1-1-2019

An SOA-Based Framework of Computational Offloading for Mobile Cloud Computing

Rajasi D. Upadhyay
University of Windsor

An SOA-Based Framework of Computational Offloading for Mobile Cloud Computing

By

Rajasi Upadhyay

A Thesis

Submitted to the Faculty of Graduate Studies

Through Computer Science

In Partial Fulfilment of the Requirements for

The Degree of Master of Science at the

University of Windsor

Windsor, Ontario, Canada

2019

An SOA-Based Framework of Computational Offloading for Mobile Cloud Computing

by

Rajasi Upadhyay

APPROVED BY:

_____

M. Monfared

Department of Mathematics and Statistics

_____

J. Lu

School of Computer Science

_____

X. Yuan, Advisor

School of Computer Science

December 16th 2019

DECLARATION OF ORIGINALITY

I hereby confirm that I am the sole creator of this theory, and no piece of this proposition has been distributed or submitted for production.

I approve to the best of my insight, my proposition does not encroach upon anybody's copyright nor damage any restrictive rights and their thoughts, methods, citations, or some other material crafted by other individuals incorporated into my postulation, circulation or something else, and are entirely recognized as per the standard referencing hone. Moreover, I have included copyrighted material that outperforms the limits of reasonable managing inside the significance of the Canada Copyright Act. I affirm to acquire a composed consent from the copyright owner(s) to incorporate such material(s) in my postulation and have included duplicates of such copyright clearances to my reference section.

I pronounce that this is a genuine copy of my thesis, including any last updates, as affirmed by my thesis advisory group and the Graduate Studies office, and this theory has not been submitted for a higher degree to some other University or Institution.

ABSTRACT

Mobile Computing is a technology that allows transmission of audio, video, and other types of data via a computer or any other wireless-enabled device without having to be connected to a fixed physical link. Despite increasing usage of mobile computing, exploiting its full potential is difficult due to its inherent problems such as resource scarcity, connection instability, and limited computational power. In particular, the advent of connecting mobile devices to the internet offers the possibility of offloading computation and data intensive tasks from mobile devices to remote cloud servers for efficient execution. This proposed thesis develops an algorithm that uses an objective function to adaptively decide strategies for computational offloading according to changing context information. By following the style of Service-Oriented Architecture (SOA), the proposed framework brings cloud computing to mobile devices for mobile applications to benefit from remote execution of tasks in the cloud. This research discusses the algorithm and framework, along with the results of the experiments with a newly developed system for self-driving vehicles and points out the anticipated advantages of Adaptive Computational Offloading.

DEDICATION

Dedicated to God, my grandparents, my beloved mummy and papa without whose support, I would not have made it this far, my loving sister, my supportive brother-in-law, my adorable nephew, my dear cousins and the rest of my family and friends.

ACKNOWLEDGEMENTS

First and foremost, I would like to express profound thankfulness to my supervisor, Dr. Xiaobu Yuan, who has supported me throughout my thesis with his knowledge and expertise in this exciting field of research. His ideas and suggestions have helped me become more creative, without which I would not have been able to complete this research.

I want to offer my sincere gratitude to the advisory group members, Dr. Jianguo Lu and Dr. Mehdi Monfared, for their significant remarks and recommendations for my research.

I want to thank all my friends, especially Saurav Agrawal, Rachit Tomar, and Chandini Nair, who have supported and helped me throughout my studies here in Canada. I also thank my parents and family for their blessings and financial support, which enabled me to complete my studies successfully.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF ABBREVIATIONS/SYMBOLS

QoS	Quality of Service
MCC	Mobile Cloud Computing
SOA	Service-Oriented Architecture
SLA	Service Level Agreement
ILP	Integer Linear Programming
LP	Linear Programming
COSMOS	Computation Offloading as a Service for Mobile devices
CPU	Central Processing Unit
UI	User Interface
OBU	On-board Unit
MACS	Mobile Augmentation Cloud Services
EEQoSA	Energy-Efficient and Quality-of-Service Architecture
HSS	Home Subscriber Server
IPMS	Internet Protocol Multimedia Subsystem
SRI	Subscription-Related Information
LU	Location Update
PRR	Periodic Re-Registration
RRCC	Re-Registration for Change Capabilities
MRA	Media Resource Agent
MRFC	Media Resource Function Controller
WP	Weight value of Performance
WM	Weight value of Memory

WE	Weight value of Economic Cost
3D	3-Dimensional
2D	2-Dimensional
VGI	Volunteered Graphic Information
API	Application Programming Interface
HTTP	HyperText Transfer Protocol
XML	Extensible Markup Language
RCNN	Region Convolutional Neural Network
SSID	Service Set Identifier
AWS	Amazon Web Services
URL	Uniform Resource Locator
CAD	Canadian Dollar
EC2	Elastic Cloud Compute
AMI	Amazon Machine Image
RDP	Remote Desktop Protocol
TCP	Transmission Control Protocol

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

The use and deployment of smartphone platforms and apps have increased phenomenally, around the globe. We use mobile devices for various tasks like communicating in different ways, sending emails, online banking, browsing internet, watching videos, using social media and navigating using online maps. Different applications are available for performing these tasks. Due to the increasing growth of mobile applications and demands of users, the Quality of Service (QoS) is obstructed by restrictions at the mobile end such as resource limitations, finite energy, limited available connectivity and shared wireless medium. Since the resources of the mobile devices are not enough for performing computation-intensive and resource-intensive tasks like image recognition and natural language processing, the concept of Mobile Cloud Computing is used. Mobile Cloud Computing (MCC) addresses these resource constraints by incorporating cloud computing into the mobile environment. This is done by using Computational Offloading, which involves enabling resource-intensive applications to be executed on remote cloud servers.

1.1 Mobile Computing

Mobile Computing is a technology that enables data, voice, and video to be transmitted using a device like a laptop without connecting to a fixed physical connection [1].

Figure 1.1: Mobile Computing

There are three basic concepts of mobile computing - portable hardware, mobile software, and mobile communication. Portable hardware includes mobile phones, and other gadgets available in the market that can support mobile computing on devices. Some examples of such portable hardware are smart phones, tablet PCs, laptops, etc. Mobile software includes various applications like web browsers, maps, and games. Mobile communication includes communication channels like Wi-Fi, Bluetooth, and cellular networks. There are numerous applications of Mobile Computing as shown in Figure 1.2, when used with wireless networks. However, there are various constraints on mobile computing like resource limitations, mobile connectivity, finite energy, and security.

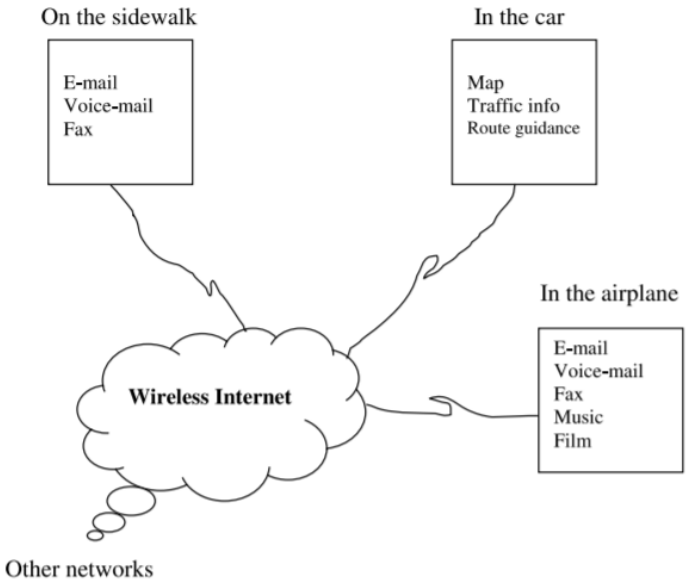

Figure 1.2 Applications of Mobile Computing [49]

1.2 Mobile Cloud Ecosystem

We explain mobile cloud ecosystem and its different components, in this section. We also discuss the Quality of Experience (QoE) of a user to study the impact of mobile cloud ecosystem. In this ecosystem, the mobile system and the cloud system are connected by network. Hence, the ecosystem consists of three distinct components [2]:

1. Mobile System
2. Network

3. Cloud System

The QoE of a user is affected by the three above-mentioned components of the mobile cloud ecosystem as shown in Figure 1.3. The QoE, in turn, has four components – memory usage, energy consumption, monetary cost, and data transmission rate. The environmental parameters in mobile cloud ecosystem are categorized into following three components based on the source of variation – applications, network, and execution platform.

1.2.1 Applications

Some mobile applications are computation and resource-intensive e.g., Object Recognition, Map Reconstruction, Image Search, etc. while other applications like Text Editor, Web Browser, etc. are not. These applications impose varying requirements on the mobile system. Handling these variations for a mobile system is a major challenge.

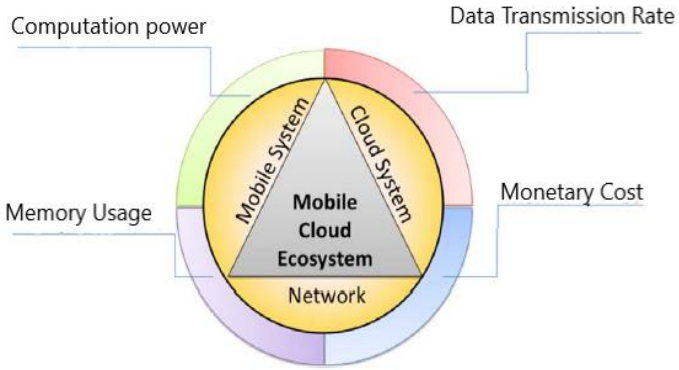

Figure 1.3: Mobile Cloud Ecosystem

1.2.2 Network

There are several types of wireless networks (cellular network, Wi-Fi, Bluetooth) available today. However, wireless networks are not reliable and stable since the location of mobile devices keeps changing from time to time. Also, the signal strength is different when mobile devices are moving. Mobile systems are required to adapt to these changes and continue to function seamlessly.

1.2.3 Execution Platform

The executing platform refers to the mobile devices and the remote cloud servers. Organization and hardware of mobile devices, and the cloud systems have a number of variations. A mobile

system should be adaptive to these changes since it is not feasible to develop a separate system for each configuration.

1.3 Mobile Cloud Computing

It is hard to exploit the complete potential of mobile computing despite its growing use. This is due to its intrinsic issues such as resource scarcity, frequent disconnections, and mobility. These issues can be addressed using Mobile Cloud Computing by executing mobile applications on external resources i.e. the cloud. By extending the storage and computing capabilities of mobile devices, cloud resources are utilized, and hence, mobile cloud computing is used to expand the mobile cloud ecosystem [3].

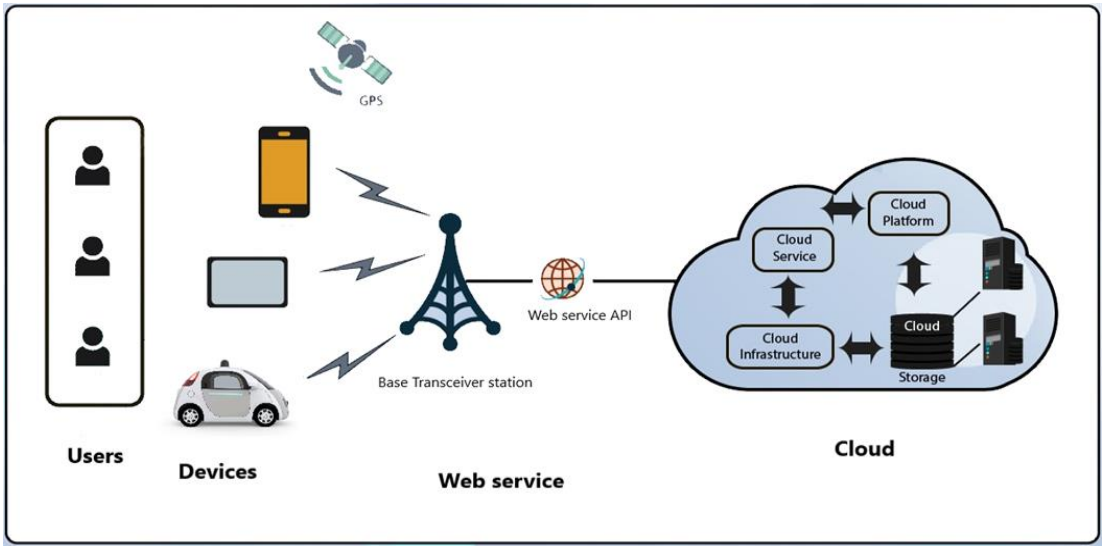

Figure 1.4: Mobile Cloud Computing

1.3.1 Mobile Cloud Computing: Service-Oriented Architecture

Service-Oriented Architecture (SOA) is an architecture used for developing distributed and interoperable applications [4]. A distributed application has application components that run in various nodes of the computer. In other words, an application (desktop or internet) that runs at one place in one system and utilizes some features or business logic from another system somewhere in the world. An interoperable application means one application developed in one language; for example, Python, can communicate with another application developed using another language such as C#. The service-oriented approach can provide a reduction in network cost. Communication overhead can also be decreased considerably as the network transactions mainly

contain only information related to the task and not all of its code. Hence, SOA can be used to provide cloud computing services to mobile applications, where task functionalities are provided as services [5].

