific group, and their prediction is passed to the next level as input. The algorithm designated at the next level is trained to combine the predictions of first-level algorithms optimally and generate a new prediction based on the previous information. In this scenario, the first-level models are MLP, KNN, and Gaussian naïve Bayes, and the second-level model is LR.

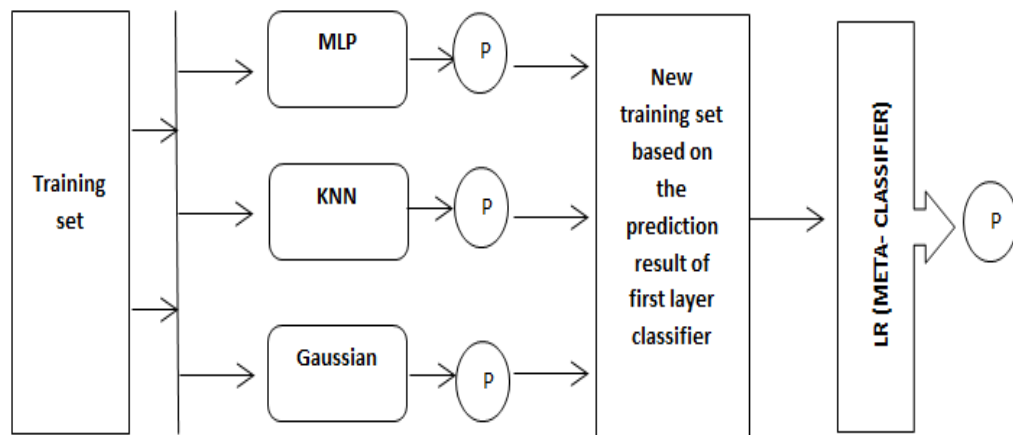


Figure 4-4 Stack ensemble approach used for predicting the offloading decision

---

**Algorithm 4.1** Stack ensemble-based approach for decision making

---

**Input:**  $D = \{x_p, y_q\}^m$  is the training data where  $x_p$  is a dataset that belongs to feature space and  $y_q$  is a label that belongs to a class label set),  $Q$  is the collection of all algorithm

**Output:** An ensemble classifier  $H$

1: Learning of first-level classifiers in the primary step

2: for  $q \leftarrow 1$  to  $Q$ , do

3: Learn base classifier  $h_q$  based on the original dataset  $D$

4: end for

5: Construct new data sets from  $D$  that contain an original class label and new features as the first-level prediction.

6: end for

7: Learn a second-level classifier based on a new dataset

8: return  $H(x)$

---

Stacking is a process of learning a high-level classifier on top of the base classifiers. It can be regarded as a meta-learning approach. The base classifiers are called first-level classifiers, and a second-level classifier is learned to combine the first-level classifiers. In fig. 4-4, the process of stacking is demonstrated in which has the following three significant steps. First-level classifier is learned on the original training data set. It can be learned either based on bootstrap sampling, boosting or performing parameter tuning for a homogeneous classifier, or applying different classification methods for generating the heterogeneous classifier. Secondary, new data has to be generated based on the first-level classifier or base classifier's output. The first-level classifier's output is fed as a new feature to the new dataset given in the second-level classifier. The class label of the second-level dataset remains the same as the first-level dataset. Based on the second level dataset, which is applied to any meta-classifier, the class level is predicted for the second level classifier.

## 4.6 Performance Evaluation

### 4.6.1 Experimental Setup

The proposed decision engine has been implemented in Python language. The technique has been executed on a device having configuration Intel (R) core i3 CPU M330 @ 2.12 GHz processor, 8 GB of RAM, and Windows 10 OS. The decision engine method's performance is assessed concerning specific parameters like battery status, bandwidth, GPS data, and the application's task size. The evaluation has been accompanied to investigate the effectiveness of the decision engine in terms of accuracy. A higher value of accuracy reflects the correct decision-making by the offloading engine. The behavior of the suggested model is also compared with the prevailing algorithm like logistic regression (LR), k-nearest neighbors (KNN), Gaussian naïve Bayes (Gaussian), and multi-layer perceptron model (MLP). The application's task size for offloading and device battery levels is taken randomly between 100 KB and 4000 KB and 800 mAh to 4000 mAh, respectively, using the uniform distribution. In this work, a lower battery level is considered a 20% percentage of the actual device battery [131], and the battery status above it is regarded as a greater battery level. The bandwidth [57] of the device is kept between 400 kbps to 800 kbps. The application task size [57] is considered between 100 KB and 4000 KB. The condition of mobility [126] is also considered based on the GPS parameter in which two states are considered, i.e., stable if it is OFF and unstable state if it is ON. For every model, i.e., LR, KNN, Gaussian, MLP, and proposed model, k-fold cross-validation (where  $k = 10$ ) is applied to seek out the best hyperparameters at the training stage. The dataset of 1600 records is created using the Python language, which depicts the mobile device's real-time scenario. The dataset is split into two sets in the current work, i.e., a training set and a testing set, where 80% is for the training set and 20% is for a testing set. The confusion matrix is used to examine the various machine learning model's performance with the proposed model more accurately. A confusion matrix is a valuable tool for examining the classifier's fitness in classifying the data correctly in the various classes of the problem scenario. The performance of the models can be measured using a variety of measures based on true positive (TP), true negative (TN), false positive (FN), and false-



negative (FN). The various measures that are calculated for the problem statements are:

$$\text{Sensitivity} = \text{TP} / \text{Positive} \quad (4.11)$$

$$\text{Specificity} = \text{TN} / \text{Negative} \quad (4.12)$$

$$\text{Accuracy} = \text{sensitivity} * (\text{Positive} / (\text{Positive} + \text{Negative})) + \text{specificity} * \text{Negative} / (\text{Positive} + \text{Negative}) \quad (4.13)$$

$$\text{Precision} = (\text{TP}) / (\text{TP} + \text{FP}) \quad (4.14)$$

$$\text{F1\_Score} = 2(\text{precision} * \text{sensitivity}) / \text{precision} + \text{sensitivity} \quad (4.15)$$

$$\text{FPR} = \text{FP} / \text{FP} + \text{TN} \quad (4.16)$$

Sensitivity or true positive states how well the model classifies the positive tuple correctly. In contrast, sensitivity or false positive talks about how well the negative tuples are correctly classified as negative. The accuracy of the model for classification states the correct prediction rate. The summarization of sensitivity and precision can be seen in F1\_Score, whereas the ratio of negative tuples classified as unfavorable can be seen as a false positive rate (FPR). Comparing the various techniques like Naïve Bayes, KNN, Logistic regression, MLP, and the proposed model has been done based on the above-listed measures.

## 4.7 Results and Discussion

Table 4-2 Performance of different algorithms and proposed methodology

Algorithm	Specificity	Sensitivity	Precision	F1_Score	Accuracy	FPR
Gaussian Naïve Bayes	92.08	88.52	83.07	85.71	91	7.91
KNN	98.2	94.26	95.83	95.04	97	1.7
Logistic regression	89.56	84.42	78.03	81.1	88	10.43
MLP	98.92	96.72	97.52	97.11	98.25	1.06
Proposed Model	<b>99.28</b>	<b>96.72</b>	<b>98.33</b>	<b>97.52</b>	<b>98.5</b>	<b>0.77</b>

Table 4-2 shows the various scores of various models and the proposed methodology computed to compare the performances. It has been found that the accuracy of the proposed model is 98.5%, which is comparatively higher than other techniques. The proposed method's specificity and sensitivity are 99.28% and 96.72%, which is better than the compared methods. The FPR parameter is low as 0.77, which signifies the false positive classification. The F1 score is 97.52%, which is higher than other methods, stating that precision and sensitivity are more valuable. It also depicts that the decision engine can adequately handle a balanced class where the decision can be biased based on the training dataset.

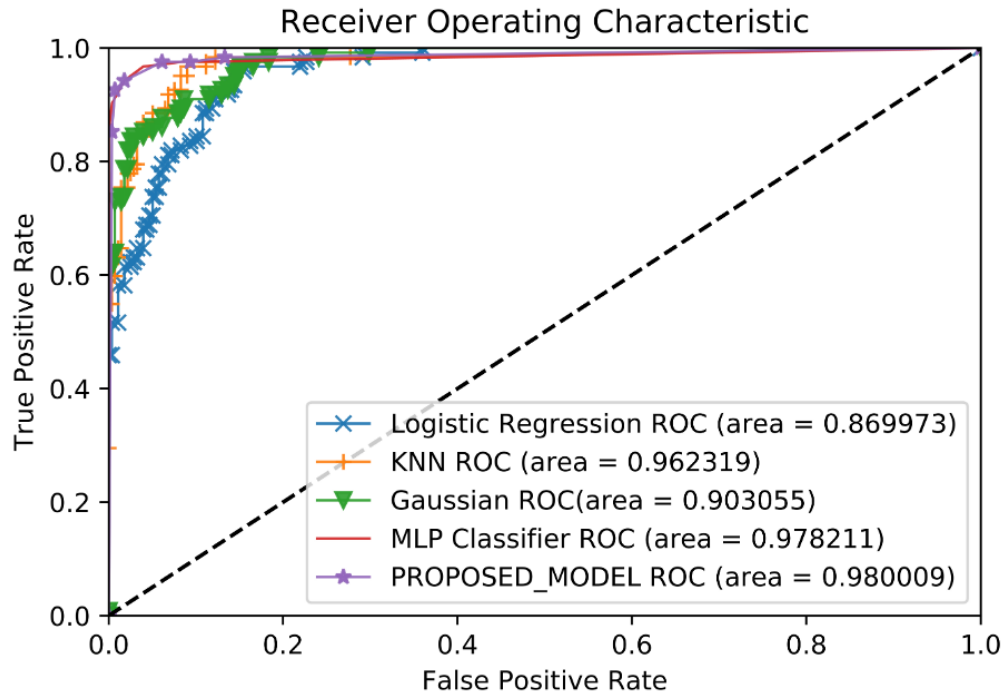


Figure 4-5 ROC curve of the different algorithm and the proposed methodology

The performances of various models are also compared based on the Receiver Operating Characteristic (ROC) curve. The area under the curve (AUC) value is the highest in the proposed model, which is 0.98 compared with other models. Fig. 4-5 states the different algorithms' ROC curves and depicts the trade between the (1-specificity) and sensitivity. When the curve value is closer to the value of 1 and lies more towards the graph's left side, it represents that the true positive rate (TPR) of the classification model is more than the false positive rate (FPR).

Table 4-3 CPU utilization of Decision engine module (in percentage)

<b>No. of computational task</b>	<b>Naïve Bayes</b>	<b>MLP</b>	<b>KNN</b>	<b>LR</b>	<b>Proposed Scheme</b>
10	5.3	4.4	4.3	4.8	<b>4.4</b>
20	7.2	7	7.2	6.8	<b>4.9</b>
30	8.2	7.7	7.6	9	<b>7.1</b>
40	10.5	13.3	7.9	6.3	<b>7.6</b>
50	8.3	6.4	7.4	6.6	<b>7.1</b>
60	8.9	7.3	9.6	8	<b>9.6</b>
70	5.7	9.8	5.9	9.4	<b>5.6</b>
80	10.5	10.7	9.4	12.3	<b>9.1</b>
90	13.3	14.1	7.7	10.9	<b>7.2</b>
100	13.5	14.6	9.6	10.5	<b>8.5</b>

Table 4-3 represents the performance of the decision engine based on CPU utilization. Different application size tasks are given as input to the decision engine module along with bandwidth, device energy, and GPS status. The proposed scheme performs better as compared to other machine learning algorithms. The CPU utilization in a particular set of computational tasks is less in an ensemble-based approach, which shows it as a powerful strategy to be considered as a decision engine. CPU utilization percentage is varied since each allocation given to the decision engine has a randomized computational task based on mobile device state. The CPU utilization of the proposed scheme is around 9% less on average than other methods.

Table 4-4 Execution time of Decision engine module (in a sec)

<b>No. of computational task</b>	<b>Naïve Bayes</b>	<b>MLP</b>	<b>KNN</b>	<b>LR</b>	<b>Proposed Scheme</b>
10	1.09	1.16	0.8	0.9	<b>0.41</b>
20	1.43	1.21	0.95	1.27	<b>0.83</b>
30	1.48	1.48	1.37	1.43	<b>0.91</b>
40	1.28	0.98	1.29	1.44	<b>0.79</b>
50	1.32	1.2	1.3	1.38	<b>0.76</b>
60	1.21	1.16	1.83	1.45	<b>0.81</b>
70	1.34	1.04	1.5	1.08	<b>0.34</b>
80	0.71	1.4	1.5	1.14	<b>0.5</b>
90	1.44	0.78	1.33	0.92	<b>0.36</b>
100	2.3	0.7	1.8	1.9	<b>0.37</b>

Table 4-4 represents the performance of the decision engine based on execution time. The decision engine's execution time is comparatively better when a randomized computational task is given to it in a different count. The proposed scheme is around 47% faster as compared to the algorithms.

A decision engine is a key component of the offloading system, since it determines when a task should be offloaded to a remote or cloud server. This research proposed a stack-based classification method for performing the decision engine's offloading decision-making duty in the computational offloading process. When compared to other machine learning models, the suggested classifier is found to be more accurate. The ROC curve for the suggested approach is shown, which has a higher TPR than other models. Based on the device's GPS, the feature of mobility is also taken into account at work. The proposed technique is a viable decision engine solution because of its shorter execution time and lower CPU use.

#### **4.8 Summary**

This work suggested a stack-based classification method that can perform the offloading decision-making task of the decision engine in the computational offloading process. The accuracy of the proposed classifier is found better when compared with different machine learning models. The proposed technique's ROC curve is presented, which has better TPR compared to other models. The feature of mobility is also considered in work based on the GPS of the device. The lower execution time and lower CPU utilization make the proposed scheme a viable decision engine approach.

---

## CHAPTER 5

# MOBILITY MANAGEMENT SCHEME DURING COMPUTATIONAL OFFLOADING

---

### 5.1 Introduction

Seamless connectivity is the prime requirement for performing computation during offloading in MCC. It enables the mobile user to sustain uninterrupted and constant connectivity. It does consider the location and environment also where the mobile device is moving. In mobile cloud computing, seamless connectivity [132] enables the device to remain connected with the cloud service providers without degrading the QoS. At present, an enormous number of wireless data network technologies are Wi-Fi, Wi-MAX, 3GPP, LTE, and, more recently, the 5G technologies also. The core technologies in a cellular network, like a circuit-switched network, are also transformed to the internet protocol (IP) based network. It leads to the usage of the IP-based packet by the LTE technologies also. Many of the IEEE standards and RFC are proposed and implemented in recent times on these new transformations. Hence, there is a need to work on the mobility management of the devices also so that robust, cost-effective, highly available services can be provided to mobile users. It needs to assure that the mobile device needs to remains connected in different geographical locations. In traditional communication systems, the applications were limited to two-way directions, like voice communication, emails, and text. The recent emerging applications like telemedicine, sensor-based IoT applications, video streaming applications have changed the focus from two-way communications and open a new dimension of mobility in the communication system. Mobility management is an essential dimension of ubiquitous computing, which makes it more valuable and usable. The problems in MCC are similar to mobile computing, such as the issues [9] [10] related to handoffs, network delays, bandwidth, and limited battery energy. In computational

offloading, the mobile device or mobile nodes (MN) roams around different access networks like a mobile device may initially start some cloud services in the 4G network and commit offloading process in the Wi-Fi network due to its mobility.

## **5.2 Mobility management in MCC**

Tracing mobile nodes, preparing a handover, picking a new network, registering the mobile device with various service providers, and conducting the handover are all responsibilities of mobility management. Finding the SMD's Point of Attachment (PoA) and keeping the SMD's connectivity while changing the PoA are key issues in mobility management. The initial matter is controlled by location management [133], and the second is handover management [134]. A geographical coverage area is an area is divided into a sub-area known as a cell. It is a cell cluster means the group of a cell. A cell is assigned a bunch of frequencies and served by a base station consisting of a trans receiving system and control unit. A base transceiver system is deployed as a hub to handle the information transfer between source and mobile terminals. Now, the mobile switching center control all BSC, MSC takes part in the registration update, authentication, and call delivery process. The basic function of a network is to allow mobile devices to communicate over GSM and UMTS networks. The data that has to be processed is transferred from the mobile device to the cloud or edge servers, where it is computed on the servers. The mobile device receives the calculation results from the cloud or edge server after the computation is completed. The offloading process in MCC may use heterogeneous types of wireless networks, which may include Wireless LAN (WLAN) and cellular services like 3G, 4G services, and even 5G services in the near future. Various issues get raised when the offloading application runs, like connectivity, the energy level of mobile devices, and the availability of the cloud or edge servers. The different types of mobile services [4] are available to the mobile device like Bluetooth, Wi-Fi, 2G/3G/4G services.



### 5.2.1 Network and Cloud Probing

The aim of MM is to track where subscribers are allowing calls, SMS, and other mobile phones. Like roaming is a significant procedure in MM, with the help of this, the customer automatically receives the calls, sends data, and travels outside the home network in Heterogeneous Access Networks (HAN) environments. The handoff procedure is a two-step process that transfers an active call from one cell to another, i.e., when a mobile node (MN) travels into a different cell while a conversation is in progress, the MSC immediately switches the call to a new channel belonging to a new base station. The handoff is initiated when the new base station's average signal level exceeds the current base station by a certain amount. When a device moves from one network to another without changing the network type, then the process is called horizontal handoff, and if it changes the network, then it is called vertical handoff. Basically, -90 dBm to -100 dBm is an acceptable voice quality range. Location management keeps track of the active mobile station within the cellular network to route the incoming call. A mobile station is active if it is powered ON. Usually, location management means how to track a mobile station between two consecutive phone calls. The goal of location management is to maintain track of the current location of users so that incoming packets can be routed to the mobile location. If a mobile station sends an update message, its specific location is unknown, which causes considerable delivery delays. If the mobile station's position is updated often, the network knows where it is, and data packets can be transmitted without any further processing. However, mobility management consumes a significant amount of uplink radio capacity and battery power.

Heterogeneous networks (Hetnets) have various types of features like data rates, received signal strength (RSS), network capacity, bandwidth, and coverage span. The mobile device perceives these features and decides to select the best available network in its current location. The mobility of the device has a greater impact on the process of offloading. While the user is in a moving state, the probability of changing the network is high. For the flawless process of offloading, the transition among the cellular network must be smooth, and handoff must be minimized so that the mobile device remains connected with

the cloud server. Many modern computing environments are cloud-specific, like gaming applications, healthcare services, natural language processing (NLP) based applications, and computer vision. The mobile device can perform computations while roaming and may require cloud services for offloading purposes.

During cloud probing and selection, the best cloud is searched for the offloading process based on QoS offered by the cloud service provider. The parameters like network throughput, CPU utilization, latency, and delay are considered for cloud probing services (CPS). Based on these parameters, the cloud services are ranked using the cloud ranking services (CRS). The mobile applications use RESTful APIs for taking the cloud services. After selecting the appropriate cellular and cloud services, the task is uploaded for computational offloading.

### **5.3 Proposed mobility scheme during computational offloading**

Computational offloading is a complex problem in mobile cloud computing. The offloaded task initiated by the mobile device during the offloading process reaches the cloud server through the heterogeneous cellular network or Wi-Fi network. In fig. 5-1, the offloading mechanism is shown where the mobile device may roam among different networks, and the mobility of the device compels it to choose the network with higher signal strength through network probing. Further, cloud probing is also required for selecting the best cloud for executing the offloaded application component. In this work, an assumption is made that mobile devices have fixed the cloud server based on the technique used in [8], Simple Additive Weighting (SAW). The contribution of this work is to devise a mechanism for network probing where an optimal network can be searched, and the device remains connected to the cloud server. It aims to have less handoff and dropped rate happen so that seamless connectivity can be realized without interrupting the cloud service.

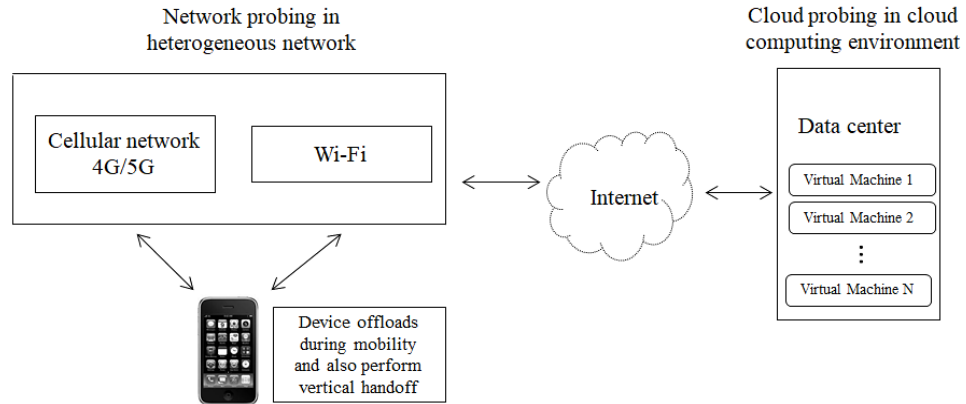


Figure 5-1 Computational offloading of task during mobility in MCC

The proposed technique aims to provide a mobility scheme where the number of handoffs can be minimized, and the handoff dropped can be reduced. In this work, the COST-231-Hata path loss model [135] is considered along with the pathway mobility model in various mobile environments like urban, semi-urban, and rural locations to simulate the cellular model of the offloading device.

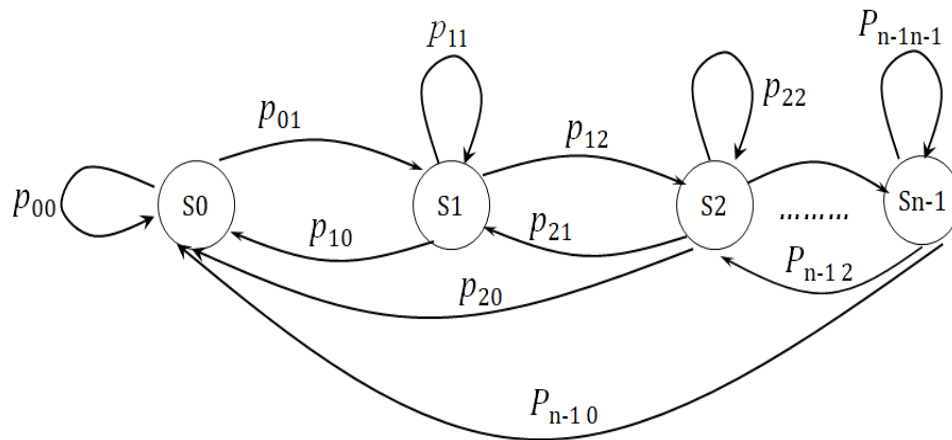


Figure 5-2 N state Markov chain model for mobility prediction in MCC

In this scenario, the mobile device roams in the different states or locations, represented with the Markov model. In fig. 5-2, the generic case of n state Markov model is presented where a device moves among different states, and the probability of moving to

the next state is updated after every movement in the cellular or Wi-Fi area.

$$P \{S_{n+1} = w \mid S_n = q_n, S_{n-1} = q_{n-1} \dots S_1 = q_1, S_0 = q_0\} \quad (5.1)$$

The device's mobility is a stochastic random process, and the device can move to any state in state space. State-space is an area which is having network coverage or Wi-Fi access point. The mobile device performs the transition from one state to another based on the transition matrix. The probability of the transition matrix gets updated after every movement initially and learning is undergone. Fig. 5-3 depict the transition matrix for N states.

$$\begin{array}{c}
 P_{ij} \\
 0 \\
 1 \\
 2 \\
 \cdot \\
 \cdot \\
 \cdot \\
 N
 \end{array}
 \begin{array}{c}
 0 \quad 1 \quad 2 \quad \dots \quad N \\
 \left[ \begin{array}{cccccc}
 P_{00} & P_{01} & P_{02} & \dots & P_{0N} \\
 P_{10} & P_{11} & P_{12} & \dots & P_{1N} \\
 P_{20} & P_{21} & P_{22} & \dots & P_{2N} \\
 & & & & \\
 & & & & \\
 & & & & \\
 P_{N0} & P_{N1} & P_{N2} & \dots & P_{NN}
 \end{array} \right]
 \end{array}$$

Figure 5-3 Transition matrix

In the first-order Markov chain, the model forecasts the next action by only seeing the user's last action. In the current scenario, the prediction of the next location movement accessed by a mobile device is the problem consideration. The Markov model consists of mobile user location as states, and the fourth-order Markov model is considered, which means the next state can be predicted based on the previous four states. It has been seen that a higher-order Markov chain [136] increases the accuracy of the prediction; there is an increase in the number of states also. So, to manage the trade-off, the fourth-level Markov model is applied for location prediction, and further higher-order may make a mobile application more complex and could affect the battery drainage also during the decision making for handoff.

## 5.4 Proposed scheme for mobility management

In this paper, the fourth-order Markov chain-based mobility scheme is proposed, which aims to reduce the number of handoffs and dropped rates also. The mobile device is initially connected with either Wi-Fi or a cellular network. Once the device starts the offloading process, it may start moving from one location to another location. The problem has considered the pathway mobility model [137] in the scenario, which has been implemented in a different cellular environment like urban, semi-urban, and rural locations and compared with SINR based handoff mechanism [138]. The cloud server is assumed to be stationary and connected to the cellular service in the current scenario. When the device starts moving, it probes for the network or access point having strong signal strength. The network probing scheme is proposed as:

---

**Pseudocode 5.1** Network probing for a candidate list of BS/AP and network selection

---

Finding the list of the predicted base station and access point based on the current location of the user equipment using a 4th order Markov model

**Input:**

Base station threshold

Access point threshold

D, database of all access point and base station

**Output:** L, list of candidate base station and access point in D

**Method:**

1. User equipment is connected to a base station or access point connected with the cloud.
2. Candidate list = { }
3. For each access point and base station in D, do
4. Repeat
5. Compare the current RSS with the existing Base station and Access point RSS

6. Update candidate list with that BS and AP whose RSS is above the threshold value in the available area
  7. until no change
  8. If (User equipment Current RSS < threshold value), Select BS / AP from the candidate list based on the predicted location of user equipment derived using 4<sup>th</sup> order Markov model and perform offload
  9. Else continue offload at the current location
- 

## 5.5 Performance evaluation

### 5.5.1 Experiment Settings

In this work, a scenario is assumed to connect the mobile device to the cloud server through various base stations and access points. When the device starts moving, it probes for the network or access point having strong signal strength and select the network based on the next predicted move. In table 5-1, the frequency of the base station is considered 2100 MHz for all towers as the 4G network is considered in work. The height of the base stations (Hbs), transmitted power (Pt), transmitted antenna (Gt), and connector loss (A) are also considered for urban, semi-urban, and rural while defining the base stations [139] during this work. The work presented also compares techniques based on the cellular area like urban, semi-urban, and rural.