1.3.2 Mobile Cloud Computing: Computational Offloading

As mentioned earlier, the user expects the mobile system to run a variety of applications. However, a mobile system is constrained by different parameters at any point in time, such as memory limitation, the limited computation power of existing mobile processors, etc. Computation offloading aims to enable memory or computation-intensive applications on mobile systems by distributed execution of mobile applications. This is achieved by running a portion of the application from the mobile device to remote computation resources such as a cloud server. To enable distributed execution of a mobile application, the offloading framework must determine how to partition any application for scheduling on a mobile device and the cloud servers [2].

Even though Computational Offloading originated around the 1970s, its potential was widely explored only after wireless communication and Internet speed were improved sufficiently and were able to support it [53]. The offloading capacity depends primarily on technology such as cellular and WiFi networks, because these networks determine the feasibility of computational offloading. In today's age, it has been seen in various scenarios where the Wi-Fi is able to deliver high bandwidth connections. Moreover, 5G network is receiving a lot of attention, as it can support increased network capacity and low latency and it paves the way for MCC to solve the problem of computing offloading.

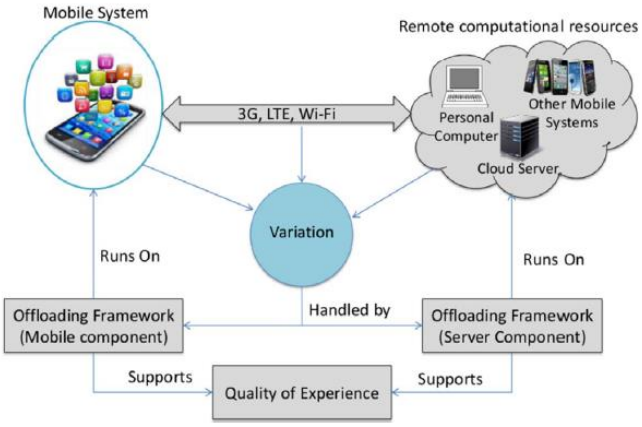

Figure 1.5: Computational Offloading

Hence, by using these wireless networks, offloading of the applications is performed to the cloud resources. Figure 1.5 represents the general architecture for computational offloading. Variations mentioned earlier in Section 1.2 such as applications, network, and execution platform should be handled while offloading. Also, the Quality of Experience of the user should be supported.

1.4 Challenges in Computational Offloading for Mobile Cloud Computing

Following challenges are taken into consideration based on the requirements of mobile devices [12]:

1. Network Connectivity and Fault Tolerance

As the mobile devices are constantly moving, freedom of movement and communication autonomy during the use of mobile cloud services are key criteria for the satisfaction of users. However, while on the move, there are some constraints that prevent seamless connectivity and uninterrupted access to cloud services. Data exchange rates and network bandwidth differ as mobile users travel. In addition, when sending or receiving data, users lose their connection in some locations; thus, appropriate fault-tolerant strategies should be provided for offloading approaches to resend the lost components and to reduce the response time.

2. Automatic Mechanism

There is still a need to automate the existing computing offloading frameworks. The automation makes the offloading process to be carried out seamlessly while taking the surrounding environment into consideration. It is not an easy task to perform this type of automation as it requires a protocol to define and discover services in accordance with the current context and its constraints.

3. Diverse Platform

The diversity and heterogeneity of mobile device architectures and operating systems are some of the issues in the current computing offloading frameworks. Consistent access to cloud services is required where mobile devices can access cloud computing services irrespective of the operating system installed or the hardware used. A standardized offloading system for various mobile platforms is still a challenging problem in the field of MCC.

4. Offloading Cost

The usage of cloud resources imposes financial costs on end-users who are required to pay in compliance with the Service Level Agreement (SLA) decided with the cloud provider supporting them. Content offloading and data transfer operations between cloud providers typically incur additional costs for end-users. Economic factors should, therefore, be taken into account when making offloading decisions.

1.5 Problem statement

Since the context of a mobile device is varying from time to time, the computational offloading strategy should adjust itself dynamically in order to achieve the best energy-efficiency, get the best performance consumption and reduce the monetary cost according to the change of these context information. Besides, because of the inherent intricacy of the context on which mobile devices are executing, the computational offloading strategy should adapt itself automatically, rather than adjust itself manually. As to be discussed in the literature survey (Chapter 2), various methods are used in offloading. However, some of them are manual, some are not flexible, and some are just surveys which do not provide specific formulas that can be used for offloading. For solving these problems caused by changes in the context of mobile devices, an SOA based offloading framework is proposed using Mobile Cloud Computing (MCC). This framework adapts itself and make a proper decision of the offloading strategy according to the changes in the context information. There is a service selection mechanism which selects the required service from several services.

1.6 Motivation

The introduction of the connectivity of the mobile device to the Internet offers the possibility of offloading computation-intensive and resource-intensive tasks from the mobile device to remote cloud servers for efficient execution. However, as mentioned in Section 1.4, there are several challenges in Computational Offloading for Mobile Cloud Computing, such as Network Connectivity and Fault Tolerance, Automatic Mechanism, Diverse Platform, and Offloading Cost. Due to these limitations, none of the existing approaches are able to solve all the issues, hence, not allowing the users to the exploit full potential of the devices. Hence, this thesis covers the problems mentioned above by creating an SOA-based system of Computational Offloading for Mobile Cloud Computing. This novel system for bringing Mobile Cloud Computing to mobile devices

creates benefits by remote execution of various application tasks that are provided as services. This system is advantageous to the end-users in terms of cost reduction, faster, and better performance.

1.7 Thesis contribution

Major contributions of this research work can be summarized as follows:

1. Created an Objective Function for deciding the executing location of a service:
   OF *= P(s1,s2,…,sn)*WP + M(s1,s2,…,sn)*WM + E(s1,s2,…,sn)*WE****,***
   where s represents the services, P represents Performance(computation), M represents Memory, E represents Economic Cost, WP represents the weight value of Performance, WM represents the weight value of Memory and WE represents the weight value of Economic Cost.

2. Created an offloading framework. Depending on the adaptive objective function and the changing context information, this offloading framework generates a most appropriate offloading strategy at a certain time.

3. Converted different self-driving modules to an SOA based system for better maintenance and easier expansion.

4. Performed number of experiments on the SOA based system to verify that the adaptive offloading framework, the adaptive objective function, and an adaptive service selection mechanism achieve the best power consumption efficiency, get the satisfying performance and reduce the monetary cost according to the changing context on which mobile device such as mobiles, cars, etc. are executing.

1.8 Structure of the thesis

Chapter 2 extensively discusses the background study and related works about Computational Offloading in Mobile Cloud Computing. Chapter 3 explains the proposed system thoroughly with the overall working of the Offloading System. In Chapter 4, we explore the details about the implementation, different scenarios that were considered during the implementation of the proposed idea, and the experimental results. Finally, in Chapter 5 we conclude the thesis and provide our future work.

CHAPTER 2

LITERATURE REVIEW

This chapter explains the different methods which can be used to implement computational offloading. The chapter further discusses the various works done so far using the computational offloading technique.

2.1 Techniques for Computational Offloading

2.1.1 Code Migration

Code Migration is a computational offloading method where the source code is offloaded to remote machines at run time. There are several works which have adopted this method. MAUI [6] is one among them. In MAUI, the specific annotated parts of the code are offloaded to a middlebox. For making the decision of identifying the partitions that are needed to be offloaded, MAUI uses the concept of Integer Linear Programming (ILP). Linear Programming is a mathematical model used for solving decision problems having many decision variables that are limited by a set of constraints. ILP is a subbranch of Linear Programming (LP) where the decision variables are constrained to hold integer values. Another work which uses Code Migration is ThinkAir [7], in which, the author has allotted virtual machines for executing the application code partitions at run time. However, it causes high management overhead if the virtual machines are commissioned at runtime. In COSMOS: computation offloading as a service for mobile devices [8], task allocation is managed and offloaded to cloud instance on virtual machines. This method, however, is customized for Android x86 processor only.

2.1.2 Offloading by Replication

Offloading by Replication is the method in which the characteristics of the mobile device are cloned to the remote machines. CloneCloud [9] proposed this method for optimizing the computation of the mobile device. It mainly focuses on improving the performance and battery life. For partitioning of applications, they used thread-level granularity in this framework. Static program analysis for discovery of migration constraints and program profiling for building cost

models are combined to find the offloadable components. In this work, the offloading decision is taken at run time, and cloning is performed by migrating the threads from mobile devices to the cloud. However, it is an inefficient process to clone the already existing components. Another work that replicates the code execution on remote server or cloud and mobile device is Tango [10]. It replicates and then selects the one which has the minimum response time. Although replicating the same is an impractical process.

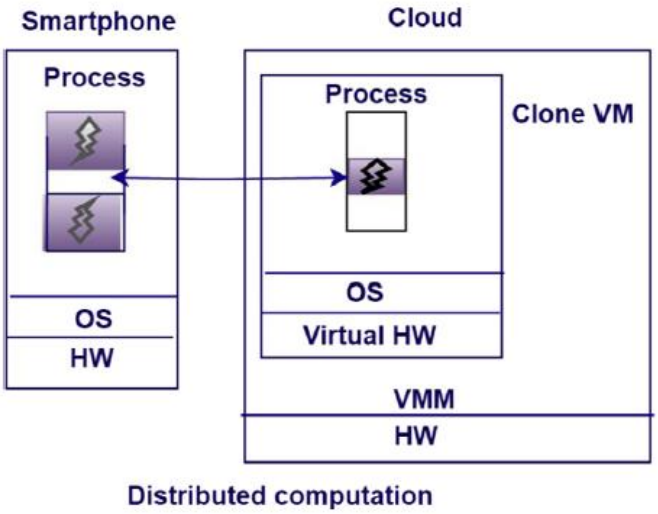

Figure 2.1: CloneCloud Execution model

2.1.3 Placement and Scheduling

Another technique for offloading is Placement and Scheduling. In this technique, Mahmoodi et al. [11] has designed an analytical framework towards scheduling for offloading, but the practicality is in question based on the assumptions about the network conditions. Considering those unfeasible assumptions, we can say that it is better to address the problem from a system point of view and build mechanisms that are practically viable.

2.1.4 Cuckoo Design

The cuckoo design was proposed for offloading from smartphones that run on the android platform to the cloud [12]. In this method, the author has used Java stub model for offloading, which integrates the Eclipse development tool with the open-source Android framework. This model supports both local and on-cloud execution of the code and works on an already existing activity/service model in android. By using that, it distinguishes between intensive and non-

intensive components of the application. In this design, static code partitioning is done in which a part of the code that is needed to be offloaded to the remote server is pre-set by the mobile devices. However, it can be said that this method is a failure in today's heterogeneous network environment. Therefore, to handle this heterogeneity, dynamic code partitioning was introduced.