Table 5-1 Base stations feature based on the cellular environment

	<b>Urban</b>	<b>Sub-Urban</b>	<b>Rural</b>
Frequency Base station (F)	2100 MHZ	2100 MHZ	2100 MHZ
Height of base station (Hbs)	30	34	38
Transmitted power (Pt)	43	46	48
Transmitted antenna (Gt)	18	18	18
Connector loss (A)	2	2	2

The cellular network 's received signal strength (RSS) can be computed as:

$$\text{RSS} = \text{Trasmit\_Power (Pt)} + \text{Trasmit\_Antenna (Gt)} - \text{Path\_loss (PL)} - \text{connect\_loss (A)} \quad (5.2)$$

The path loss model expressed for the cellular network COST-231-Hata model [23]

$$\text{Path\_loss (dB)} = A + B \log_{10} (d) + C \quad (5.3)$$

In equation 3,

$$A = 46.3 + 33.9 \log_{10} (\text{carrier\_freq}) - 13.28 \log_{10} (\text{BS\_height}) - a (\text{Mobile\_height}) \quad (5.4)$$

$$B = 44.9 - 6.55 \log_{10} (\text{BS\_height}) \quad (5.5)$$

and the value of C is 0 for rural and suburban areas and 3for urban areas

In table 5-2, various parameters are presented, which are used in the WLAN simulation setting.

The path loss [24] in WLAN is represented in dB as:

$$\text{PL(WLAN)} = \text{constant power loss(L)} + 10 (\text{path loss exponent n}) \log (d) + \text{Fading effect(S)} \quad (5.6)$$

The RSS for WLAN is articulated in dBm as:

$$\text{RSS (WLAN)} = \text{Trasmit\_Power (Pt)} - \text{Path\_loss (PL)} \quad (5.7)$$

Table 5-2 WLAN access point features

<b>Features of WLAN</b>	<b>Parameter Values</b>
Constant power loss (L)	147dB
Path loss exponent (n)	3 dB
Shadow fading (S)	2
Transmit Power (Pt)	1dBm

### 5.5.2 Results and Discussion

During the implementation of the strategy, the numbers of user equipment were considered in the range of 100 to 500, and the three environments were considered, i.e., urban, suburban, and rural.

Table 5-3 Handoff results in a different environment

No. of user equipment	Urban		Sub-Urban		Rural	
	Urban SINR Handoff	Urban Proposed Handoff	Sub-Urban SINR Handoff	Sub-Urban Proposed Handoff	Rural SINR Handoff	Rural Proposed Handoff
100	4404	3849	3740	3280	2866	2450
200	6924	5919	5885	4930	6705	5820
300	8651	7462	9262	7790	6090	5156
400	10397	8703	10428	8849	7401	6427
500	12938	10714	11912	9990	9559	8002

Table 5-3 represents the count of handoff in the urban, sub-urban, and rural environments. It is calculated on different numbers of users in different cellular network features.

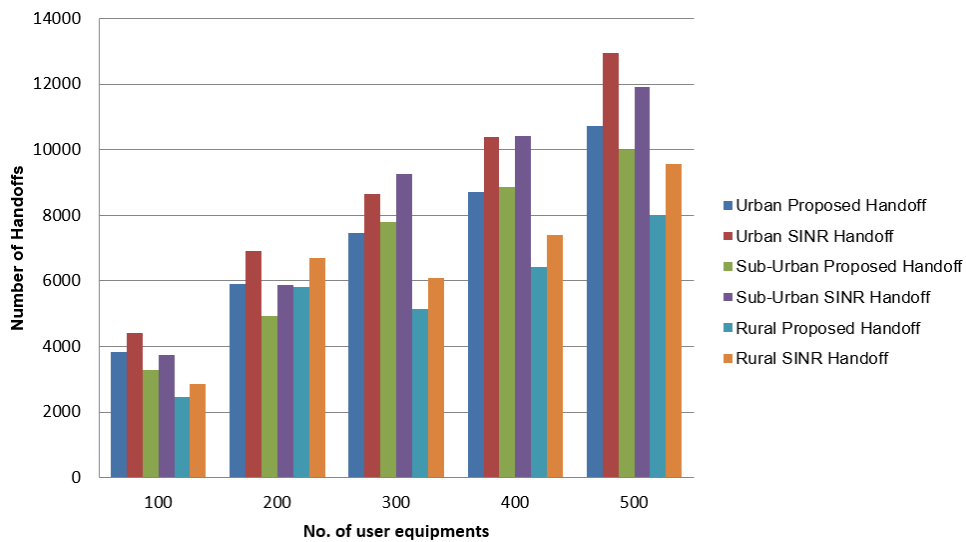


Figure 5-4 Handoff comparison between the two different strategies



Fig. 5-4 presents the comparison of the number of handoffs in the proposed technique with SINR based approach in Urban, Sub-Urban, and Rural. Based on the simulation, it is founded that the proposed handoff strategy based on location prediction performs better as compared SINR based handoff. The numbers of handoffs are less in three different environments when compared with SINR based handoff. The lower handoff count depicts seamless connectivity with the cloud server and a flawless offloading process.

Table 5-4 Handoff dropped results in a different environment

No. of user equipment	Urban		Sub-Urban		Rural	
	Urban SINR Dropped	Urban Proposed Dropped	Sub-Urban SINR Dropped	Sub-Urban Proposed Dropped	Sub-Urban SINR Dropped	Sub-Urban Proposed Dropped
<b>100</b>	2557	1053	2189	882	1270	560
<b>200</b>	4235	2408	3580	2203	4089	2274
<b>300</b>	5494	4396	5801	4530	3688	2273
<b>400</b>	7025	5537	7164	5649	4680	3702
<b>500</b>	9411	7124	8809	6709	6430	5100

Table 5-4 presents the handoff dropped results in the urban, sub-urban, and rural environments.

In this work, a mobility-based offloading system in MCC is suggested, with the goal of reducing the number of mobile device handoffs as well as the number of handoffs dropped. The 4th order Markov model is developed to anticipate the user equipment's next location. The technology will allow user equipment to stay connected to a cellular or Wi-Fi network, which will then connect to a cloud or edge server for the computational offloading operation to be completed. For implementing the proposed work, the work encompasses diverse mobility situations such as urban, semi-urban, and rural, and it has been found to perform better than the comparative SINR-based technique.

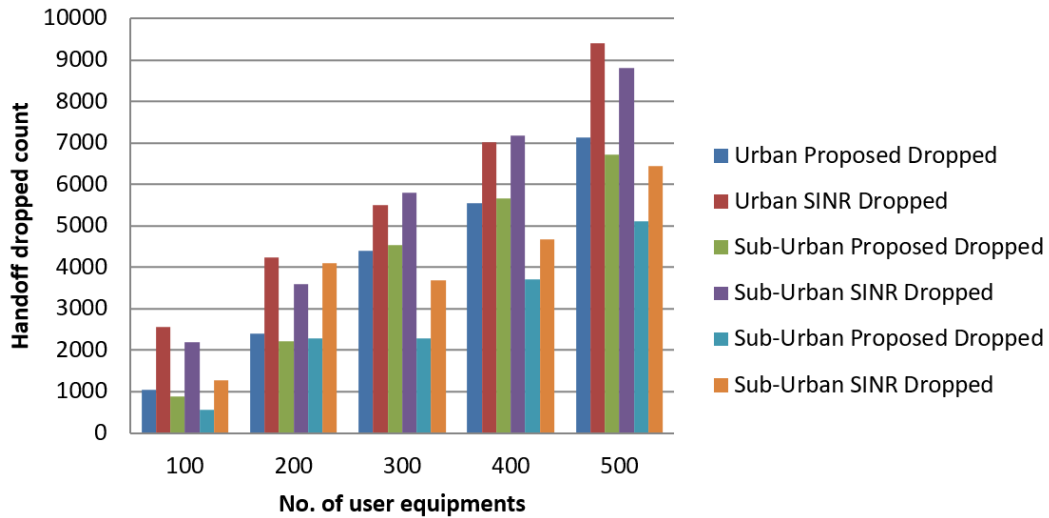


Figure 5-5 Comparison of the number of handoffs dropped

Table 5-4 shows the comparison of the number of handoffs dropped in the proposed technique with SINR based approach in Urban, Sub-Urban, and rural environments. Handoff dropped count depicts the scenario when user equipment does not get the required signal strength during handover and loses the connectivity with the base station of the cellular network. The results shown the fig. 5-5 interprets that the proposed technique reduces the dropped rate in the cellular network, and altogether reduction in dropped rate provides seamless connectivity of the mobile device to the cloud server. In all three cellular environments, the proposed technique has performed well in handoff dropped to count.

## 5.6 Summary

This chapter proposes a seamless mobility scheme based on the heterogeneous network of Wi-Fi and 4G networks. The proposed scheme is based on the fourth-order Markov model for mobility prediction and received signal strength (RSS) of the network nearest to the next predicted move of the device. The chapter is arranged in different sections where the related work in mobility in mobile cloud computing has been presented. The proposed mobility scheme for the offloading process has been presented in the chapter, along with the results of the proposed approach.

---

## CHAPTER 6

# A MULTI-OBJECTIVE TASK SCHEDULING SCHEME IN MOBILE CLOUD COMPUTING

---

### 6.1 Introduction

Cloud computing accomplishes a range of virtualized resources, which creates scheduling a significant component. A client may need several virtualized resources to process the task on the cloud. That is why scheduling cannot be done manually. The scheduling process needs to be modelled automatically through scheduling schemes that aim to maximize the central processing unit (CPU) utilization and reduce the energy consumption of the virtual machine. MCC is currently an encouraging field in the cyber-physical world. It is an amalgamation of mobile computing and cloud computing. Computational offloading is one feature in the mobile cloud application that offloads the task to the cloud server, processes it, and gets the results back on the mobile device. During offload, the job needs to be queued on the cloud servers and allocated to the virtual machines. Task scheduling is an important step where the mobile task is assigned to the servers and processed somehow. In the overall offloading process, energy conservation is a significant concern. The scheduling problem involves mapping the offloaded task to the cloud server while satisfying the energy and time constraints. This chapter presents a hybrid scheduling scheme based on Gaussian-based multi-objective particle swarm optimization (GMOPSO) and bacterial foraging optimization (BFO). This scheme performs better when compared to other variants of particle swarm optimization (PSO) in terms of makespan and energy efficiency. The cloud schedulers are a fully-managed entity in the cloud service providers. It minimizes the human intervention in scheduling the task and provides a reliable solution. The tasks are scheduled on various virtual machines available in the physical servers of the data centers.

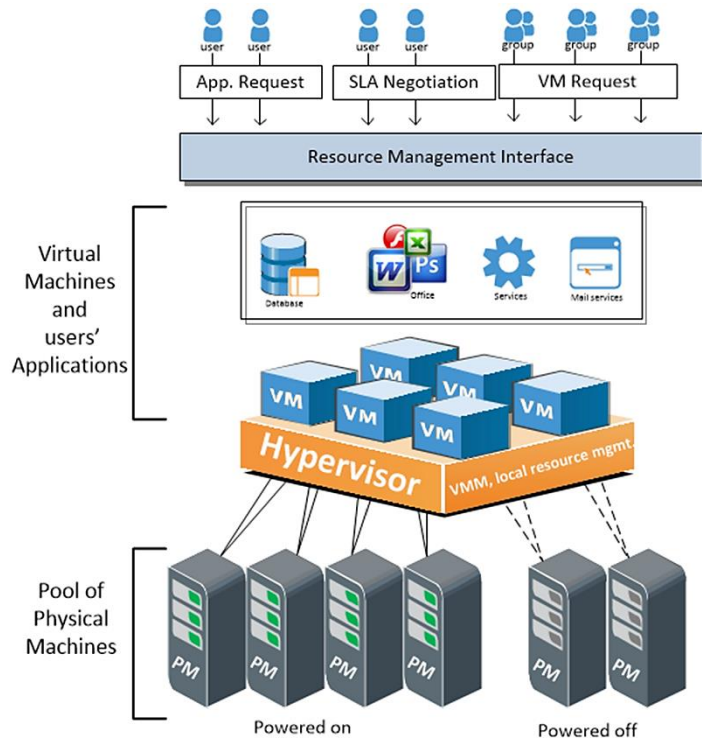


Figure 6-1 Virtualization in cloud computing

The fig. 6-1 depicts a process of virtualization [140] which is a core technology of cloud computing that permits running different OSs concurrently on one physical machine (PM). These operating systems run in isolation from each other on these physical machines by particular middleware technology known as virtual machines. The middleware software which enables creating, running, and managing these virtual machines on single or multiple pools of physical machines is called hypervisors. The brokers or hypervisors enable scheduling on the virtual machines of the physical system. The cloud service providers monitor all the requests coming to the server and keeps the information about the utilization level of physical machines present in the data center. The characteristics of the task are recorded, and dependencies are also closely watched. The tasks that reach the server are broadly classified as CPU bounded, and Input-output (IO) bounded. The compute-intensive tasks require a large amount of Random access memory (RAM) to get processed. The IO-based tasks are mostly requiring the peripherals devices connected nearby the servers.

## 6.2 Related work

### 6.2.1 Task scheduling during computational offloading

Computation offloading is the technique inside mobile cloud computing where an application is partitioned upon local and remote execution based on some criteria. Fig. 6-2 depicts the system model for task scheduling of offloaded tasks. In the offloading process, an application is partitioned, and based upon some measures; the decision has been taken to offload the task or execute it locally. Those tasks which are identified to be performed on an edge server are offloaded on it. The objective functions considered in the work are to minimize the makespan of the mobile task and minimize the mobile task's energy cost.