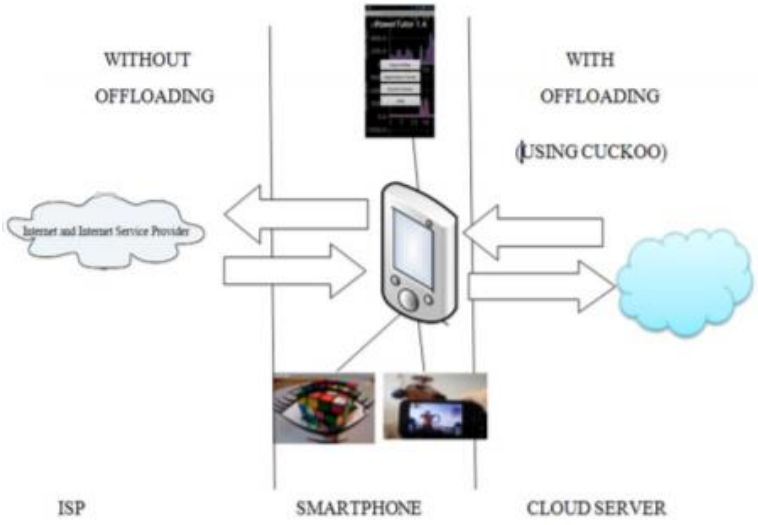

Figure 2.2: Offloading using Cuckoo Framework

2.2 Background Study

Many previous works have focused on Computational Offloading for Mobile Cloud Computing. Some of the papers use a function/formula to decide the location where the tasks need to be executed. For example, Kovachev et al. [20] provide a middleware MACS (Mobile Augmentation Cloud Services) that enables adaptive extension of Android application execution from a mobile device into cloud. Middleware is responsible for heavy lifting of application partitioning, resource monitoring, and computation offloading. The mobile applications are elastic, which can run as standard mobile applications and can also reach transparently remote computing resources. In a MACS application, there is an application core, which cannot be offloaded, and there are multiple services (S_i) that encapsulate separate application functionality which can be offloaded (S_{Ri}). The following metadata is profiled for each service:

Memory cost – memory consumption of the service,

Type – can be offloaded or not,

Transfer size – amount of data to be transferred,

Send size – amount of data to be sent,

Receive size – amount of data to be received,

Code size – size of compiled code.

For offloading, this approach considers *k* offloadable modules. For those modules, transfer size is $tr_1, tr_2, \ldots tr_k$. Send size is $send_1, send_2, \ldots send_k$ and receive size is $rec_1, rec_2, \ldots rec_k$. The cost function is depicted as follows:

$$\min_{0,1} x \left(c_{transfer} * w_{tr} + c_{memory} * w_{mem} + c_{cpu} * w_{cpu}\right)$$

The cost function is used to make the decision of executing the module locally or remotely. This framework, however, does not consider minimizing the execution cost and the network uncertainty.

Considering the limitations of the MACS, Deng et al. [17] proposes the offloading framework in which services in workflows are invoked, and a decision is made on whether to offload the services or not. In this paper, an offloading system is proposed to address the issue of unstable network connectivity of mobile devices. This approach focusses on optimizing the energy consumption and execution time of mobile services. The offloading system is based on genetic algorithm. In the class of evolutionary algorithms, genetic algorithms are based on the concept of "survival of the fittest" [21]. By recombining a population's finest alternatives and mutating them now and then, one can solve extremely challenging issues for which writing programs would otherwise be hopelessly difficult. It uses a mobility enabled and fault-tolerance offloading system for making computation offloading decisions. The main goal of the offloading planner is to propose an optimal offloading strategy that will minimize energy consumption and execution time for the executing mobile applications (workflows). Function F(*m*) is used in the offloading strategy to make decisions. F(*m*) for each mobile device *m* is defined as:

$$\mathrm{F}(m) = w_m \times L_m + (1 - w_m) \times E_m$$

Here, w_m is the weight coefficient, L_m is the overall execution time and E_m is overall energy consumed when a workflow is executing.

The author has covered the drawbacks of MACS and has also proposed an algorithm for optimized fault tolerance offloading. However, the algorithm presumes the user's moving path, but there are other possibilities.

As there are several assumptions made in the previous work, Ashok et al. [5] has proposed an architecture where the vehicular applications are remotely executed on the cloud, and the tasks are provided as services. Authors of this paper has also considered the network instability, which was a shortcoming of MACS framework. In this paper, firstly, the challenges are identified for cloud computing in vehicles. Challenges are a heterogeneous environment of the wireless network; different software and hardware architecture of the vehicle's on-board unit (OBU) and cloud; seamless offloading of tasks. In this approach, the service-oriented approach is used to provide cloud computing services to the applications. Computation and data-intensive tasks are identified, and then an offloading framework is used in which the tasks marked as offloadable are executed as services and provided in the cloud. The online placement framework is used to make decisions of offloading the tasks to cloud, based on a set of variables and a set of costs. Variables are network speed, availability of the server, offloadable modules, and other optimization parameters. Costs include execution time of modules, storage space, CPU Usage, and energy expensed. For offloading, the framework follows two steps: Profiling and Conditioning. In profiling step, the information such as input (output) data sizes to (from) each module, depicted as d_{in} (d_{out}), network uplink (W_{up}) and downlink (W_{down}) speeds, the execution time of the module on the vehicle (e) and the cloud (e*) and the storage size of the module application code, depicted as s is obtained. In Conditioning step, those parameters are used to obtain Module Execution Time ratio, E, which is as follows:

$$E = \frac{e* + \frac{din}{Wup} + \frac{dout}{Wout}}{e}$$

Based on the value of E, the decision for marking a module as offloadable is made. However, the modules to be offloaded are statically marked by the developer. Hence, the offloading is not dependent upon the actual context information.

This paper by Chen et al. [13] proposed a framework for context-aware Computation Offloading in Mobile Cloud Computing, which tries to provide a solution for the previous paper. As the previous paper by Ashok et al. [5] did not perform offloading based on the context information, this paper supports applications with the context-aware computation offloading capability. It is based on the working of mobile applications. In this approach, to enable an application to be offloaded, a design pattern is proposed. Then, for selecting the cloud resource, an estimation model is shown. After that, a framework is presented, which is implemented at the client-side and server-side to support the previously mentioned design pattern and estimation model. In this framework, during the development of an application, the methods are classified into different categories. The first category is Anchored applications, which interact with input or output devices and external services, or which implement applications UI (User Interface). The second category is Movable applications, which can run either on different execution platforms like mobile device or cloud server. The parameters considered for evaluation are battery power consumption and execution time of the applications. The disadvantage of this method is that service-oriented architecture is not used. Hence, distributed and interoperable applications will not be supported.

To take advantage of SOA's benefits, Hani et al. [18] introduced a secure energy-efficient and quality-of-service architecture (EEQoSA) for mobile cloud handover. This is a service-oriented architecture, and consists of four layers:

1. Application layer

   This layer consists of the home subscriber server (HSS) that interconnects with the cloud computing servers as an enterprise server. To maintain data communication, HSS links to Internet protocol multimedia subsystem (IPMS) layer. It includes subscription-related information (SRI) server, role manager server, and location update (LU) server. Before transferring the cloud data to a legitimate mobile cloud user, encryption is performed for secure data communication

2. IPMS layer

   This layer provides utility services, like web browsing, email, video-on-demand, videoconferencing and Internet service. It includes a registration process that helps to

obtain updated location information from the mobile cloud user. IPMS uses a call session control function to bind a public user identity to the IP address of a mobile cloud user.

3. Communication layer

   It routes the data and synchronizes the media and IPMS layers. The LU server starts the re-registration process, comprising of two levels - periodic re-registration (PRR) and re-registration for change capabilities (RRCC). These levels required the messaging process to complete the re-registration. Therefore, at the end of the handover process, the energy for registration should be calculated to determine the remaining power of the mobile device.

4. Media with connectivity layer

   It consists of a media resource agent (MRA) and a media resource function controller (MRFC). MRA controls existing media resource function information and transmits the suitable information to the authentication server. MRFC combines the streams of media and manages the shared resources. It allows only authentic users to complete the re-registration process after starting the handover and reduces the occurrence of extended delays during the handover.

This architecture ensures the security of the data handover and guarantees Quality of Service; however, it only considers energy efficiency for the performance.

Since all these previous works, which use different objective functions and algorithms for decision making has many limitations, we have designed a framework that considers all the parameters and can perform in a robust manner. Following are the survey papers that we reviewed for our research to obtain the information about various techniques used for offloading, the issues faced, and the solutions.

In the paper by Bhattacharya et al. [2], a survey of state-of-the-art adaptive algorithms which are used for Computational Offloading is presented. The entities are described using a mobile-cloud ecosystem. Those are then used to define different sources of variation in the system. Due to the mobility of the mobile device, parameters change during the run time. Hence, the mobile-cloud ecosystem is used to define the parameters and the effect of those changing parameters. Different solutions are classified for adaptive offloading based on different parameters and their adaptable

solutions. For various offloading environments, there is a Quality of Experience metrics, including energy saved, monetary cost, etc. Finally, the effect of those parameters on the user's Quality of Experience is provided.

Another paper by Zhou et al. [15] provides a survey and future directions of the augmentation techniques that can be used for Mobile Cloud Computing. Augmentation techniques are computing models and solutions for outsourcing mobile device computing and storage to more strong computing resources that can improve the computing capacity and energy efficiency of a mobile device. Hence, these techniques are used to increase, enhance, and optimize the computing capabilities of mobile devices. It offers an extensive taxonomy and survey of current mobile cloud augmentation methods and frameworks for computing and storage. Taxonomy of computation augmentation techniques is provided, as shown in Figure 2.3.

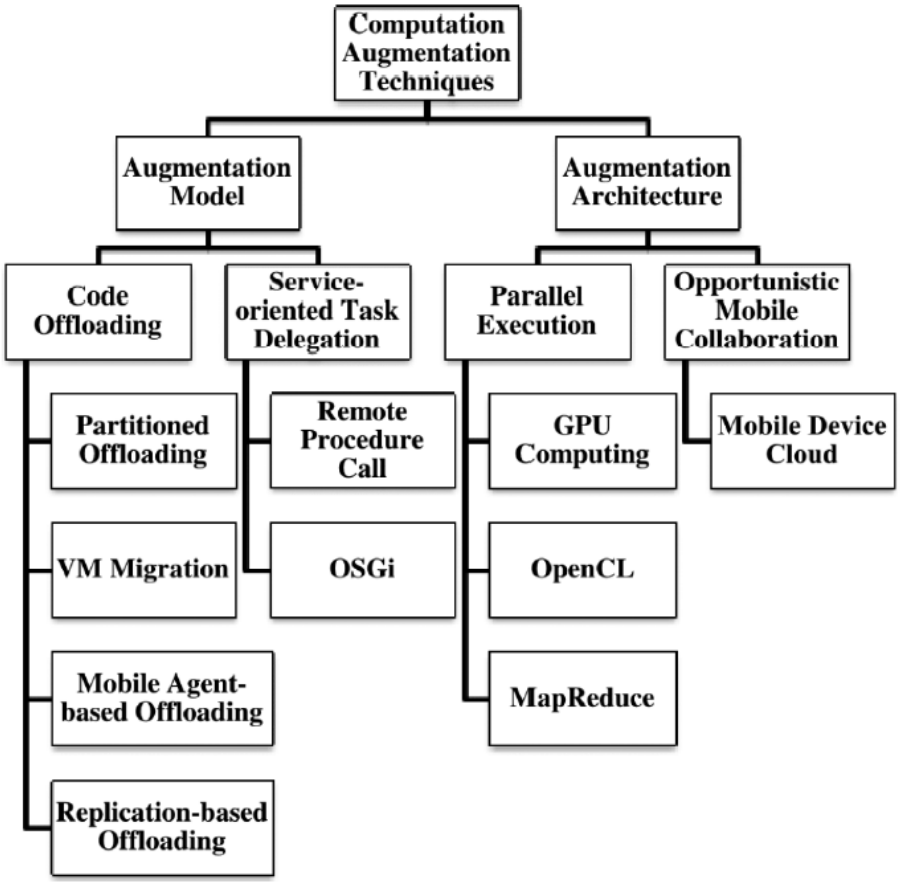

Figure 2.3: Taxonomy of computation augmentation techniques

This taxonomy addressed the methods and approaches used in mobile cloud augmentation to merge hybrid cloud resources into a shared mobile device resource pool that will provide reliable and energy-efficient computing outsourcing through a mobile cloud-as-a-service.

Next is the taxonomy of storage augmentation, which is shown in Figure 2.4 in which data-oriented architecture for storing data on clouds and the mobile device cloud is discussed. It also studies various vital issues, like data protection and data interoperability.

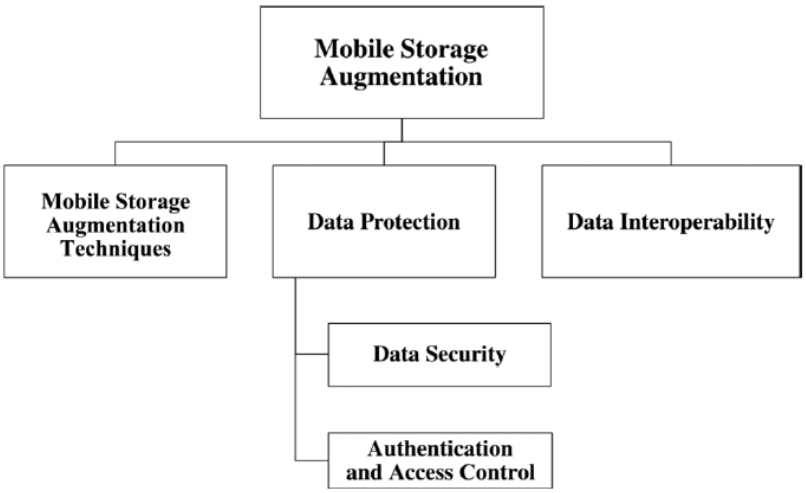

Figure 2.4: Taxonomy of Mobile Storage Augmentation

The survey also analyzed different significant technical gaps for further study.

The paper by Boukerche et al. [14] focusses on Sustainable Offloading in Mobile Cloud Computing. This survey provides a comparison of state-of-the-art works related to energy-aware offloading in the scope of MCC based offloading. They have gathered the existing studies and have classified those according to the perspective of the mobile device and the Cloud Performance. These aspects are observed in terms of the energy-aware offloading processing and the trade-off between energy reservation and execution efficiency. It includes information on algorithmic design and the implementation of various techniques. Critical analysis of the offloading techniques shows that the following three functionalities can be used for categorizing open issues:

1. Task Partitioning

Task Partitioning functionality is based on compatibility and usability. Since the hardware of mobile device and cloud are different, it is difficult to process the mobile applications on cloud. Additionally, as the performance of task offloading is not officially defined, it is estimated by the developers, and the task partitioning is done by supportive Application programming interfaces (APIs). For using the traditional applications to execute on Mobile Cloud Computing flow, a remarkable amount of work is required.

2. Profiling

   Profiling functionality gathers the profile information of the device like CPU Usage, battery value, network information, etc. which keeps changing continuously. In most cases the profile information is not accurate as the average of the specific interval is taken.

3. Decision Making

   For any offloading system, decision making is the most critical and challenging part. The operation of decision making is dependent on the profiling step mentioned earlier. For the actual execution, the decision-making process uses the profiling information and provides result based on that.

This paper then explains how these open issues are handled in various offloading techniques. Following that is the comparison between Grid Computing and Cloud Computing. Additionally, the network aspects are investigated, as it is an essential factor that affects the performance. Hence, all this information is gathered, and various solutions from existing studies are provided in terms of energy-aware offloading.

Parsa et al. [19] proposes an approach to wrap the existing programs into web service layers, such that the component can be accessed through web services. In this approach, first, the code is analyzed, and then it is wrapped into a web service. The analysis phase determines a way of using the functions of the analyzed program into a web environment. It is done in four steps: Evaluation, Conversion, Reengineering, and Web Service Generation. A new code is obtained after the completion of these steps, which contains the valuable functions of the program. In the next step, the important functions are wrapped into web services by using Service Bus Class. The Service Bus Class acts as an intermediate between the existing system and web service. Its primary duty

is to migrate obtained functions to the external environment. These two steps are performed to create a tool which can be installed in a programming environment to wrap a program into service.

2.3 Related works

The table below gives the information about the work done so far by researchers in the area closely related to this research work; also, mentioned are year and contributions.

Research Paper	**Contributions**	**Limitations**
Framework for context-aware computation offloading in mobile cloud computing by Chen X., Chen S., Zeng X., Zheng X., Zhang Y., and Rong C. (2017).	- Proposes a design pattern for Computational Offloading - Presents estimation model to calculate reduced execution time and network delay then selects cloud resource for offloading - Implements a framework to support the design pattern and estimation model	- The framework does not include resource-intensive applications. It only considers computation-intensive applications. - Only execution time and power consumption are the considered parameters. Monetary Cost, Memory Consumption, and Economic Cost are not considered. - Service-oriented Architecture is not used in this approach; hence, the benefits of SOA, as discussed in Section 1.3.1, are not applicable.
A survey of adaptation techniques in computation offloading by	- Adaptive algorithms used for computation offloading are surveyed - Parameters that influence the mobile systems and offloading system are identified	- This is a survey paper that just provides the information about different algorithms that are already developed. - It does not provide a new idea for Computational Offloading.

Bhattacharya A. and De P. (2017).	- Solutions are classified for adaptive offloading based on the parameters that the system can adapt to	
Energy-efficient service-oriented architecture for mobile cloud handover by Hani Q. B. and Dichter J. P. (2017).	- Introduces a secure energy-efficient and quality-of-service architecture (EEQoSA) for the handover process in the mobile cloud computing environment - It handles parameters like energy-efficiency, security and QoS	- Security parameter is considered in this paper, however, the experiments are not performed, so solutions for possible malicious attacks are not provided. - Parameters like Memory and Cost are not considered which shows that resource limitations of mobile devices are not taken into account.
Augmentation techniques for mobile cloud computing: A taxonomy, survey, and future directions by Zhou B., and Buyya R. (2018).	- This paper studies the augmentation techniques to increase, enhance, and optimize computing capabilities of mobile devices - It aims at execution of computation-intensive and resource-intensive mobile applications - It mainly provides a guide on what available augmentation techniques can be adopted in mobile cloud computing systems	- It only gives the information on existing taxonomies in the field and augmentation techniques that are already available for mobile cloud computing systems. - It does not provide any new Augmentation technique.

Computation offloading for service workflow in mobile cloud computing by Deng S., Huang L., Taheri J., and Zomaya A. Y. (2014).	- This paper proposes a mobility enabled and fault-tolerance offloading system for making computation offloading strategies - The proposed offloading algorithm is based on genetic algorithm - This paper is considered mainly due to its fault-tolerance technique, in the case of lost connectivity	- In this paper, user's moving path is presumed. But since there are other possibilities, it will not perform accurately in all scenarios. - The signal strength is also assumed to be persistent in some part of the path.
Enabling vehicular applications using cloud services through adaptive computation offloading by Ashok A., Steenkiste P., and Bai, F. (2015).	- This paper proposes a service-based architecture and designs a framework for computation offloading of vehicular applications. - It prototypes an end to end offloading system which uses the proposed service driven framework. - Experiments are performed on real world vehicular settings.	- In this paper, the modules to be offloaded are statically marked by the developer. Hence, the offloading is not dependent upon the context information.
Sustainable Offloading in Mobile Cloud Computing: Algorithmic Design and Implementation by Boukerche A., Guan S., and	- In this survey, a comparison of state-of-the-art works related to energy-aware offloading in the scope of MCC based offloading is provided - The existing studies are gathered and classified according to the perspective of the mobile device and the Cloud Performance aspects in terms of the energy-aware offloading processing and the tradeoff between energy	- This is a survey paper that just provides the information about existing energy-aware offloading algorithms. - It does not include other important parameters like computation power, memory and cost.

Grande R. E. D. (2019).	reservation, and execution efficiency is observed.	

Table 2.1: Review of research work based on offloading

CHAPTER 3

PROPOSED METHODOLOGY

This chapter firstly discusses about the general structure of an offloading system and further dives into the detailed explanation of the structure used in our Computational Offloading System. The General Structure of Offloading System consists of different modules like Client Proxy, Profiler and Solver. These modules are responsible for executing the method on the server side. Alternatively, our Computational Offloading System is based on Service-oriented Architecture, hence the methods are converted into services and then the offloading is performed. Our Offloading System consists of modules like Context Monitor, Decision Making Engine, Communication Manager and Offloading Planner.

In the upcoming section, the structure of general offloading system is discussed followed by the overall flowchart of the offloading system.

3.1 General System Structure

In this method, initially two requirements are taken into consideration:

1. The compiled code is on both, the mobile side and the server side

2. The client proxy, profiler and solver are installed on mobile and server side [22].

The profiler collects information about the network characteristics from the beginning and monitors that information after that. Every time a method is called, the profiler measures its energy saving potential and profiles the device and the network to get the status information. The solver then works on the results provided by the profiler and determines where the method will be executed remotely. The proxy is responsible for the server-device control and data transmission.

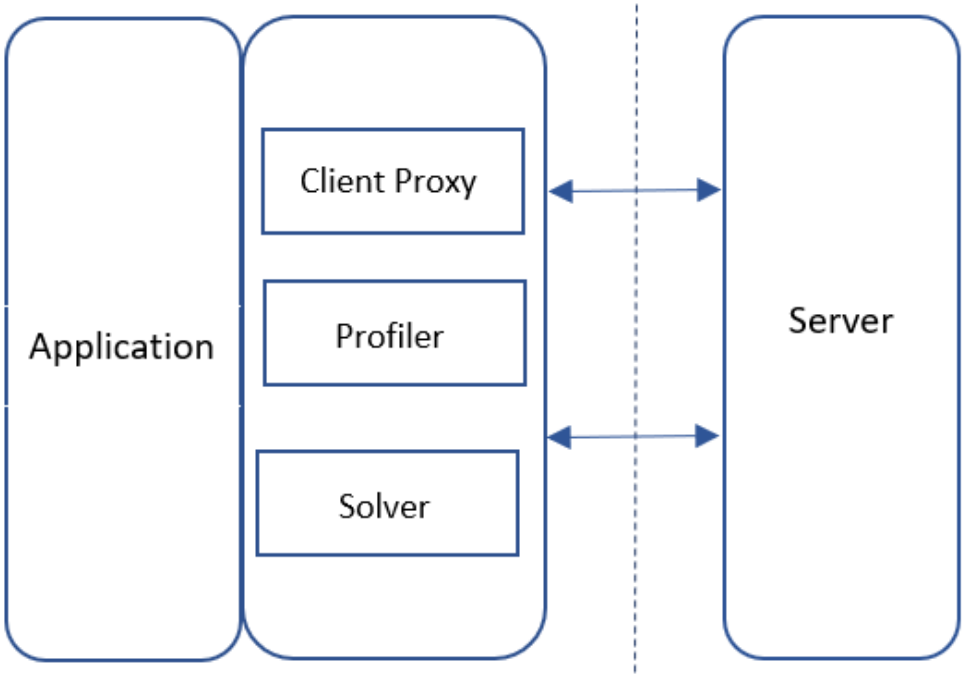

Figure 3.1: General System Structure for Offloading

Figure 3.1 shows the general system structure for Offloading in which code migration is performed.

3.2 Proposed Computational Offloading System

Since the proposed Computation Offloading System is based on Service-Oriented Architecture, we will first discuss the working and importance of SOA.

3.2.1 Service-Oriented Architecture

In the proposed system, the components are converted to services and the interaction of services is performed using Service-Oriented Architecture. SOA is a software design style in which services are provided by application components to the other components via a network communication protocol [23]. By using SOA, the components can be described more clearly, which makes it possible to structure the services. Hence, in a system, some components are service providers, who provides the required service, and some are the service consumers, who requests and receives the service. SOA architecture has a middleware, which is the intermediator between the provider and the consumer. The middleware controls the communication between the provider

and the consumer. SOA is particularly useful in the proposed system because of its loosely coupled nature i.e. the service interface is independent of the implementation. Different developers can build different applications by creating one or more services without knowing underlying implementations of the services. For example, a service can be implemented in python or .Net, and the application consuming the service can be on a different platform or language. It also helps systems to adapt while keeping service consumers separate from changes that occur while service implementation [24]. Finally, by utilizing existing software infrastructure to build new services, SOA provides better flexibility in building applications and processes in an agile manner.