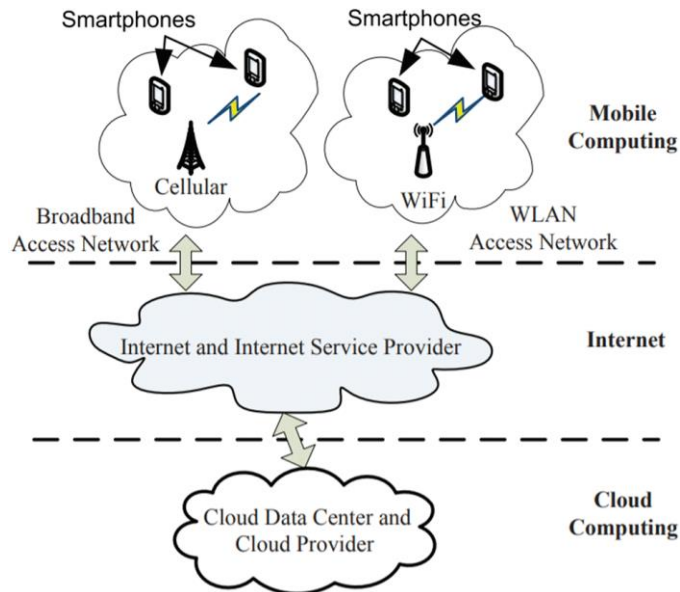


Figure 6-2 System model for scheduling the offloaded task to the cloud server

These jobs reach the cloud server and get scheduled by some scheduling technique. Task scheduling on the cloud server is one of the prime tasks on mobile cloud computing. The virtual machines (VM) need to be allocated to the task's execution by the cloud service provider. Major thrust has been given to research in the field of mobile computing by a

framework like Chroma(RPC) [14], Cuckoo(RMI) [6], spectra [13], MAUI [15], Mobicloud [19], and Clonecloud [16]. These are popular frameworks in this cloud computing domain that empower the concept of offloading the task to the cloud server either by task partitioning or considering a complete application for offloading purposes. Various studies have been done in the past, trying to achieve optimization in different objective functions like makespan, energy, quality of service (QoS), load balancing, and cost. The problem of task scheduling has much scope for optimization since of its NP-hard nature. The mobile application consists of many computational tasks represented as nodes, and dependency among these nodes is defined as a cloud. Resources are required in the cloud servers for the execution of these offloaded computational tasks. The availability of these resources needs to be assured by the cloud service providers, and also, the pricing of services may vary from country to country.

The mobile task offloading model consists of two ways to execute the task, i.e., either to offload the task on the cloud server or to execute the task locally on the mobile phone. After the initial task partitioning phase, the decision of offloading is made by the decision engine by gathering various device and network parameters through the profiling process. Now, through a cellular network or Wi-Fi network, the task reaches the cloud server. The objective of offloading is to transfer the computation to the resourceful server at a distant place to improve the device's performance and save energy. Taking the offload decision to a remote server is not always mandatory but depends on the various parameters affecting the device's performance. In some scenarios, partial offloading is also performed. One part of the application task is processed on a mobile device, and the other is offloaded to the surrogate or cloud server. The task's computation time depends on the computation amount required and the mobile device's processing speed. In a scenario, let assume the job is divided into two partitions where the first partition executes locally and the second partition runs on a remote server. For local execution, let  $C_{T\_LOCAL}$  be the computation time required on the local device,  $C_{A\_LOCAL}$  is the computation amount, and  $P_{S\_LOCAL}$  is the mobile device's processing speed. The relationship among these values will be:

$$C_{T\_LOCAL} = C_{A\_LOCAL} / P_{S\_LOCAL} \quad (6.1)$$

For remote execution, the second partition is executed on the cloud/edge server. Let  $B_{AVAILABLE}$  be the available bandwidth in the device and the amount of data to be transferred be  $D_{AMOUNT\_OF\_DATA}$ . The time taken to transfer the data to/from the server will be  $C_{T\_REMOTE}$  will be

$$C_{T\_REMOTE} = D_{AMOUNT\_OF\_DATA} / B_{AVAILABLE} + \text{Cloud processing time} \quad (6.2)$$

Total time  $C_{T\_TOTAL}$  taken to execute the application both locally and remotely will be a summation of the above two equations, which is

$$C_{T\_TOTAL} = C_{T\_LOCAL} + C_{T\_REMOTE} \quad (6.3)$$

When a task is transferred from a mobile device to a cloud server, it is delivered to the cloud service provider's server. The cloud service provider manages all information about the task that approached it for processing. The Datacenter Broker policy [11] helps the cloudlets (task) to assign the virtual machines. The data center policy must be appropriate for the minimum execution time of the cloudlet. Similar to web applications, a mobile application consists of different tasks. These tasks can be represented as a directed acyclic graph (DAG). While the application's independent task can be executed simultaneously in multiple virtual machines, the dependent job needs to be synchronized as per their precedence order.

### 6.2.2 Multi-objective approach of task scheduling

The multi-objective approach focuses on optimizing more than one objective function simultaneously. Energy and makespan are considered objective functions for the study. A task offloading scheme is based on optimizing the multi-objective function, where minimizing both functions is the approach's actual goal.

Makespan is defined as the time required for the processing of the task CPU and its transmission time. The makespan of a task on the virtual machine is calculated considering the computing power of the VM and the size of the task. It can be defined the following equation:

$$\text{Makespan}(T) = \text{size of the task} / \text{computational power} \quad (6.4)$$

Two factors calculate energy cost: virtual machine usage charges, which are usually different for cloud service providers, and calculated on a second basis. The other is the execution time of the task. It can be defined the following equation:

$$\text{Energy Cost} = \text{execution time} \times \text{virtual machine usage charge} \quad (6.5)$$

### **6.3 Proposed approach for task scheduling model**

The proposed approach (GMOPSO-BFO) is based on a hybrid approach of particle swarm optimization (PSO) and Bacteria foraging optimization (BFO). The PSO approach works excellent in searching the solution globally, whereas the BFO works optimally with local search capabilities. The combined approach of these two techniques generates an optimal solution globally and locally in search capability and higher convergence time.

In this work, the particular task's execution time depends on the task size and the virtual machine's property. Following are the basic definitions regarding mobile task scheduling:

a) Consider a set of  $n$  virtual machines as  $V = \{V_1, V_2, V_3, \dots, V_n\}$

b) A task of the application tasks  $T = \{T_1, T_2, \dots, T_x\}$

c)  $E$  is the set of connections between any two tasks,  $T_i$  and  $T_j$ .

d) Collection of physical machines (PMs) in the data center =  $(PM_1, PM_2, PM_3, \dots, PM_n)$

It is assumed in the work that the cloud service provider has a sufficient number of computational resources. The  $V$  number of virtual machines are deployed on the physical machines, and different virtual machines have a variety of processing units (CPU), random access memory (RAM), and networking capabilities. The data center brokers monitor all available resources and assign the machine to the task once approached. All jobs requiring the processing resources need to stand in a queue and based on the task scheduling scheme, tasks are planned to execute on the machine.

#### **6.3.1 Bacteria Foraging Optimization**

The bacteria foraging method is a natural selection method in which microorganisms like bacteria tend to search or forage the food to survive in the E-coli (intestine) of the human



body [141]. The primary strategy of bacteria to survive is by locating the nutrients, handling them, and ingesting the food to get the energy to live and reproduce. Those bacteria which do not successfully forage the nutrient typically get eliminated from the system. It follows the concept of survival for the fittest. This evolutionary concept made the scientist fascinated and motivated them to use it as an optimization process. Most of the optimization processes can be performed with such an evolutionary approach. The main aim of the bacteria is to maximize the energy attained during foraging per unit time. It depends on certain factors like prey density in the environment and characteristics of the environment. It also depends on the sensing and cognitive capabilities of the bacteria.

The E. coli bacteria have a cell structure having various biological features like nucleoid, ribosomes, cytoplasm, pilus, and plasma membrane. As these attributes do normal cell process, another critical feature, i.e., flagellum, helps bacteria propagate or move in different directions. Chemotaxis is the process of movement of the organism from its position in the presence of some chemical attractants and repellents. With the help of flagella, there are two possible movements, i.e., either its moves clockwise or tumble, and the other is counter clockwise or swim. Fig. 6-3 depicts the movement of bacteria like tumbling and swimming in the E. coli. In a favourable condition of the environment where sufficient nutrients are available and the non-acidic and non-alkaline nature of the intestine, it swims and the opposite of it. It tumbles typically, which is changing the direction of the swim.

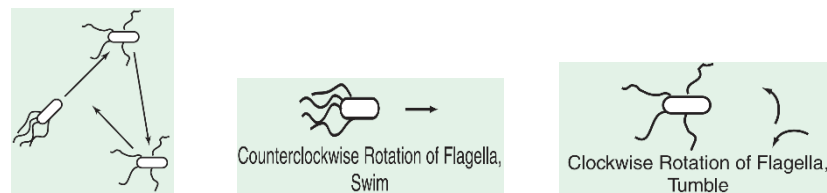


Figure 6-3 Chemotaxis process of the bacteria [141]

The other significant process related to bacteria is swarming, where bacteria release some attractants to swarm together, searching for food. If the attractants are released high and deep, there are chances that different bacteria explore food together; otherwise, they go

alone in the reverse situation. During the reproduction process, the bacteria get split into two parts to increase their population. Bacteria reproduce based upon the nutrient available in the bacteria or the fitness function. The bacteria also go through the elimination and dispersal phase in their lifetime due to their local environment. Sometimes, the condition to survive gets reduced when the sudden rise in heat or nutrients is finished. In terms of computing, to avoid trapping in the local optima, the elimination and dispersal process is used.

### 6.3.2 Particle swarm optimization (PSO)

Particle swarm optimization (PSO) is a nature-inspired algorithm [142] [143] based on social behavior and a flock of birds' dynamic movement. A group of birds known as swarm moves together, searching for food in a particular direction and different velocities. Each bird or particle looks for food and is usually followed by other birds. These birds communicate with each other during their search and typically follow each other closer to the food. The closeness from the food is calculated as a fitness value after a periodic interval of time. Each bird in the swarm is represented as a particle in multidimensional space with a certain velocity and position. Each particle keeps two things in its memory, i.e., their own best position pbest and other is the global best position of gbest of their group. In the standard PSO, the velocity of the particle is updated with the equation

$$v_i^{(k+1)} = \left[ \omega v_i^{(k)} + c_1 \xi_1 (b_i^{(k)} - x_i^{(k)}) + c_2 \xi_2 (y^{(k)} - x_i^{(k)}) \right] \quad (6.6)$$

where  $v_i^{(k+1)}$  is its velocity  $x_i^{(k)}$  is the  $i^{\text{th}}$  particle's position at step  $k$ ,  $b_i^{(k)}$  is the best position visited by the  $i^{\text{th}}$  particle,  $y^{(k)}$  is the overall best position ever visited,  $\omega$  is inertia- weight and  $\xi_1, \xi_2$  are the random numbers between 0 and 1 and  $c_1, c_2$  are the acceleration coefficients. Velocity is updated by inertia, cognitive and social behaviour of the particle. The updated version of PSO, which improve the convergence rate, was constriction factor where the velocity vector as

$$v_i^{(k+1)} = \chi \left[ v_i^{(k)} + c_1 \xi_1 (b_i^{(k)} - x_i^{(k)}) + c_2 \xi_2 (y^{(k)} - x_i^{(k)}) \right] \quad (6.7)$$

$$x_i^{(k+1)} = x_i^{(k)} + v_i^{(k+1)} \quad (6.8)$$

where  $\chi$  is a constriction factor in the above equation and  $x_i^{(k)}$  is the  $i^{\text{th}}$  particle's position at step  $k$ ,  $v_i^{(k+1)}$  is its velocity,  $b_i^{(k)}$  is the best position visited by the  $i^{\text{th}}$  particle,  $y^{(k)}$  is the overall best position ever visited. It has been observed that after incorporating the Gaussian density function in the above equation, the results come better in terms of the global solution. The updated velocity equation will be

$$v_i^{(k+1)} = \left[ \text{randn} \mid (b_i^{(k)} - x_i^{(k)}) + \text{Randn} \mid (y^{(k)} - x_i^{(k)}) \right] \quad (6.9)$$

where the randn and Randn are based on the Gaussian density function's absolute value.