3.2.2 Computational Offloading for Mobile Cloud Computing Architecture

Our proposed system includes the mobile device that consists of the applications and the cloud server that provides the functionalities of the applications as services. The communication between the service provider and service consumer is performed by the middleware. The middleware is the main part of our Offloading System. Figure 3.2 shows the proposed Offloading System structure.

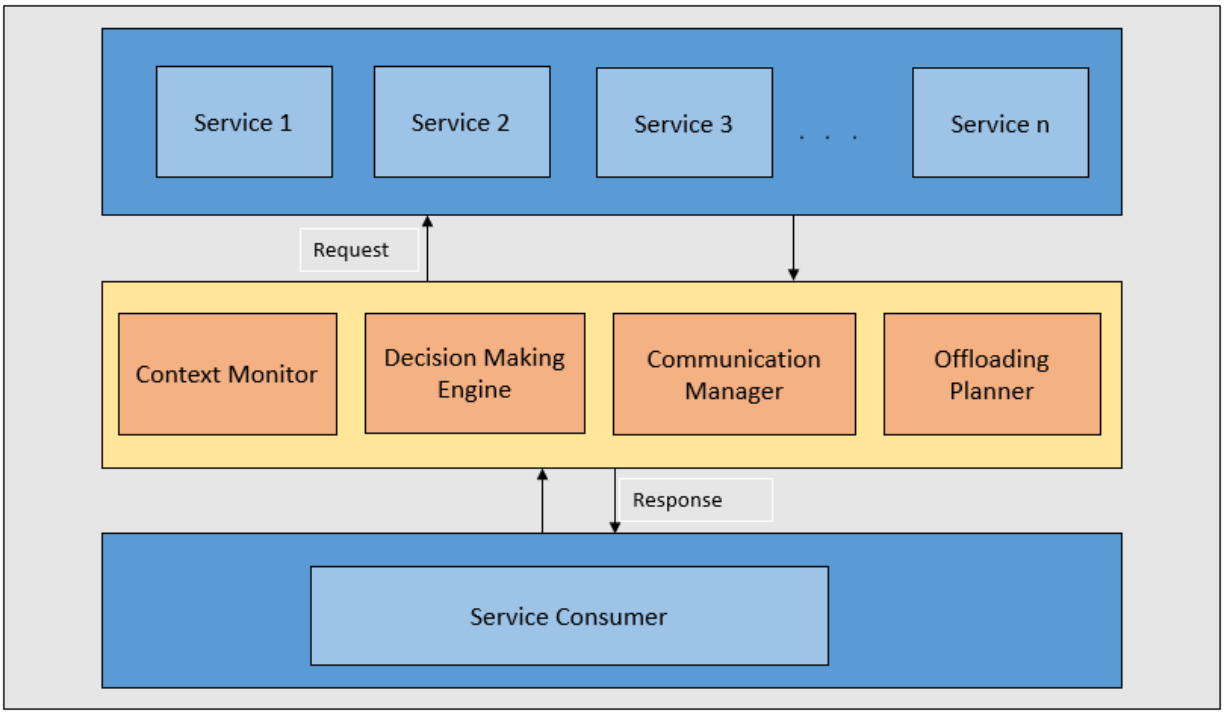

Figure 3.2: Proposed System Structure for Offloading

In this system, the middleware consists of four different modules:

1. Context Monitor
2. Decision Making Engine
3. Communication Manager
4. Offloading Planner

The next sections provide detailed information about the modules of the middleware for the Offloading System:

1. Context Monitor

The context monitor is responsible for collecting the context information and providing these values to the decision-making engine when service is executing. This context information includes:

a. The profile of mobile device: the computation power, the average CPU usage, the available memory size;
b. Mobility model at runtime: location of the car;
c. Network Condition: the availability of cell network(4G,3G) and its signal strength, the availability of Wireless network (WiFi) and signal strength

For any mobile device that the service is executing on, Context Monitor gathers this information (mentioned above) and it also updates the information from time to time. This information is then provided to the Decision Making Engine. In Figure 3.3, we have displayed the flowchart for the Context Monitor.

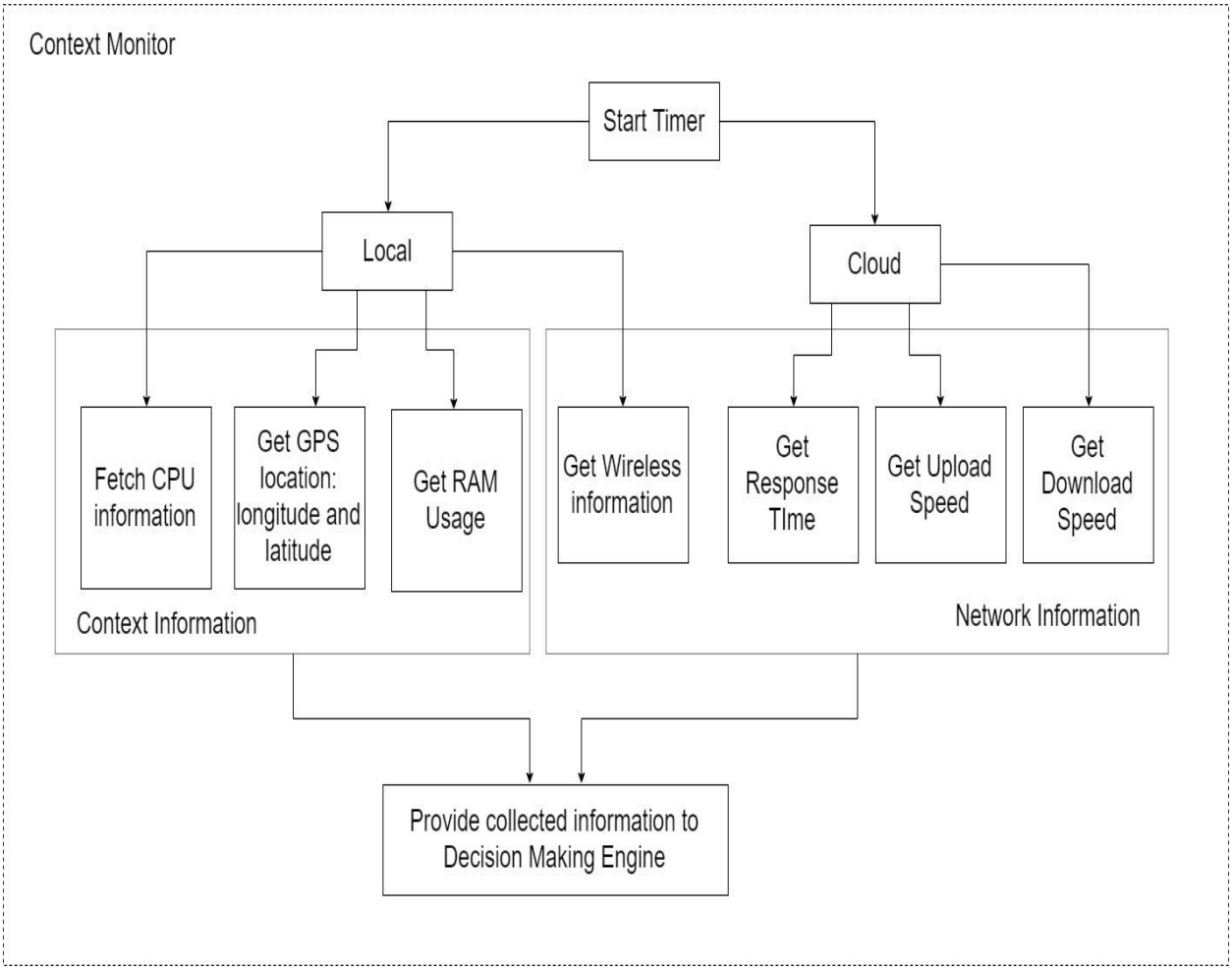

Figure 3.3: Context Monitor

2. Decision Making Engine

The Decision Making Engine first analyzes the context information, like the available network connections, computation power, the state of the vehicle, available memory size, upload speed and download speed. It then leverages the result of analysis of the context information for decision-making. For making the decision about the executing location of the service. Objective Function is used. The Objective Function is defined as follows:

$$\boldsymbol{OF} = P(s1,s2,\ldots,sn)*WP + M(s1,s2,\ldots,sn)*WM + E(s1,s2,\ldots,sn)*WE,$$

where s represents the services, P represents Performance(computation), M represents Memory, E represents Economic Cost, WP represents the weight value of Performance, WM represents the weight value of Memory and WE represents the weight value of Economic Cost.

The objective function is calculated as the weighted sum of Performance, Memory, and Economic Cost. The weight values are set according to the context of the mobile device. For example, if the device's memory is less than the threshold value then we increase the weight value in the objective function. So, the weight values are set according to the preference of any parameter at that time. The weight values are adjusted in this way to handle different units of the parameters. The sum of the weight values will always be 1.

So, for each service *p.local* and *p.cloud* is calculated, where $p = p.local = W/C$, where W is the workload of the service and C is the CPU capacity,

$p.cloud = TR + RT + TO$,

where *RT* is the response time of a cloud service;

TR is the time of uploading input data = input data size /data transferring rate;

TO is the time of downloading output data = output data size /data transferring rate;

Following that, is the calculation if Memory and Economic Cost.

Economic Cost for a service s_i is defined as:

$E(s_i) = E.Cloud(s_i) * IsOffloading$,

where *IsOffloading* is value which either 0 or 1. 0 represents that s_i is executed by a local service while 1 represents that it is executed by a cloud service.

$E.Cloud(s_i) = DI + DO$,

where *DI* is input data size of a service invocation;

DO is output data size of a service invocation.

Decision Making Engine then makes the decision of changing (if required) the weight values of computation power, memory and monetary cost in Objective Function according to varying context information. For example, if the memory on the local system is below the threshold point, the weight value of memory (*WM*), which indicates the priority of that parameter, will be increased. Finally, if the result of analysis of the current context information suggests that executing on local

is better, the Decision Making Engine makes the decision that the service should run locally, otherwise, it makes the decision to offload the service on the cloud. Hence, the decision is taken based on the Objective Function value. Figure 3.4 shows the flowchart of Decision Making Engine.

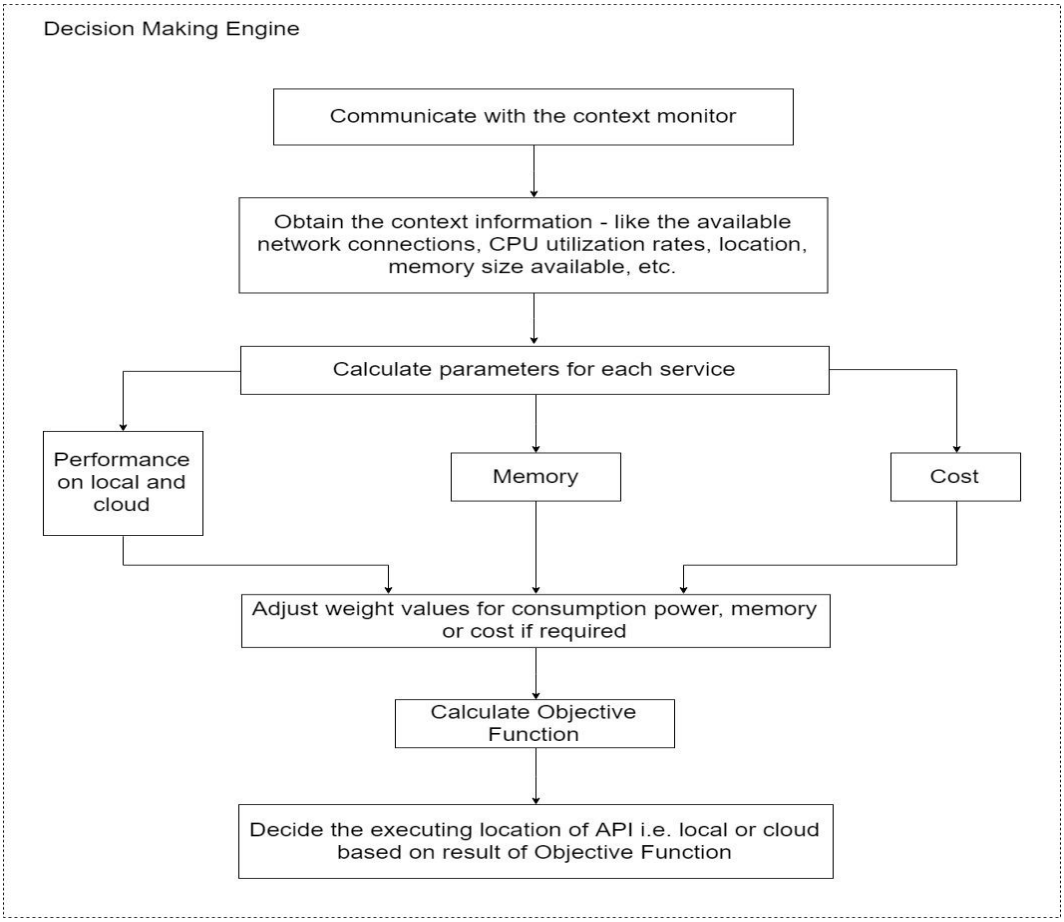

Figure 3.4: Decision Making Engine

3. Communication Manager

Communication manager is responsible for transferring the information between local and cloud. It encapsulates the tedious details of serializing data and deserializing data from one format to another. It also synchronizes the offloading plan between the mobile device and cloud.