The Gaussian random density function is represented by

$$f(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2} \quad (6.10)$$

---

### **Pseudocode 6.1 GMOPSO-BFO approach for task scheduling:**

---

Initialize the Bacteria Foraging Optimization (BFO) parameters and Particle swarm optimization (GMOPSO) parameters:

$N_p, N_c, S_l, N_r, N_e, C, P_{\text{dispersal}}, d_{\text{attract}}, W_{\text{attract}}, h_{\text{attract}}, W_{\text{attract}}, p_i, f, v_i$

**Input:** a collection of all bacteria where each bacteria represented as  $\theta^i(j, k, l)$

**Output:** a collection of information on how much these bacteria collect nutrients

1. **begin:** Let  $\theta^i(j, k, l)$  be the position of the  $i^{\text{th}}$  bacteria in the environment where  $j$  defines the chemotaxes step,  $k$  defines the reproduction step, and  $l$  defines the dispersal elimination step.
2. for all bacteria in the list:

3. **Loop** elimination-dispersal step
4. **Loop** reproduction step
5. **Loop** Chemotaxis step
6. go for chemotactic steps using (a) and (b), respectively
7. Initialize the value of  $v_i$  and position  $p_i$  of the  $i^{\text{th}}$  bacteria
- (a) Compute tumbling step:

$$\theta^i(j+1, k, l) = \theta^i(j, k, l) + C(i) \frac{dt(i)}{\sqrt{dt^T(i)dt(i)}}$$

- (b) Compute Swim step:

$$J(i, j, k, l) = J(i, j, k, l) + J_{cc}(\theta^i(j, k, l), P(j, k, l))$$

8. Set  $J_{\text{last}} = J(i, j, k, l)$
9. If  $J(i, j+1, k, l) < J_{\text{last}}$
10. Update  $J_{\text{last}}$
11. For the reproduction phase: calculate the fitness function using:

$$J_{\text{health}}^i = \sum_{j=1}^{N_c+1} J(i, j, k, l)$$

12. Sort in ascending order the bacteria and chemotactic parameters  
If  $(k < N_r)$ , perform reproduction step again till  $k = N_r$
13. For elimination and dispersal:
14. for each bacteria,
15. if  $(p_{\text{ed}} < P_{\text{dispersal}})$ ,

16. do elimination and dispersal till  $l = N_e$
17. Do Mutation of the remaining bacteria (particles) using PSO scheme
18. Update  $p_i, best$  and  $g_i, best$  upon meeting the condition

$$\begin{aligned}
 p_{i,best} &= p_i && \text{if } f(p_i) > f(p_{i,best}) \\
 g_{i,best} &= g_i && \text{if } f(g_i) > f(g_{i,best})
 \end{aligned}$$

19. Update velocity of each bacteria (particle) after every iteration by the Gaussian based velocity-

$$v_i^{(k+1)} = \left[ \text{randn} | (b_i^{(k)} - x_i^{(k)}) + \text{Randn} | (y^{(k)} - x_i^{(k)}) \right]$$

20. Update position of each bacteria (particle) after every iteration by the formula-

$$x_i^{(k+1)} = x_i^{(k)} + v_i^{(k+1)}$$

21. Check  $p_i$ , which should exist within the range
22. Repeat step reproduction and PSO until convergence is achieved.
23. After the stopping criteria are met, the value of  $g_{best}$  and  $f(g_{best})$  must be recorded.
24. End

## 6.4 Performance and Evaluation

### 6.4.1 Experimental setup

The proposed approach has been developed in the language Python in the window 10 environment on Intel (R) Core (TM) i5, 1.80 GHz, CPU 8 GB. Various parameters considered during the simulation of the proposed technique have been presented in table 6-1. In evaluating the proposed method, five virtual machines are considered, and a collection of tasks is assumed between 100 and 1000. The results are compared with the existing work on MOPSO [144] and BFO [145] regarding the energy efficiency and makespan of the task execution. The proposed scheme is based Gaussian swarm approach

implemented in MOPSO along with the BFO. The experiment has been performed by considering the number of bacteria ( $Np$ ) as 20 and No\_of\_chemotactics ( $Nc$ ) as 10. In the same way, the initial size of PSO is considered as 20 in the experiment. The experiment runs iteratively about ten times to find the average of makespan and energy values. The experiment has been performed by considering  $m$  random task to  $n$  virtual machine. The task size and required execution time are uniformly distributed. It has been found that the Gaussian scheme has outperformed the standard PSO and increases the convergence ability of PSO. Since our problem is multi-objective, when Gaussian is implemented with MOPSO and BFO, it gives better results in energy efficacy and reduced makespan time. Both factors are required for the offloading problem in mobile cloud computing.

Table 6-1 Parameters considered in the simulation

Parameters for BFO and PSO	Value Used
No_of_bacteria ( $Np$ )	20
No_of_chemotactics ( $Nc$ )	10
swim_length ( $Sl$ )	4
No_of_reproductions ( $Nr$ )	4
No_of_dispersals ( $Ne$ )	2
step_size ( $C$ )	1.45
probability_dispersal ( $P$ dispersal)	0.25
d_attractant ( $d_{attract}$ )	0.1
w_attractant ( $w_{attract}$ )	0.2
h_repellant ( $h_{attract}$ )	0.1
w_repellant ( $w_{attract}$ )	10
PSO Swarm size	20
Self-recognition coefficient	1

Social coefficient	2
Inertial weight	0.5

### 6.4.2 Results and Discussion

Table 6-2 presents the various task execution times, and it can be seen that the GMOPSO-BFO approach has performed better than the other algorithms. As the number of tasks increases on the virtual machine, the proposed scheme maintains the lowest makespan. The proposed scheme has less makespan for the various range of tasks from 100 to 1000 compared to MOPSO, BFO, and MOPSO-BFO.

Table 6-2 Execution time of the task in different techniques

TASK	MOPSO	BFO	PSO-BFO	GMOPSO-BFO
<b>100</b>	41.47	38.66	37.65	37.18
<b>200</b>	155	151.81	155.05	145.25
<b>300</b>	345.4	335.53	335	327.38
<b>400</b>	594.85	597.26	594.85	567.8
<b>500</b>	926.43	916.66	913.98	878.55
<b>600</b>	1332.22	1333.36	1324.33	1284.5
<b>700</b>	1813.15	1885.31	1803.56	1750.51
<b>800</b>	2349.87	2390.75	2310.6	2316.66
<b>900</b>	2934.87	3034.93	2924.65	2916.73
<b>1000</b>	3743.78	3692.85	3655.96	3618.93

In this work, the energy consumption is calculated of the proposed GMOPSO-BFO technique and compared with the method like MOPSO, BFO, and MOPSO-BFO. In this experiment, the number of virtual machines is considered 5, and the number of tasks ranges

from 100 to 1000. The experiment aimed to determine the energy consumption of the various techniques on virtual machines. The unit of the energy consumption is considered as joules/minute.

Table 6-3 Energy consumption of the task in different techniques

<b>TASK</b>	<b>MOPSO</b>	<b>BFO</b>	<b>PSO-BFO</b>	<b>GMOPSO-BFO</b>
<b>100</b>	1.05	1.05	1.03	1.03
<b>200</b>	2.15	2.15	2.15	2.14
<b>300</b>	3.14	3.12	3.16	3.11
<b>400</b>	4.15	4.13	4.15	4.12
<b>500</b>	5.18	5.19	5.17	5.18
<b>600</b>	6.33	6.33	6.3	6.18
<b>700</b>	7.45	7.46	7.5	7.42
<b>800</b>	8.49	8.47	8.46	8.45
<b>900</b>	9.6	9.62	9.61	9.59
<b>1000</b>	10.72	10.75	10.71	10.66

Table 6-3 presents the various tasks on the virtual machine and GMOPSO-BFO approach, which has consumed less energy in joules than the other algorithms. It has been observed that when the number of tasks increases from 100 to 1000, the machine's energy consumption also increases. The proposed schemes perform better as compared to the other algorithm. The proposed scheme is able to save energy consumption in the virtual machine. It is clear from the experimental results that the proposed scheme GMOPSO-BFO performs better in completion time and energy consumption.



## **6.5 Summary**

The chapter proposed a hybrid scheduling technique based on Gaussian-based multi-objective particle swarm optimization (GMOPSO) and Bacterial foraging optimization (BFO). The GMOPSO provides us the global best solution, whereas using the BFO, the local best solution is tried to be improvised. The work methodology and the detailed design approach of the suggested scheduling system are presented in the chapter. The evaluation results compared with existing works are also discussed in the chapter.

---

## CHAPTER 7

### CONCLUSION AND FUTURE WORK

---

#### 7.1 Conclusion and Discussion

Code partitioning is the foremost important task in computation offloading. In this study, a heuristic has been developed based on a multilevel hybrid approach for balanced graph partitioning. Although perfect partitioning in the mobile cloud computing offloading process is a challenging task, it can be achieved by optimizing the partitions at different levels. Heavy edge matching is performed on the graph initially for coarsening, and then spectral graph partitioning is applied for the initial partitioning of the graph. In the last stage, the Kernighan Lin algorithm is used for the refinement of the partitioned graph. The spectral partitioning along with the Kernighan Lin algorithm has performed optimally as compared to the existing approach of random partitioning and the multilevel KL approach in terms of edge cuts. The spectral approach, along with KL, improves the edge cut size, which plays a key parameter in the communication between the client device and the cloud side. The initial design of the application partitioning scheme for mobile applications aims to develop an optimal code partitioning scheme and focused to optimize the edge cut. It has been aimed to minimize the number of edge cuts that minimize the communication between partitions. In this work, there are two partitions of the application where one partition resides on the mobile site and the other on the cloud server. Scheme is not applicable for multi-site offloading method. The energy consumption during the offloading process of partitioned task is not considered in the scope of the work and will be explored in the future work.

A decision engine is a significant component in the offloading framework, which helps decide when to offload the task to the remote or cloud server. This work suggested a stack-based classification method that can perform the offloading decision-making task of the decision engine in the computational offloading process. The accuracy of the proposed classifier is found better when compared with different machine learning models. The proposed technique's ROC curve is presented, which has better TPR compared to other models. The feature of mobility is also considered in work based on the GPS of the device. The lower execution time and lower CPU utilization make the proposed scheme a viable decision engine approach.

A mobility-based offloading scheme in MCC has been proposed in this work where the emphasis is given to reduce the number of handoff of the mobile device and also the handoff dropped count. The 4th order Markov model has been devised for predicting the next location of the user equipment. The technique will enable the user equipment to remain connected to the cellular network or Wi-Fi network, which at the end gets connected to the cloud or edge server for completing the computational offloading task. The work includes the various mobility environments like urban, semi-urban, and rural for implementing the proposed work, and it has found that it has performed better than the compared SINR based approach. In this work, the cloud server is considered to be stationary and connected to the cellular service in the current scenario and in case of tie-breaking situation between two signals having same signal strengths during network selection, mobile device will select any network in random order.

The jobs reach the cloud server during offloading and get scheduled by some scheduling technique. Task scheduling on the cloud server is one of the prime tasks on mobile cloud computing. This work presents a hybrid scheduling approach based upon the Gaussian multi-objective particle swarm optimization and bacteria foraging optimization. Both makespan and energy consumption are essential factors in offloading method of MCC. The proposed scheme performs better in makespan and energy consumption. The results are compared with the MOPSO, BFO, and hybrid MOPSO-BFO. The scheme leverages the global optima of GMOPSO and the local optima by BFO.

## 7.2 Future scope

- In the future, the approach will be implemented in the real software design in the mobile application, and further investigation will be carried out considering different parameters in the account. Partitioning of an application during offloading in mobile cloud computing is a critical problem. Different heuristics can be developed considering various parameters during the study. In this work, the mobility of the device is not affecting the partitioning results. It can be an open area where researchers can work and develop better models based on mobility.
- Different parameters like data privacy, data confidentiality, and device security will be explored for decision-making in the future. Unsupervised learning, Reinforcement learning, and the evolutionary-based scheme will be studied and used to make the decision-making techniques during offloading.
- 5G and 6G networks will be explored in coming future work, where the handoff mechanism will be studied and implemented for providing relevant ground for the computational offloading process in mobile cloud computing. Further, cloud probing strategies will be devised for finding the best cloud services during offloading.
- A scheduling scheme will be developed based on other optimization parameters like a load on the servers, scalability, latency, and resource utilization. The multi-site scheduling will also be explored to offload the task on cloud servers, and task dependency will be managed in such schemes of scheduling. Resource allocation schemes on cloud server will be explored since in current work, only task scheduling scheme has been worked upon.

---

## REFERENCES

- [1] "Ericsson Mobility Report 2020," <https://www.ericsson.com/en/mobility-report/reports>.
- [2] A. Zaslavsky and Z. Tari, "Mobile computing: Overview and current status," *J. Res. Pract. Inf. Technol.*, vol. 30, no. 2, pp. 42–52, 1998.
- [3] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Futur. Gener. Comput. Syst.*, vol. 25, no. 6, pp. 599–616, 2009, doi: 10.1016/j.future.2008.12.001.
- [4] H. T. Dinh, C. Lee, D. Niyato, and P. Wang, "A survey of mobile cloud computing: architecture, applications, and approaches," *Wirel. Commun. Mob. Comput.*, vol. 13, no. 18, pp. 1587–1611, Dec. 2013, doi: 10.1002/wcm.1203.
- [5] A. Rudenko, P. Reiher, G. J. Popek, and G. H. Kuenning, "Saving portable computer battery power through remote process execution," *ACM SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 2, no. 1, pp. 19–26, Jan. 1998, doi: 10.1145/584007.584008.
- [6] R. Kemp, N. Palmer, T. Kielmann, and H. Bal, "Cuckoo: A computation offloading framework for smartphones," in *Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, LNICST*, vol. 76 LNICST, 2012, pp. 59–79.
- [7] H. Flores, P. Hui, S. Tarkoma, Y. Li, S. Srirama, and R. Buyya, "Mobile code offloading: from concept to practice and beyond," *IEEE Commun. Mag.*, vol. 53, no. 3, pp. 80–88, Mar. 2015, doi: 10.1109/MCOM.2015.7060486.
- [8] Y. Ge, Y. Zhang, Q. Qiu, and Y.-H. Lu, "A game theoretic resource allocation for overall energy minimization in mobile cloud computing system," in *Proceedings of the 2012 ACM/IEEE international symposium on Low power electronics and design - ISLPED '12*, 2012, p. 279, doi: 10.1145/2333660.2333724.