4. Offloading planner

Offloading planner is in charge of offloading the service to the cloud. Offloading planner also selects one service, in the situation when there are more than one candidate services. It should be

noted that, as the decision of where the service runs is dependent on the context information at a certain time, an offloading plan for a certain workflow model will also vary from time to time due to the change of context information. So, the offloading planner executes the plan based on the decision and the context information. Figure 3.5 displays the flow of Communication Manager and Offloading Planner

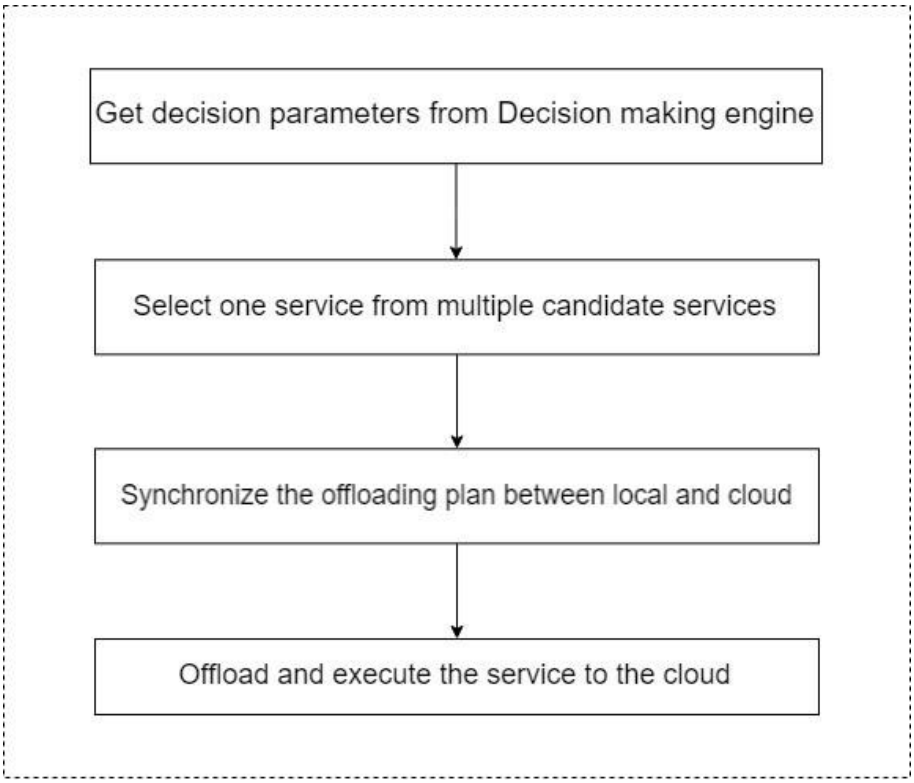

Figure 3.5: Communication Manager and Offloading Planner

3.3 Overall Flowchart

Figure 3.5 shows the overall flowchart of our Offloading System. As shown in the figure, first, the user's data plan expiration is checked. If it is expired, then the user's preference will be asked on whether user wants to allow additional charges on their account or not. If the data plan is not expired, then the process goes to next step otherwise the service is executed locally.

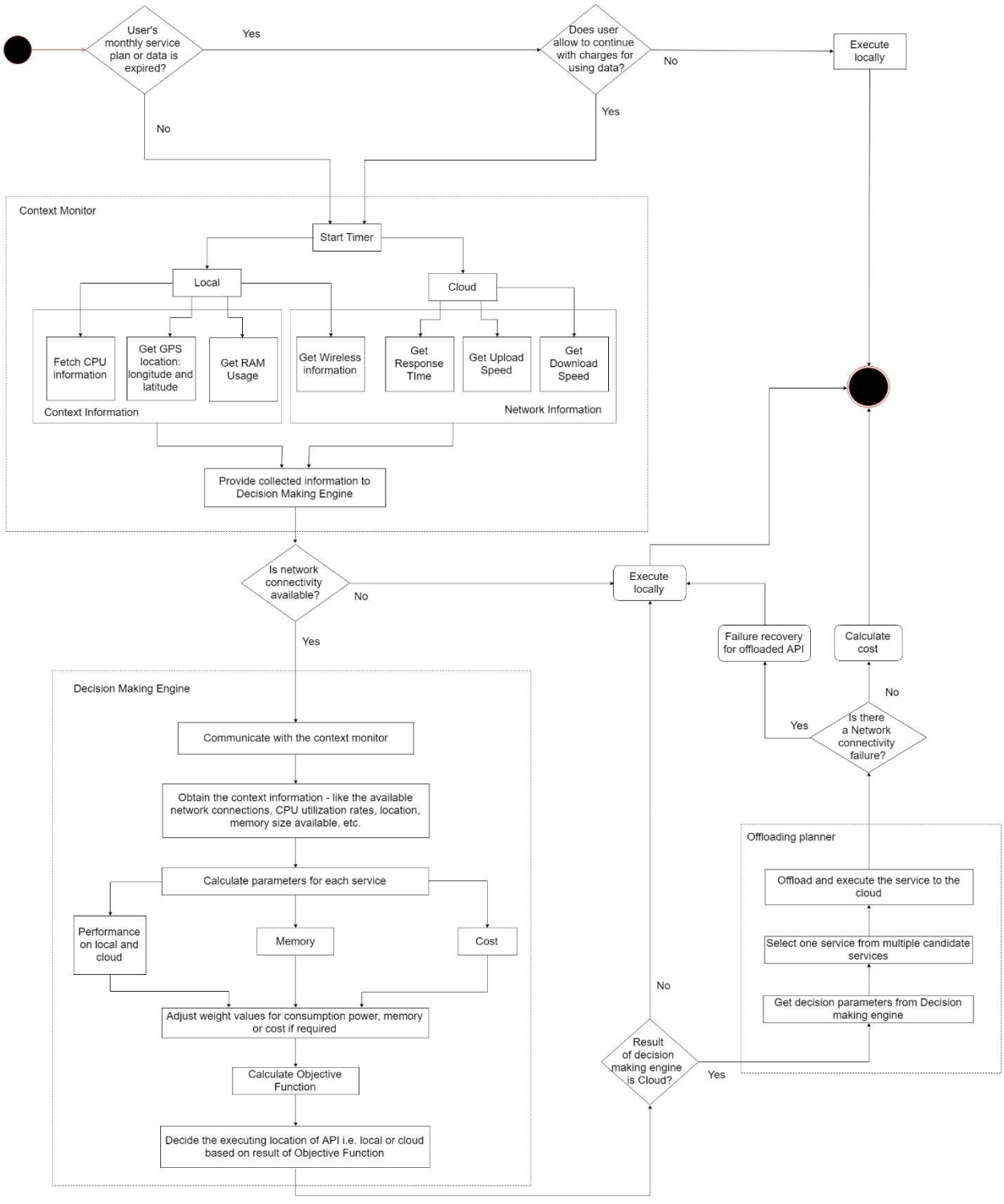

Figure 3.6: Overall Flowchart of the Offloading System

Then the Context Monitor collects all the required information like network connections, computation power, longitude and latitude of the device location, memory size available, upload

speed, download speed, etc. From the Context Monitor, the information of whether the network connection is available or not, is obtained. If it is not available, then the service will be executed locally but if it is available then the information is passed to the Decision Making Engine. The Decision Making Engine analyzes the information and obtains the values of different parameters like performance, memory and economic cost, as discussed in the previous section. Then, in the Objective Function, it adjusts the weight value of any parameter if required. Following that is the calculation of the Objective Function. Based on the result of the Objective Function, the decision of executing location (local or cloud) is obtained. If the decision is local, then the service is executed locally otherwise the information is passed to Communication Manager and Offloading Planner. Communication Manager synchronizes the offloading plan between the mobile device and cloud. Offloading Planner manages the offload plan and executes the service on cloud. Then the cost is calculated in case the data plan was expired and the user needs to be charged additionally for data usage.

CHAPTER 4

IMPLEMENTATION AND EXPERIMENTS

The proposed approach is implemented on Windows OS using C# programming language and .NET Framework. For the experiments, the approach is used on autonomous car modules. The list of software and tools used is given in Table 4.1.

4.1 Software information

The implementation of proposed methodology was performed on Dell Laptop with Intel(R) Wireless-AC 9560.

ITEM	**DETAILS**
Operating System	Windows
Languages	C#
IDE	Visual Studio 2017
Application Framework	ASP.NET
Cloud Server	Amazon Web Services

Table 4.1: List of tools used for implementation and experiments

4.2 Datasets used in the implementation

For different autonomous car modules, various datasets are used for experimentation. ShapeNet dataset [25] is used, which is a repository with the keypoint information for the different rendered views of 3D car models. DensePose [26] model which has its own manually collected ground truth dataset called the COCO-DensePose dataset is adopted and integrated into our system for 3D pose estimation of pedestrians on the road. For object verification and object elimination, the 3D virtual world has been produced using OpenStreetMap data (VGI/crowdsourced) and the façade texture from Google street view images (2D street views and satellite images) [26][27]. This 3D

environment comprises stationary (e.g. Buildings), and variable (e.g. Trees) objects. Figure 4.1 and 4.2 below displays an example of a constructed 3D virtual world.

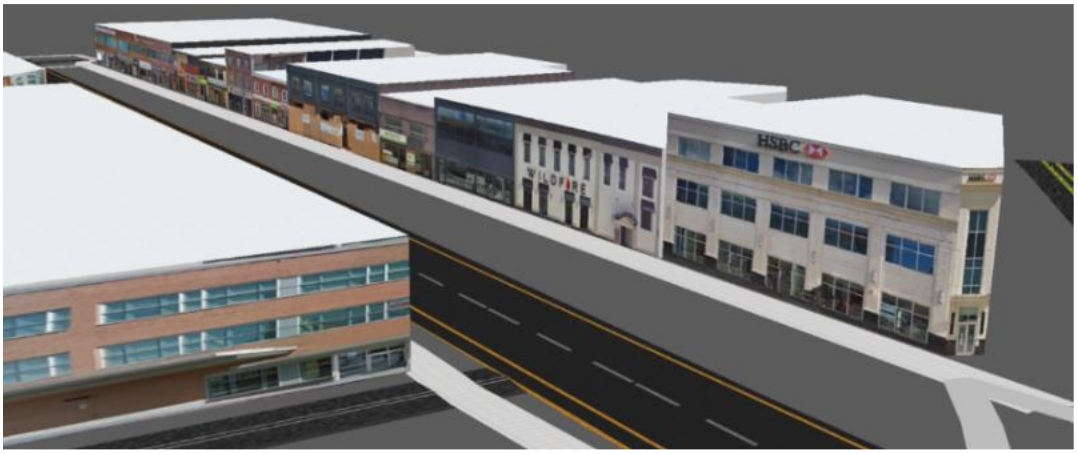

Figure 4.1: Constructed 3D Virtual World

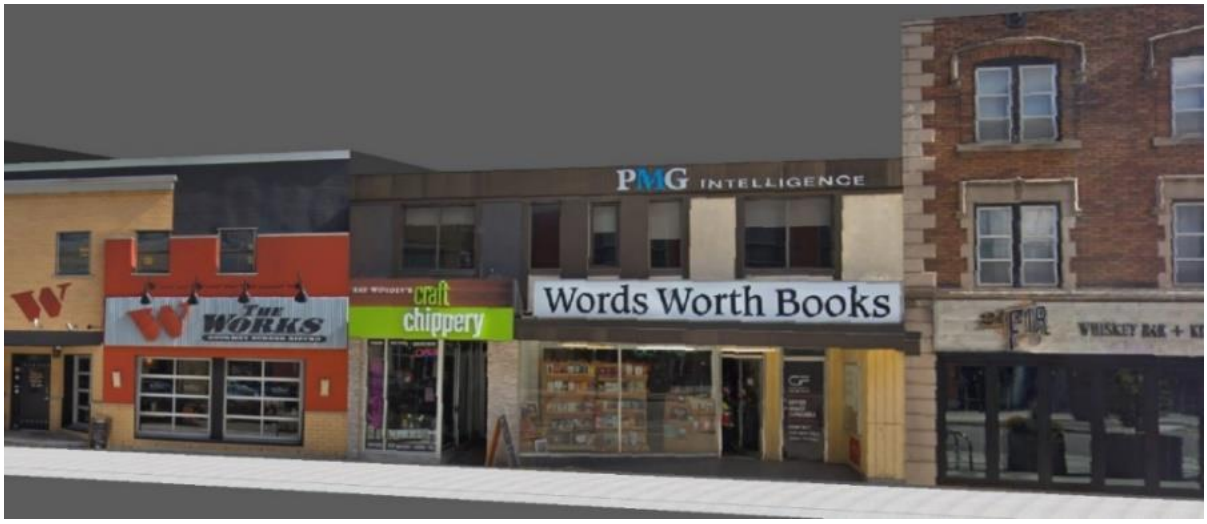

Figure 4.2: Constructed 3D Objects

4.3 A New System for Self-Driving Vehicles

As mentioned in the beginning of Chapter 4, experiments for our offloading system are performed on autonomous car modules. There are basically six different modules for this system which are interconnected with each other, as shown in Figure 4.3. Those six modules are:

1. Construction on virtual 3D environment
2. Rendered images of real-time video

3. 3D feature and keypoint extraction
4. Removal of static and variable objects
5. Dynamic object recognition
6. Dynamic object detection

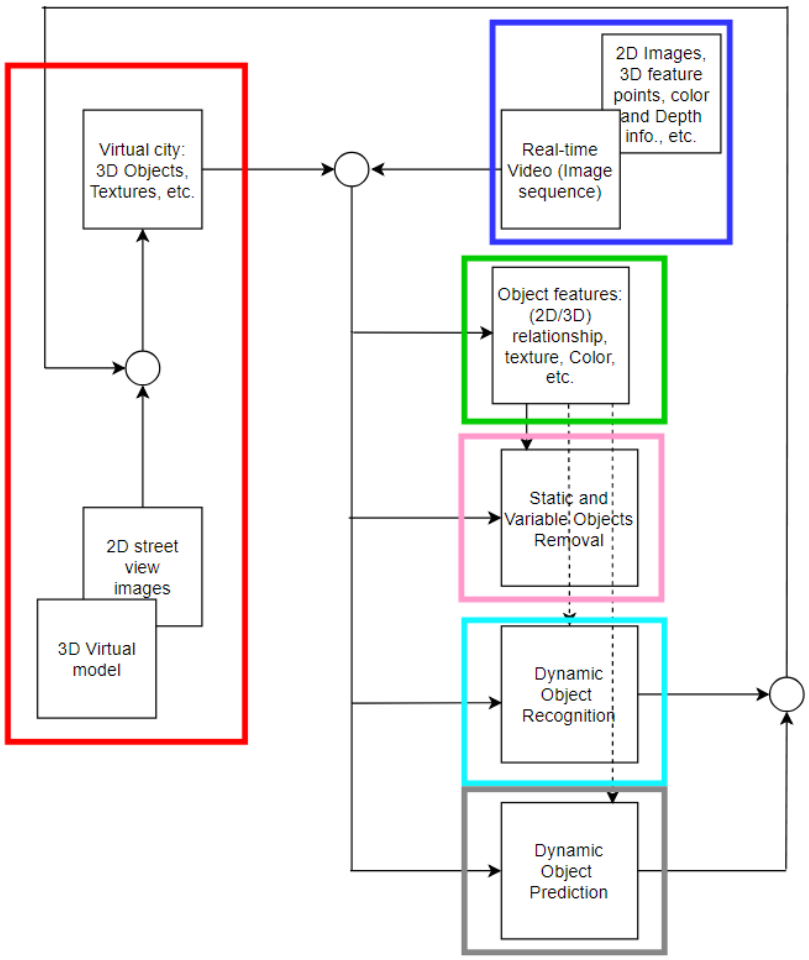

Figure 4.3: Overall system

Following is the description of the overall system in reference to Figure 4.3: the overall system first deals with the construction of a virtual 3D environment with the use of OpenStreetMap data (VGI/crowdsourced) and the façade texture from Google street view images. The virtual 3D city model consists of static objects, such as buildings, and some of the variable objects, such as trees. Apart from this, there is a separate repository which contains 3D models of dynamic objects, such as cars. The module marked in the blue-colored box in Figure 4.3 shows the real-time video (image sequences) passed as input to the system. The virtual environment is rendered, and keypoint

features and 3D features are stored in a repository; this work is performed in the module colored in pink. The module marked in yellow is the static and variable object elimination module. In this module, the keypoint features of the input image (blue module) and keypoint features of the virtual environment (pink module) are matched. Matching the keypoint features of the virtual environment and real-time image confirms the location of the car in the real-world; this solves the problem of geo-localization of the self-driving car. With the matching, location of the static objects is also confirmed, and they are eliminated from the object identification process; which provides more time for the identification and prediction of dynamic objects such as human beings or animals on the road, as those are the ones which have impact on the navigation of the self-driving system. The module marked in cyan deals with the object recognition and pose estimation of dynamic objects present in the real-time input image, such as cars. Additionally, this module tracks the recognized objects from multiple frames of the video and calculates the speed of the dynamic object. The recognized object with the pose information along with the object speed and location is used to update dynamic objects into the 3D virtual environment. The module marked in grey color updates the dynamic objects' information into the virtual environment.

4.3.1 Code Partitioning

Offloading is the process of transferring a computation or resource intensive tasks from a mobile device to a remote server or cloud resources, as discussed earlier. As this approach uses a Service-Oriented Architecture, the modules that are discussed in the previous section are wrapped as services and those services are offloaded to the cloud. Before offloading a service, the process of code partition is required on the modules. Firstly, the code is partitioned by simply marking some parts of the code as non-offloadable. Those parts are the ones that interact with any input or output device or it can be the part that implements application's UI (user interface). Then, source code analysis is performed for individual modules. During the source code analysis, if there are some sub modules that are common among different modules, then those are identified, and only single service is created for them. That service can then be shared between those modules. The final offloading is done based on the Decision Making Engine, as discussed in Chapter 3.

4.3.2 Implementation of the SOA System

After performing initial code partitioning, the modules like 3D feature and keypoint extraction, Removal of static and variable objects and Dynamic object recognition and Pose Estimation are

used for getting the sub modules. Sub modules such as Finding Key Point List, Matched Key Points, Object Detection, Pose Estimation, etc. are created and then are wrapped as services.

Following are the services created for different modules and the flow:

1. A service is created to obtain the 3D object, which has the previously constructed virtual 3D environment.
2. Then, a service (FasterRCNN) is created for object detection for static and variable objects like buildings, benches, and trees.
3. Then the feature points extraction service (CrtFeatExtraction) is created for various buildings and car models.
4. Following the object detection is the object verification service (ObjVerification), again for static and variable objects.
5. Then service (HeatmapGeneration) is created to generate the heatmaps for buildings and trees.
6. Then, object elimination service (ObjElimination) is created, which includes the heatmap information.
7. For object recognition and pose estimation of dynamic objects like cars and pedestrians, Finding Key Point List, Matched Key Points, Object Detection and Pose Estimation services are created.

The interaction of these modules as services in shown in Figure 4.4. Parallelogram represents our identified services, the services in dotted box are the common services that are shared among modules, ellipse indicates input or output data, and the different colored lines demonstrates flow of individual modules.

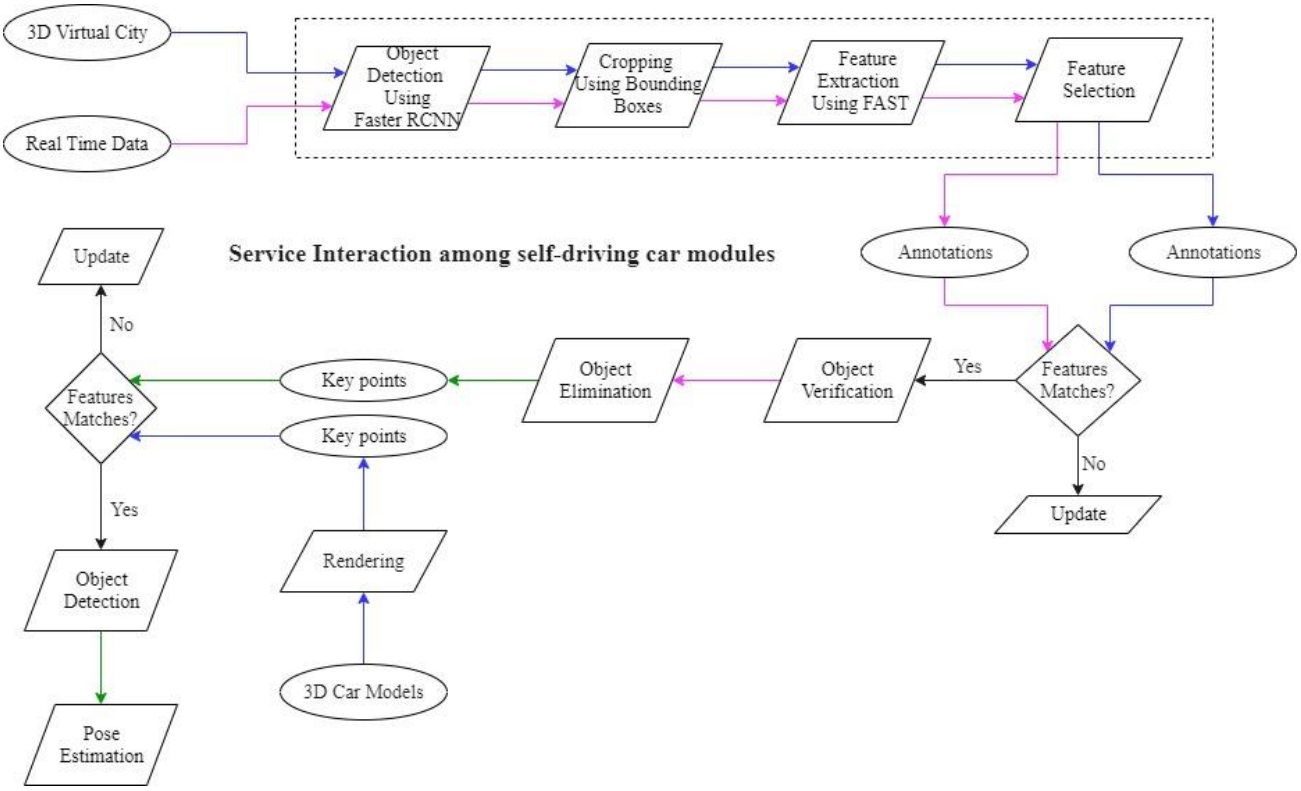

Figure 4.4: Service Interaction among self-driving car modules

For wrapping the code as service, Restful APIs are created. This means that the API is designed to allow to get, create, update, & delete objects with the Http verbs GET, POST, PUT, PATCH, & DELETE.