- [9] P. Bahl, R. Y. Han, L. E. Li, and M. Satyanarayanan, “Advancing the state of mobile cloud computing,” in *Proceedings of the third ACM workshop on Mobile cloud computing and services - MCS '12*, 2012, p. 21, doi: 10.1145/2307849.2307856.
- [10] N. Fernando, S. W. Loke, and W. Rahayu, “Mobile cloud computing: A survey,” *Futur. Gener. Comput. Syst.*, vol. 29, no. 1, pp. 84–106, Jan. 2013, doi: 10.1016/j.future.2012.05.023.
- [11] S. Singh and I. Chana, “A Survey on Resource Scheduling in Cloud Computing: Issues and Challenges,” *J. Grid Comput.*, vol. 14, no. 2, pp. 217–264, Jun. 2016, doi: 10.1007/s10723-015-9359-2.
- [12] E. E. Marinelli, “Hyrax : Cloud Computing on Mobile Devices,” vol. 0389, no. September, 2009.
- [13] J. Flinn, SoYoung Park, and M. Satyanarayanan, “Balancing performance, energy, and quality in pervasive computing,” in *Proceedings 22nd International Conference on Distributed Computing Systems*, pp. 217–226, doi: 10.1109/ICDCS.2002.1022259.
- [14] R. K. Balan, M. Satyanarayanan, S. Y. Park, and T. Okoshi, “Tactics-based remote execution for mobile computing,” in *Proceedings of the 1st international conference on Mobile systems, applications and services - MobiSys '03*, 2003, pp. 273–286, doi: 10.1145/1066116.1066125.
- [15] E. Cuervo *et al.*, “MAUI,” in *Proceedings of the 8th international conference on Mobile systems, applications, and services - MobiSys '10*, 2010, p. 49, doi: 10.1145/1814433.1814441.
- [16] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, “CloneCloud,” in *Proceedings of the sixth conference on Computer systems - EuroSys '11*, 2011, p. 301, doi: 10.1145/1966445.1966473.
- [17] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, “The Case for VM-Based Cloudlets in Mobile Computing,” *IEEE Pervasive Comput.*, vol. 8, no. 4, pp. 14–23, Oct. 2009, doi: 10.1109/MPRV.2009.82.
- [18] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, “ThinkAir: Dynamic

- resource allocation and parallel execution in the cloud for mobile code offloading,” *Proc. - IEEE INFOCOM*, pp. 945–953, 2012, doi: 10.1109/INFOCOM.2012.6195845.
- [19] D. Huang, X. Zhang, M. Kang, and J. Luo, “MobiCloud: Building Secure Cloud Framework for Mobile Computing and Communication,” in *2010 Fifth IEEE International Symposium on Service Oriented System Engineering*, Jun. 2010, pp. 27–34, doi: 10.1109/SOSE.2010.20.
- [20] M. Daro Kristensen, “Scavenger: Transparent development of efficient cyber foraging applications,” in *2010 IEEE International Conference on Pervasive Computing and Communications (PerCom)*, Mar. 2010, pp. 217–226, doi: 10.1109/PERCOM.2010.5466972.
- [21] M. Satyanarayanan, “Pervasive computing: vision and challenges,” *IEEE Pers. Commun.*, vol. 8, no. 4, pp. 10–17, 2001, doi: 10.1109/98.943998.
- [22] R. K. K. Ma, K. T. Lam, C. L. Wang, and C. Zhang, “A stack-on-demand execution model for elastic computing,” *Proc. Int. Conf. Parallel Process.*, pp. 208–217, 2010, doi: 10.1109/ICPP.2010.79.
- [23] E. Ahmed, A. Gani, M. Khurram Khan, R. Buyya, and S. U. Khan, “Seamless application execution in mobile cloud computing: Motivation, taxonomy, and open challenges,” *J. Netw. Comput. Appl.*, vol. 52, pp. 154–172, 2015, doi: 10.1016/j.jnca.2015.03.001.
- [24] D. Kovachev and R. Klamma, “Framework for Computation Offloading in Mobile Cloud Computing,” *Int. J. Interact. Multimed. Artif. Intell.*, vol. 1, no. 7, p. 6, 2012, doi: 10.9781/ijimai.2012.171.
- [25] S. H. Hung, J. P. Shieh, and Y. W. Chen, “A profile-driven dynamic application offloading scheme for Android systems,” *1st IEEE Glob. Conf. Consum. Electron. 2012, GCCE 2012*, pp. 540–541, 2012, doi: 10.1109/GCCE.2012.6379903.
- [26] S. Yang, “Manageable granularity in mobile application code offloading for energy savings,” *Proc. - 2012 IEEE Int. Conf. Green Comput. Commun. GreenCom 2012, Conf. Internet Things, iThings 2012 Conf. Cyber, Phys. Soc. Comput. CPSCom*

- 2012, pp. 611–614, 2012, doi: 10.1109/GreenCom.2012.93.
- [27] P. Balakrishnan and C. K. Tham, “Energy-efficient mapping and scheduling of task interaction graphs for code offloading in mobile cloud computing,” *Proc. - 2013 IEEE/ACM 6th Int. Conf. Util. Cloud Comput. UCC 2013*, pp. 34–41, 2013, doi: 10.1109/UCC.2013.23.
- [28] A. Mtibaa, K. A. Harras, and A. Fahim, “Towards computational offloading in mobile device clouds,” *Proc. Int. Conf. Cloud Comput. Technol. Sci. CloudCom*, vol. 1, pp. 331–338, 2013, doi: 10.1109/CloudCom.2013.50.
- [29] H. Qian and D. Andresen, “Jade: Reducing energy consumption of android app,” *Int. J. Networked Distrib. Comput.*, vol. 3, no. 3, pp. 150–158, 2015, doi: 10.2991/ijndc.2015.3.3.2.
- [30] M. P. Anastasopoulos, A. Tzanakaki, and D. Simeonidou, “Energy-aware offloading in mobile cloud systems with delay considerations,” *2014 IEEE Globecom Work. GC Wkshps 2014*, pp. 42–47, 2014, doi: 10.1109/GLOCOMW.2014.7063383.
- [31] M. B. Terefe, H. Lee, N. Heo, G. C. Fox, and S. Oh, “Energy-efficient multisite offloading policy using Markov decision process for mobile cloud computing,” *Pervasive Mob. Comput.*, vol. 27, pp. 75–89, 2015, doi: 10.1016/j.pmcj.2015.10.008.
- [32] X. Zhang, A. Kunjithapatham, S. Jeong, and S. Gibbs, “Towards an elastic application model for augmenting the computing capabilities of mobile devices with cloud computing,” *Mob. Networks Appl.*, vol. 16, no. 3, pp. 270–284, 2011, doi: 10.1007/s11036-011-0305-7.
- [33] Y. Zhang, D. Niyato, and P. Wang, “Offloading in Mobile Cloudlet Systems with Intermittent Connectivity,” *IEEE Trans. Mob. Comput.*, vol. 14, no. 12, pp. 2516–2529, 2015, doi: 10.1109/TMC.2015.2405539.
- [34] M. Chen, Y. Hao, M. Qiu, J. Song, D. Wu, and I. Humar, “Mobility-aware caching and computation offloading in 5G ultra-dense cellular networks,” *Sensors (Switzerland)*, vol. 16, no. 7, pp. 1–13, 2016, doi: 10.3390/s16070974.
- [35] M. V. Barbera, S. Kosta, A. Mei, and J. Stefa, “To offload or not to offload? the



- bandwidth and energy costs of mobile cloud computing,” *Proc. - IEEE INFOCOM*, pp. 1285–1293, 2013, doi: 10.1109/INFOCOM.2013.6566921.
- [36] M. S. Gordon, D. Anoushe Jamshidi, S. Mahlke, Z. Morley Mao, and X. Chen, “CoMET: Code offload by migrating execution transparently,” *Proc. 10th USENIX Symp. Oper. Syst. Des. Implementation, OSDI 2012*, pp. 93–106, 2012.
- [37] H. Qian and D. Andresen, “Automate scientific workflow execution between local cluster and cloud,” *Int. J. Networked Distrib. Comput.*, vol. 4, no. 1, pp. 45–54, 2016, doi: 10.2991/ijndc.2016.4.1.5.
- [38] D. Lima, H. Miranda, and F. Taïani, “Towards a new model for cyber foraging,” *Proc. 13th Work. Adapt. Reflective Middleware, ARM 2014, co-located with ACM/IFIP/USENIX Int. Middlew. Conf.*, 2014, doi: 10.1145/2677017.2677023.
- [39] E. Cuervoy *et al.*, “MAUI: Making smartphones last longer with code offload,” *MobiSys’10 - Proc. 8th Int. Conf. Mob. Syst. Appl. Serv.*, pp. 49–62, 2010, doi: 10.1145/1814433.1814441.
- [40] H. Wu, W. Knottenbelt, K. Wolter, and Y. Sun, “An optimal offloading partitioning algorithm in mobile cloud computing,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 9826 LNCS, pp. 311–328, 2016, doi: 10.1007/978-3-319-43425-4\_21.
- [41] D. Kovachev and R. Klamma, “Framework for Computation Offloading in Mobile Cloud Computing,” *Int. J. Interact. Multimed. Artif. Intell.*, vol. 1, no. 7, p. 6, 2012, doi: 10.9781/ijimai.2012.171.
- [42] M.-R. Ra, B. Priyantha, A. Kansal, and J. Liu, “Improving energy efficiency of personal sensing applications with heterogeneous multi-processors,” in *Proceedings of the 2012 ACM Conference on Ubiquitous Computing - UbiComp ’12*, 2012, p. 1, doi: 10.1145/2370216.2370218.
- [43] M. Goraczko, J. Liu, D. Lymberopoulos, S. Matic, B. Priyantha, and F. Zhao, “Energy-optimal software partitioning in heterogeneous multiprocessor embedded systems,” in *Proceedings of the 45th annual conference on Design automation - DAC ’08*, 2008, p. 191, doi: 10.1145/1391469.1391518.

- [44] K. Sinha and M. Kulkarni, “Techniques for Fine-Grained, Multi-site Computation Offloading,” in *2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, May 2011, pp. 184–194, doi: 10.1109/CCGrid.2011.69.
- [45] M. Smit, M. Shtern, B. Simmons, and M. Litoiu, “Partitioning Applications for Hybrid and Federated Clouds,” 2012, [Online]. Available: <http://www.mikesmit.com/wp-content/papercite-data/pdf/cascon2012.pdf%5Cnpapers2://publication/uuid/8BB68396-C5E5-475D-833B-CC1C08B39FD8>.
- [46] C. Wang and Z. Li, “Parametric analysis for adaptive computation offloading,” in *Proceedings of the ACM SIGPLAN 2004 conference on Programming language design and implementation - PLDI '04*, 2004, p. 119, doi: 10.1145/996841.996857.
- [47] J. Niu, W. Song, and M. Atiquzzaman, “Bandwidth-adaptive partitioning for distributed execution optimization of mobile applications,” *J. Netw. Comput. Appl.*, vol. 37, pp. 334–347, Jan. 2014, doi: 10.1016/j.jnca.2013.03.007.
- [48] T. Verbelen, T. Stevens, F. De Turck, and B. Dhoedt, “Graph partitioning algorithms for optimizing software deployment in mobile cloud computing,” *Futur. Gener. Comput. Syst.*, vol. 29, no. 2, pp. 451–459, 2013, doi: 10.1016/j.future.2012.07.003.
- [49] J. Liu, E. Ahmed, M. Shiraz, A. Gani, R. Buyya, and A. Qureshi, “Application partitioning algorithms in mobile cloud computing: Taxonomy, review and future directions,” *J. Netw. Comput. Appl.*, vol. 48, pp. 99–117, Feb. 2015, doi: 10.1016/j.jnca.2014.09.009.
- [50] G. Karypis and V. Kumar, “A fast and high quality multilevel scheme for partitioning irregular graphs,” *SIAM J. Sci. Comput.*, vol. 20, no. 1, pp. 359–392, 1998, doi: 10.1137/S1064827595287997.
- [51] M. A. Hassan, K. Bhattarai, Q. Wei, and S. Chen, “POMAC: Properly offloading mobile applications to clouds,” *6th USENIX Work. Hot Top. Cloud Comput. HotCloud 2014*, pp. 1–6, 2014.
- [52] B. Gao, L. He, L. Liu, K. Li, and S. A. Jarvis, “From mobiles to clouds: Developing energy-aware offloading strategies for workflows,” *Proc. - IEEE/ACM Int. Work.*

- Grid Comput.*, pp. 139–146, 2012, doi: 10.1109/Grid.2012.20.
- [53] B. Zhou, A. V. Dastjerdi, R. N. Calheiros, S. N. Srirama, and R. Buyya, “MCloud: A Context-Aware Offloading Framework for Heterogeneous Mobile Cloud,” *IEEE Trans. Serv. Comput.*, vol. 10, no. 5, pp. 797–810, 2017, doi: 10.1109/TSC.2015.2511002.
- [54] M. Amoretti, A. Grazioli, and F. Zanichelli, “A modeling and simulation framework for mobile cloud computing,” *Simul. Model. Pract. Theory*, vol. 58, pp. 140–156, 2015, doi: 10.1016/j.simpat.2015.05.004.
- [55] M. E. Khoda, M. A. Razzaque, A. Almogren, M. M. Hassan, A. Alamri, and A. Alelaiwi, “Efficient Computation Offloading Decision in Mobile Cloud Computing over 5G Network,” *Mob. Networks Appl.*, pp. 1–16, 2016, doi: 10.1007/s11036-016-0688-6.
- [56] R. Kemp, N. Palmer, T. Kielmann, and H. Bal, “Cuckoo: A computation offloading framework for smartphones,” *Lect. Notes Inst. Comput. Sci. Soc. Telecommun. Eng. LNICST*, vol. 76 LNICST, pp. 59–79, 2012, doi: 10.1007/978-3-642-29336-8\_4.
- [57] F. Xia, F. Ding, J. Li, X. Kong, L. T. Yang, and J. Ma, “Phone2Cloud: Exploiting computation offloading for energy saving on smartphones in mobile cloud computing,” *Inf. Syst. Front.*, vol. 16, no. 1, pp. 95–111, 2014, doi: 10.1007/s10796-013-9458-1.
- [58] H. Eom, P. St. Juste, R. Figueiredo, O. Tickoo, R. Illikkal, and R. Iyer, “Machine Learning-Based Runtime Scheduler for Mobile Offloading Framework,” in *2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing*, Dec. 2013, pp. 17–25, doi: 10.1109/UCC.2013.21.
- [59] T. Truong-Huu, C. K. Tham, and D. Niyato, “To offload or to wait: An opportunistic offloading algorithm for parallel tasks in a mobile cloud,” *Proc. Int. Conf. Cloud Comput. Technol. Sci. CloudCom*, vol. 2015-Febru, no. February, pp. 182–189, 2015, doi: 10.1109/CloudCom.2014.33.
- [60] E. Hyytia, T. Spyropoulos, and J. Ott, “Offload (only) the right jobs: Robust offloading using the Markov decision processes,” *Proc. WoWMoM 2015 A World*

- Wirel. Mob. Multimed. Networks*, 2015, doi: 10.1109/WoWMoM.2015.7158127.
- [61] K. Lee and I. Shin, “User mobility model based computation offloading decision for mobile cloud,” *J. Comput. Sci. Eng.*, vol. 9, no. 3, pp. 155–162, 2015, doi: 10.5626/JCSE.2015.9.3.155.
- [62] J. Wang, J. Peng, Y. Wei, D. Liu, and J. Fu, “Adaptive application offloading decision and transmission scheduling for mobile cloud computing,” *2016 IEEE Int. Conf. Commun. ICC 2016*, 2016, doi: 10.1109/ICC.2016.7510721.
- [63] J. L. D. Neto, D. F. Macedo, and J. M. S. Nogueira, “Location aware decision engine to offload mobile computation to the cloud,” *Proc. NOMS 2016 - 2016 IEEE/IFIP Netw. Oper. Manag. Symp.*, no. Noms, pp. 543–549, 2016, doi: 10.1109/NOMS.2016.7502856.
- [64] Z. Liu, X. Zeng, W. Huang, J. Lin, X. Chen, and W. Guo, “Framework for context-aware computation offloading in mobile cloud computing,” *Proc. - 15th Int. Symp. Parallel Distrib. Comput. ISPDC 2016*, pp. 172–177, 2017, doi: 10.1109/ISPDC.2016.30.
- [65] P. A. L. Rego, E. Cheong, E. F. Coutinho, F. A. M. Trinta, M. Z. Hasany, and J. N. D. Souza, “Decision Tree-Based Approaches for Handling Offloading Decisions and Performing Adaptive Monitoring in MCC Systems,” *Proc. - 5th IEEE Int. Conf. Mob. Cloud Comput. Serv. Eng. MobileCloud 2017*, pp. 74–81, 2017, doi: 10.1109/MobileCloud.2017.19.
- [66] H. Ko, J. Lee, and S. Pack, “Spatial and Temporal Computation Offloading Decision Algorithm in Edge Cloud-Enabled Heterogeneous Networks,” *IEEE Access*, vol. 6, pp. 18920–18932, 2018, doi: 10.1109/ACCESS.2018.2818111.
- [67] A. Ravi and S. K. Peddoju, “Mobile computation bursting-An application partitioning and offloading decision engine,” *ACM Int. Conf. Proceeding Ser.*, pp. 1–10, 2018, doi: 10.1145/3154273.3154299.
- [68] W. Zhou, W. Fang, Y. Li, B. Yuan, Y. Li, and T. Wang, “Markov Approximation for Task Offloading and Computation Scaling in Mobile Edge Computing,” *Mob. Inf. Syst.*, vol. 2019, pp. 1–12, 2019, doi: 10.1155/2019/8172698.

- [69] Q. Qi *et al.*, “Knowledge-Driven Service Offloading Decision for Vehicular Edge Computing: A Deep Reinforcement Learning Approach,” *IEEE Trans. Veh. Technol.*, vol. 68, no. 5, pp. 4192–4203, 2019, doi: 10.1109/TVT.2019.2894437.
- [70] S. Misra, B. E. Wolfinger, M. P. Achuthananda, T. Chakraborty, S. N. Das, and S. Das, “Auction-Based Optimal Task Offloading in Mobile Cloud Computing,” *IEEE Syst. J.*, vol. 13, no. 3, pp. 2978–2985, Sep. 2019, doi: 10.1109/JSYST.2019.2898903.
- [71] S. T. Manukumar and V. Muthuswamy, “A Novel Multi-Objective Efficient Offloading Decision Framework in Cloud Computing for Mobile Computing Applications,” *Wirel. Pers. Commun.*, vol. 107, no. 4, pp. 1625–1642, Aug. 2019, doi: 10.1007/s11277-019-06348-4.
- [72] X. Jin, Z. Wang, and W. Hua, “Cooperative Runtime Offloading Decision Algorithm for Mobile Cloud Computing,” *Mob. Inf. Syst.*, vol. 2019, pp. 1–17, Sep. 2019, doi: 10.1155/2019/8049804.
- [73] A. Elhosuieny, M. Salem, A. Thabet, and A. Ibrahim, “ADOMC-NPR Automatic Decision-Making Offloading Framework for Mobile Computation Using Nonlinear Polynomial Regression Model,” *Int. J. Web Serv. Res.*, vol. 16, no. 4, pp. 53–73, Oct. 2019, doi: 10.4018/IJWSR.2019100104.
- [74] A. Shahidinejad and M. Ghobaei-Arani, “Joint computation offloading and resource provisioning for e <scp>dge-cloud</scp> computing environment: A machine learning-based approach,” *Softw. Pract. Exp.*, vol. 50, no. 12, pp. 2212–2230, Dec. 2020, doi: 10.1002/spe.2888.
- [75] T. Mshvidobadze, “Evolution mobile wireless communication and LTE networks,” in *2012 6th International Conference on Application of Information and Communication Technologies (AICT)*, Oct. 2012, pp. 1–7, doi: 10.1109/ICAICT.2012.6398495.
- [76] N. Nasser, A. Hasswa, and H. Hassanein, “Handoffs in fourth generation heterogeneous networks,” *IEEE Commun. Mag.*, vol. 44, no. 10, pp. 96–103, Oct. 2006, doi: 10.1109/MCOM.2006.1710420.

- [77] K. Mitra, S. Saguna, C. Ahlund, and D. G. Lulea, “M2C2: A mobility management system for mobile cloud computing,” in *2015 IEEE Wireless Communications and Networking Conference (WCNC)*, Mar. 2015, pp. 1608–1613, doi: 10.1109/WCNC.2015.7127708.
- [78] Q. Qi, J. Liao, and Y. Cao, “Integrated multiple services handoff in mobile cloud computing,” in *Global Information Infrastructure Symposium - GIIS 2013*, Oct. 2013, pp. 1–3, doi: 10.1109/GIIS.2013.6684380.
- [79] A. S. Alnezari and N.-E. Rikli, “Achieving Mobile Cloud Computing through Heterogeneous Wireless Networks,” *Int. J. Commun. Netw. Syst. Sci.*, vol. 10, no. 06, pp. 107–128, 2017, doi: 10.4236/ijcns.2017.106006.
- [80] Q. Bani Hani and J. P. Dichter, “Energy-efficient service-oriented architecture for mobile cloud handover,” *J. Cloud Comput.*, vol. 6, no. 1, p. 9, Dec. 2017, doi: 10.1186/s13677-017-0079-y.
- [81] T. Ali and M. Saquib, “Analysis of an Instantaneous Packet Loss Based Vertical Handover Algorithm for Heterogeneous Wireless Networks,” *IEEE Trans. Mob. Comput.*, vol. 13, no. 5, pp. 992–1006, May 2014, doi: 10.1109/TMC.2013.42.
- [82] A. Sgora, C. A. Gizelis, and D. D. Vergados, “Network selection in a WiMAX–WiFi environment,” *Pervasive Mob. Comput.*, vol. 7, no. 5, pp. 584–594, Oct. 2011, doi: 10.1016/j.pmcj.2010.10.001.
- [83] Z. Sanaei, S. Abolfazli, A. Gani, and R. Buyya, “Heterogeneity in Mobile Cloud Computing: Taxonomy and Open Challenges,” *IEEE Commun. Surv. Tutorials*, vol. 16, no. 1, pp. 369–392, 2014, doi: 10.1109/SURV.2013.050113.00090.
- [84] D. Bhattacharjee, A. Rao, C. Shah, M. Shah, and A. Helmy, “Empirical modeling of campus-wide pedestrian mobility: observations on the USC campus,” in *IEEE 60th Vehicular Technology Conference, 2004. VTC2004-Fall. 2004*, vol. 4, pp. 2887–2891, doi: 10.1109/VETECF.2004.1400588.
- [85] Tong Liu, P. Bahl, and I. Chlamtac, “Mobility modeling, location tracking, and trajectory prediction in wireless ATM networks,” *IEEE J. Sel. Areas Commun.*, vol. 16, no. 6, pp. 922–936, 1998, doi: 10.1109/49.709453.

- [86] C. Tang, M. Hao, X. Wei, and W. Chen, “Energy-aware task scheduling in mobile cloud computing,” *Distrib. Parallel Databases*, vol. 36, no. 3, pp. 529–553, Sep. 2018, doi: 10.1007/s10619-018-7231-7.
- [87] Hsu Mon Kyi and Thinn Thu Naing, “Stochastic Markov Model Approach for Efficient Virtual Machines Scheduling on Private Cloud,” *Int. J. Cloud Comput. Serv. Archit.*, vol. 1, no. 3, pp. 1–13, Nov. 2011, doi: 10.5121/ijccsa.2011.1301.
- [88] K. Jagannathan and E. Modiano, “The Impact of Queue Length Information on Buffer Overflow in Parallel Queues,” *IEEE Trans. Inf. Theory*, vol. 59, no. 10, pp. 6393–6404, Oct. 2013, doi: 10.1109/TIT.2013.2268926.
- [89] D. Wang, J. Chen, and W. Zhao, “A Task Scheduling Algorithm for Hadoop Platform,” *J. Comput.*, vol. 8, no. 4, Apr. 2013, doi: 10.4304/jcp.8.4.929-936.
- [90] H. Khojasteh, J. Mistic, and V. Mistic, “Prioritization of Overflow Tasks to Improve Performance of Mobile Cloud,” *IEEE Trans. Cloud Comput.*, vol. 7161, no. c, pp. 1–1, 2016, doi: 10.1109/TCC.2016.2535240.
- [91] X. Nan, Y. He, and L. Guan, “Queueing model based resource optimization for multimedia cloud,” *J. Vis. Commun. Image Represent.*, vol. 25, no. 5, pp. 928–942, 2014, doi: 10.1016/j.jvcir.2014.02.008.
- [92] Yuan Zhang, Jinyao Yan, and Xiaoming Fu, “Reservation-based resource scheduling and code partition in mobile cloud computing,” in *2016 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPs)*, Apr. 2016, pp. 962–967, doi: 10.1109/INFOCOMW.2016.7562219.
- [93] X. Wei, J. Fan, Z. Lu, and K. Ding, “Application scheduling in mobile cloud computing with load balancing,” *J. Appl. Math.*, vol. 2013, 2013, doi: 10.1155/2013/409539.
- [94] M. Nir, A. Matrawy, and M. St-Hilaire, “An energy optimizing scheduler for mobile cloud computing environments,” *Proc. - IEEE INFOCOM*, pp. 404–409, 2014, doi: 10.1109/INFOCOMW.2014.6849266.
- [95] X. Lin, Y. Wang, Q. Xie, and M. Pedram, “Task Scheduling with Dynamic Voltage and Frequency Scaling for Energy Minimization in the Mobile Cloud Computing

- Environment,” *IEEE Trans. Serv. Comput.*, vol. 8, no. 2, pp. 175–186, 2015, doi: 10.1109/TSC.2014.2381227.
- [96] W. Zhang, Y. Wen, and D. O. Wu, “Collaborative task execution in mobile cloud computing under a stochastic wireless channel,” *IEEE Trans. Wirel. Commun.*, vol. 14, no. 1, pp. 81–93, 2015, doi: 10.1109/TWC.2014.2331051.
- [97] S. Guo, B. Xiao, Y. Yang, and Y. Yang, “Energy-efficient dynamic offloading and resource scheduling in mobile cloud computing,” *Proc. - IEEE INFOCOM*, vol. 2016-July, 2016, doi: 10.1109/INFOCOM.2016.7524497.
- [98] Lin Wang, L. Jiao, D. Kliazovich, and P. Bouvry, “Reconciling task assignment and scheduling in mobile edge clouds,” in *2016 IEEE 24th International Conference on Network Protocols (ICNP)*, Nov. 2016, pp. 1–6, doi: 10.1109/ICNP.2016.7785317.
- [99] T. Wang, X. Wei, C. Tang, and J. Fan, “Efficient multi-tasks scheduling algorithm in mobile cloud computing with time constraints,” *Peer-to-Peer Netw. Appl.*, vol. 11, no. 4, pp. 793–807, 2018, doi: 10.1007/s12083-017-0561-9.
- [100] H. Shah-Mansouri, V. W. S. Wong, and R. Schober, “Joint Optimal Pricing and Task Scheduling in Mobile Cloud Computing Systems,” *IEEE Trans. Wirel. Commun.*, vol. 16, no. 8, pp. 5218–5232, 2017, doi: 10.1109/TWC.2017.2707084.
- [101] T. Zhao, S. Zhou, X. Guo, and Z. Niu, “Tasks scheduling and resource allocation in heterogeneous cloud for delay-bounded mobile edge computing,” in *2017 IEEE International Conference on Communications (ICC)*, May 2017, pp. 1–7, doi: 10.1109/ICC.2017.7996858.
- [102] J. Zhang *et al.*, “Hybrid computation offloading for smart home automation in mobile cloud computing,” *Pers. Ubiquitous Comput.*, vol. 22, no. 1, pp. 121–134, 2018, doi: 10.1007/s00779-017-1095-0.
- [103] L. Lin, P. Li, J. Xiong, and M. Lin, “Distributed and Application-Aware Task Scheduling in Edge-Clouds,” in *2018 14th International Conference on Mobile Ad-Hoc and Sensor Networks (MSN)*, Dec. 2018, pp. 165–170, doi: 10.1109/MSN.2018.000-1.
- [104] C. Tang *et al.*, “A Mobile Cloud Based Scheduling Strategy for Industrial Internet



- of Things,” *IEEE Access*, vol. 6, pp. 7262–7275, 2018, doi: 10.1109/ACCESS.2018.2799548.
- [105] Q. Jiang, V. C. M. Leung, H. Tang, and H. S. Xi, “Adaptive Scheduling of Stochastic Task Sequence for Energy-Efficient Mobile Cloud Computing,” *IEEE Syst. J.*, vol. 13, no. 3, pp. 3022–3025, 2019, doi: 10.1109/JSYST.2019.2922436.
- [106] T. Oo and Y.-B. Ko, “Application-aware Task Scheduling in Heterogeneous Edge Cloud,” in *2019 International Conference on Information and Communication Technology Convergence (ICTC)*, Oct. 2019, pp. 1316–1320, doi: 10.1109/ICTC46691.2019.8939927.
- [107] W. Tian, R. Gu, R. Feng, X. Liu, and S. Fu, “A QoS-Aware workflow scheduling method for cloudlet-based mobile cloud computing,” *Proc. - 2019 IEEE Int. Congr. Cybermatics 12th IEEE Int. Conf. Internet Things, 15th IEEE Int. Conf. Green Comput. Commun. 12th IEEE Int. Conf. Cyber, Phys. So*, pp. 164–169, 2019, doi: 10.1109/iThings/GreenCom/CPSCoM/SmartData.2019.00048.
- [108] L. Chen, K. Guo, G. Fan, C. Wang, and S. Song, “Resource Constrained Profit Optimization Method for Task Scheduling in Edge Cloud,” *IEEE Access*, vol. 8, pp. 118638–118652, 2020, doi: 10.1109/ACCESS.2020.3000985.
- [109] C. Arun and K. Prabu, “A multi-objective EBCO-TS algorithm for efficient task scheduling in mobile cloud computing,” *Int. J. Netw. Virtual Organ.*, vol. 22, no. 4, pp. 366–386, 2020, doi: 10.1504/IJNVO.2020.107570.
- [110] M. Garg and R. Nath, “Autoregressive dragonfly optimization for multiobjective task scheduling (ado-mts) in mobile cloud computing,” *J. Eng. Res.*, vol. 8, no. 3, pp. 71–90, 2020, doi: 10.36909/JER.V8I3.7643.
- [111] B. Hendrickson and T. G. Kolda, “Graph partitioning models for parallel computing,” *Parallel Comput.*, vol. 26, no. 12, pp. 1519–1534, 2000, doi: 10.1016/S0167-8191(00)00048-X.
- [112] M. Kaya, A. Koçyıldız, and P. E. Eren, “An adaptive mobile cloud computing framework using a call graph based model,” *J. Netw. Comput. Appl.*, vol. 65, pp. 12–35, 2016, doi: 10.1016/j.jnca.2016.02.013.

- [113] G. Karypis and V. Kumar, “Parallel multilevel graph partitioning,” in *Proceedings of International Conference on Parallel Processing*, pp. 314–319, doi: 10.1109/IPPS.1996.508075.
- [114] B. Hendrickson and R. Leland, “An Improved Spectral Graph Partitioning Algorithm for Mapping Parallel Computations,” *SIAM J. Sci. Comput.*, vol. 16, no. 2, pp. 452–469, Mar. 1995, doi: 10.1137/0916028.
- [115] T. Chaco, “Bruce Hendrickson and Robert Leland,” *Sandia Natl. Lab.*, no. July, pp. 1–44, 1995.
- [116] “Traceview. Profiling with traceview and dmtracedump,” <http://developer.android.com/tools/debugging/debugging-tracing.html>.  
<http://developer.android.com/tools/debugging/debugging-tracing.html>.
- [117] A. Carroll and G. Heiser, “An analysis of power consumption in a smartphone,” *Proc. 2010 USENIX Annu. Tech. Conf. USENIX ATC 2010*, pp. 271–284, 2019.
- [118] K.-T. (Tim) Cheng and Y.-C. Wang, “Using Mobile GPU for General-Purpose Computing - A Case Study of Face Recognition on Smartphones,” *2011 Int. Symp. Vlsi Des. Autom. Test*, pp. 54–57, 2011.
- [119] Seungjun Yang *et al.*, “Fast dynamic execution offloading for efficient mobile cloud computing,” in *2013 IEEE International Conference on Pervasive Computing and Communications (PerCom)*, Mar. 2013, pp. 20–28, doi: 10.1109/PerCom.2013.6526710.
- [120] K. Akherfi, M. Gerndt, and H. Harroud, “Mobile cloud computing for computation offloading: Issues and challenges,” *Appl. Comput. Informatics*, vol. 14, no. 1, pp. 1–16, Jan. 2018, doi: 10.1016/j.aci.2016.11.002.
- [121] P. A. L. Rego, P. B. Costa, E. F. Coutinho, L. S. Rocha, F. A. M. Trinta, and J. N. de Souza, “Performing computation offloading on multiple platforms,” *Comput. Commun.*, vol. 105, pp. 1–13, Jun. 2017, doi: 10.1016/j.comcom.2016.07.017.
- [122] A. Ferrari, S. Giordano, and D. Puccinelli, “Reducing your local footprint with anyrun computing,” *Comput. Commun.*, vol. 81, pp. 1–11, May 2016, doi: 10.1016/j.comcom.2016.01.006.

- [123] H. R. Flores Macario and S. Srirama, “Adaptive code offloading for mobile cloud applications,” in *Proceeding of the fourth ACM workshop on Mobile cloud computing and services - MCS '13*, 2013, p. 9, doi: 10.1145/2497306.2482984.
- [124] A. A. Majeed, A. U. R. Khan, R. UlAmin, J. Muhammad, and S. Ayub, “Code offloading using support vector machine,” in *2016 Sixth International Conference on Innovative Computing Technology (INTECH)*, Aug. 2016, pp. 98–103, doi: 10.1109/INTECH.2016.7845057.
- [125] and L. Y. M. Gordon, L. Zhang, B. Tiwana, R. Dick, Z. M. Mao, “PowerTutor: A power monitor for android-based mobile platforms,” 2013. <http://ziyang.eecs.umich.edu/projects/powertutor/>, 2.
- [126] L. Zhang *et al.*, “Accurate online power estimation and automatic battery behavior based power model generation for smartphones,” in *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis - CODES/ISSS '10*, 2010, p. 105, doi: 10.1145/1878961.1878982.
- [127] S. Bermejo and J. Cabestany, “Adaptive soft k-nearest-neighbour classifiers,” *Pattern Recognit.*, vol. 33, no. 12, pp. 1999–2005, Dec. 2000, doi: 10.1016/S0031-3203(99)00186-7.
- [128] W. B. Claster, “Naïve Bayes Classifier,” in *Mathematics and Programming for Machine Learning with R*, CRC Press, 2020, pp. 141–160.
- [129] S. Menard, *Applied Logistic Regression Analysis*. 2455 Teller Road, Thousand Oaks California 91320 United States of America: SAGE Publications, Inc., 2002.
- [130] O. Sagi and L. Rokach, “Ensemble learning: A survey,” *Wiley Interdiscip. Rev. Data Min. Knowl. Discov.*, vol. 8, no. 4, Jul. 2018, doi: 10.1002/widm.1249.
- [131] M. Tanriverdi and A. A. M, “Context-Aware Decision Making System for Mobile Cloud Offloading,” *Int. J. Comput. Networks Commun.*, vol. 7, no. 6, pp. 69–85, Nov. 2015, doi: 10.5121/ijcnc.2015.7605.
- [132] N. Golmie, “Seamless mobility: are we there yet?,” *IEEE Wirel. Commun.*, vol. 16, no. 4, pp. 12–13, Aug. 2009, doi: 10.1109/MWC.2009.5281249.
- [133] I. Al-Surmi, M. Othman, and B. Mohd Ali, “Mobility management for IP-based next

- generation mobile networks: Review, challenge and perspective,” *J. Netw. Comput. Appl.*, vol. 35, no. 1, pp. 295–315, Jan. 2012, doi: 10.1016/j.jnca.2011.09.001.
- [134] D. Le, X. Fu, and D. Hogrefe, “A review of mobility support paradigms for the internet,” *IEEE Commun. Surv. Tutorials*, vol. 8, no. 1, pp. 38–51, 2006, doi: 10.1109/COMST.2006.323441.
- [135] Y. Singh, “Comparison of Okumura, Hata and COST-231 Models on the Basis of Path Loss and Signal Strength,” *Int. J. Comput. Appl.*, vol. 59, no. 11, pp. 37–41, Dec. 2012, doi: 10.5120/9594-4216.
- [136] F. Bai and A. Helmy, “A Survey of Mobility Models in Wireless Adhoc Networks,” *Wirel. Ad Hoc Sens. Networks*, pp. 1–30, 2004.
- [137] M. Deshpande and G. Karypis, “Selective Markov models for predicting Web page accesses,” *ACM Trans. Internet Technol.*, vol. 4, no. 2, pp. 163–184, May 2004, doi: 10.1145/990301.990304.
- [138] K. Ayyappan, K. Narasimman, and P. Dananjayan, “SINR Based Vertical Handoff Scheme for QoS in Heterogeneous Wireless Networks,” in *2009 International Conference on Future Computer and Communication*, Apr. 2009, pp. 117–121, doi: 10.1109/ICFCC.2009.121.
- [139] B. O. H. Akinwale and J. J. Biebuma, “Comparative Analysis of Empirical Path Loss Model for Cellular Transmission in Rivers State,” *Am. J. Eng. Res.*, vol. 02, no. 08, pp. 24–31, 2013.
- [140] H. Nashaat, N. Ashry, and R. Rizk, “Smart elastic scheduling algorithm for virtual machine migration in cloud computing,” *J. Supercomput.*, vol. 75, no. 7, pp. 3842–3865, Jul. 2019, doi: 10.1007/s11227-019-02748-2.
- [141] K.M. Passino, “Biomimicry of bacterial foraging for distributed optimization and control,” *IEEE Control Syst.*, vol. 22, no. 3, pp. 52–67, Jun. 2002, doi: 10.1109/MCS.2002.1004010.
- [142] C. A. Coello Coello and M. S. Lechuga, “MOPSO: a proposal for multiple objective particle swarm optimization,” in *Proceedings of the 2002 Congress on Evolutionary Computation. CEC’02 (Cat. No.02TH8600)*, vol. 2, pp. 1051–1056, doi:

10.1109/CEC.2002.1004388.

- [143] R. A. Krohling, “Gaussian swarm: a novel particle swarm optimization algorithm,” in *IEEE Conference on Cybernetics and Intelligent Systems, 2004.*, vol. 1, pp. 372–376, doi: 10.1109/ICCIS.2004.1460443.
- [144] E. S. Alkayal, N. R. Jennings, and M. F. Abulkhair, “Efficient Task Scheduling Multi-Objective Particle Swarm Optimization in Cloud Computing,” in *2016 IEEE 41st Conference on Local Computer Networks Workshops (LCN Workshops)*, Nov. 2016, pp. 17–24, doi: 10.1109/LCN.2016.024.
- [145] Rajni and I. Chana, “Bacterial foraging based hyper-heuristic for resource scheduling in grid computing,” *Futur. Gener. Comput. Syst.*, vol. 29, no. 3, pp. 751–762, Mar. 2013, doi: 10.1016/j.future.2012.09.005.

---

## LIST OF PUBLICATION

- Robin P. Mathur and Manmohan Sharma, "Graph-Based Application Partitioning Approach for Computational Offloading in Mobile Cloud Computing", Recent Advances in Computer Science and Communications (2021) 14: 92. <https://doi.org/10.2174/2213275912666190716114033> (SCOPUS)
- Robin P. Mathur and Manmohan Sharma, "A survey on computational offloading in mobile cloud computing," 2019 Fifth International Conference on Image Information Processing Solan (ICIIP), 2019, pp. 515-520, doi: 10.1109/ICIIP47207.2019.8985893. IEEE- (SCOPUS)
- Mathur R.P., Sharma M. (2021) Mobility Management Scheme During Offloading in Mobile Cloud Computing. In: Choudhury S., Gowri R., Sena Paul B., Do DT. (eds) Intelligent Communication, Control and Devices. Advances in Intelligent Systems and Computing, vol 1341. Springer, Singapore. [https://doi.org/10.1007/978-981-16-1510-8\\_13](https://doi.org/10.1007/978-981-16-1510-8_13)
- R. P. Mathur and M. Sharma, "A Machine Learning Approach for offload Decision Making in Mobile Cloud Computing," 2021 7th International Conference on Advanced Computing and Communication Systems (ICACCS), 2021, pp. 1148-1154, doi: 10.1109/ICACCS51430.2021.9441795. IEEE- (SCOPUS)