Http is used for communication and ASP.NET core has a middleware pipeline that is invoked for each request. For flexibility, all the paths are added in project's Web config file, so that any file can be added at that path and can be run as service. In the case that a service provides text result for e.g. the list of key points, those are displayed in XML format, following the SOA standard. In the case that image is provided as a result, it is saved/downloaded, and the path of resulting image is saved in the Web config file so that the requesting service can access the image from that path, as shown in Figure 4.5. As per industry standards (considering Uber and Tesla), the concept of Web API has been used. Important data repositories (i.e. images of car models, humans, etc.) from the modules of other students are kept in local. Less important images like wild animals are kept on the cloud so that it will not acquire memory on local.

```
  <add key="CarResults" value="C:\\Users\\Rajasi\\WebA
  <add key="BuildingResults" value="C:\\Users\\Rajasi\
  <add key="Result_Image" value="D:\\result.jpg" />
</appSettings>
```

Figure 4.5: Project's Web Config file snapshot

As mentioned in Section 4.3.1 and earlier in Section 4.3.2, some sub modules in the code has common functionalities. For this, it is redundant to create same service for different modules. Hence, a single service is created which can be used in various modules. For example, faster rcnn method is used for object detection of real time images and for images from the virtual world as well. So, a service is created named fasterRcnn, which can be shared for both the sub modules. The only difference is the input images but since the code is dynamic, the image file name is specified in the web config and then the service is called according to the requirement. Another similar example is for corner detection of the buildings code, in which, the same method is followed. Figure 4.6 displays the input image for corner detection and Figure 4.7 displays the output image which has corner points detected by fast algorithm. 32 points close to the corners are selected by this algorithm.

Figure 4.6: Input image to fast algorithm

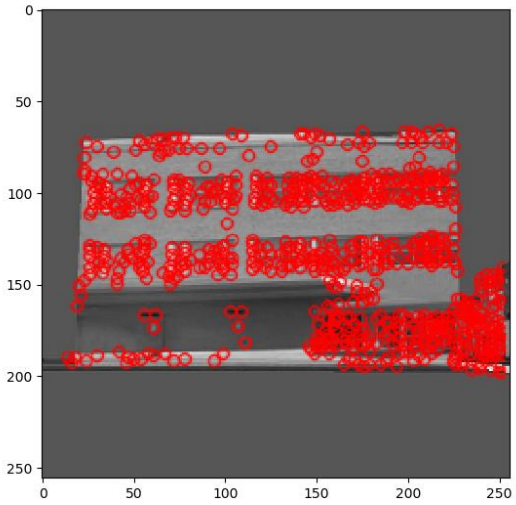

Figure 4.7: Output of fast algorithm

4.4 Implementation of Context Monitor

As mentioned in Chapter 3, the context monitor is responsible for collecting the context information which includes the profile of mobile device: the computation power, the average CPU usage, the used memory size; mobility model at runtime: location of the car and network condition: the availability of cell network(4G,3G) and its signal strength, the availability of Wireless network (Wi-Fi) and signal strength.

For obtaining this information, a form application has been created in C# and by using ASP.NET framework. Also, the information that is collected is saved in a file so that it can be monitored from time to time.

Following are the classes or methods that are used to get the relevant information:

Location: longitude and latitude using GeoCoordinate class of System.Device.Location

Wireless: wireless information using WlanClient of Managed Wifi library

Computation Power: Get computation power/cpu usage using processor information

Memory: information of RAM memory using PerformanceCounter

Economic Cost: input data size using request.ContentLength and output data size using response.GetResponseStream()

Cost: data rate from the telecom operator based on usage (per MB)

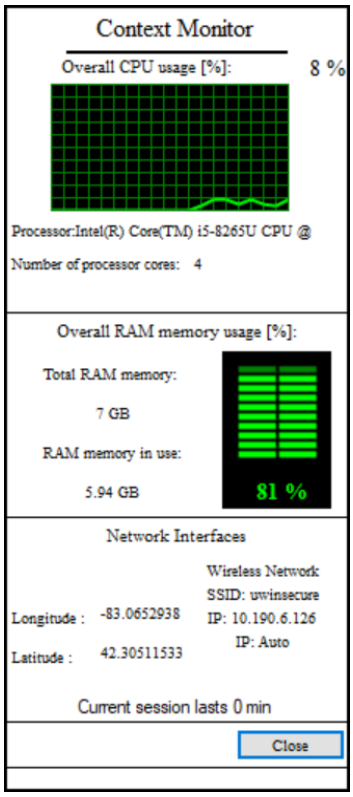

Figure 4.8: Context Monitor Output UI

Figure 4.8 shows the UI part of the Context Monitor output which includes the information mentioned above. Figure 4.9 shows the information saved in a file for monitoring purposes.

```
RAM Percentage: 81 %
Total RAM: 7 GB
RAM Usage: 5.92 GB
Wireless IP: 10.190.6.126
Wireless SSID: uwinsecure
Wired IP: Not Detected
Wired Connected: Not Detected
Longitude: -83.0652938293894
Latitude: 42.3051153385679
End Data


Start Data 2019-10-29 10:38:31 AM
CPU Usage: 8 %
Battery Value: 42 %
RAM Percentage: 81 %
Total RAM: 7 GB
RAM Usage: 5.94 GB
Wireless IP: 10.190.6.126
Wireless SSID: uwinsecure
Wired IP: Not Detected
Wired Connected: Not Detected
Longitude: -83.0652938293894
Latitude: 42.3051153385679
End Data
```

Figure 4.9: Context Monitor Output in file

4.5 Implementation of Decision Making Engine

In the Decision Making Engine, as discussed earlier, firstly we will find out if the user allows for additional charges in case the data limit is exhausted, or Wi-Fi is disconnected. Figure 4.10 shows the pop up which is displayed to the users.

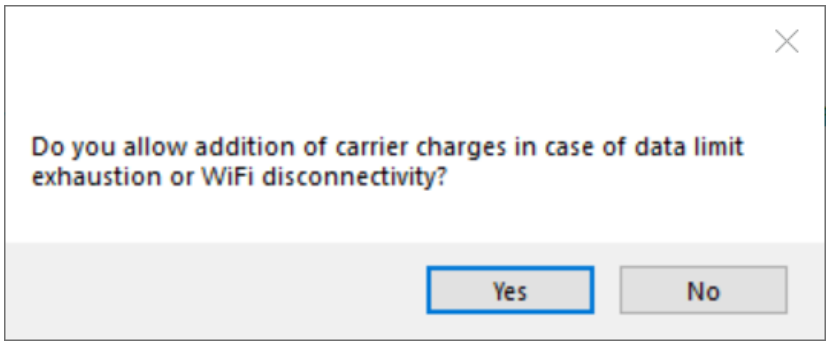

Figure 4.10: Pop up for user's consent

If the user selects 'No' then the service will be executed locally when the Wi-Fi is disconnected, or the data limit is exhausted.

We have considered following scenarios based on the working of self-driving cars:

Scenario 1: Default Scenario – This is considered for a normal situation, considering the car has just started. So, at this time, less memory is used, Wi-Fi is also connected at that time and the service executed is regarding obtaining the input images and getting the key point based on those images.

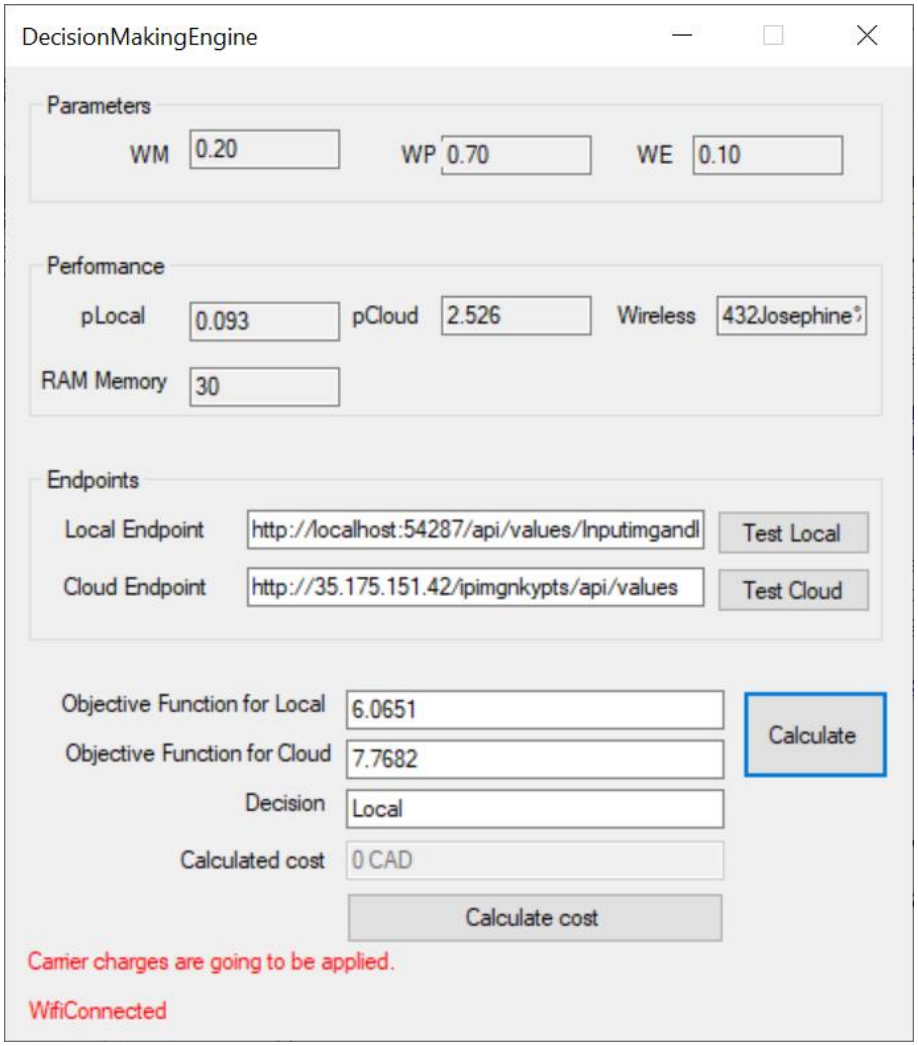

Figure 4.11: Decision Making Engine Scenario 1

As we know, there are three parameters Memory, Performance and Economic Cost. Figure 4.11 shows the result of the Decision Making Engine implementation, which has three weighted values

WM, WP and WE, representing weight values of Memory, Performance and Economic Cost respectively. The weight values indicate the priority of each parameter, by default we have considered Performance to be of highest priority i.e. 70% then the priority of Memory i.e. 20% and then Economic Cost i.e. 10%. However, if there is a requirement of changing the priority on basis of the context information of the device, then the Decision Making Engine will do that dynamically. In this way, the weight values are modified to handle different units of the parameters. The sum of the weight values will always be 100%.

Figure 4.11 also has information like Wi-Fi connectivity (bottom left corner), the user's selection in previous step (bottom left corner), Wireless SSID and the Memory used on the device. Local endpoint is the local (on the device) URL of the service. Cloud endpoint is the AWS (Amazon Web Service) URL of the service. The information about AWS will be provided in later section. The values *plocal* and *pcloud* are obtained after clicking on the buttons Test Local and Test Cloud respectively.

Here, $p.local = W/C$, where W is the workload of the service and C is the CPU capacity,

$p.cloud = TR + RT + TO$,

where RT is the response time of a cloud service;

TR is the time of uploading input data = input data size /data transferring rate;

TO is the time of downloading output data = output data size /data transferring rate;

Economic Cost for cloud is $E.Cloud = DI + DO$,

where DI is input data size of a service invocation;

DO is output data size of a service invocation.

As the Memory available in the given scenario is 30%, the weight values are not changed.

All this information is placed in the Objective Function:

$\boldsymbol{OF} = P(s1,s2,\ldots,sn)*WP + M(s1,s2,\ldots,sn)*WM + E(s1,s2,\ldots,sn)*WE$,

Based on this, the decision is obtained whether to offload the service to cloud or not. Here, the local endpoint value indicates the URL of a service, for which we are taking the decision. The URL is for an individual service in this situation, but if required, we can add a semicolon and write another URL to make the offloading decision. In that scenario, it will first execute the service one by one. After that it will calculate the decision for each individual service. Hence, after obtaining the information for local and cloud, the decision is provided.

In the given scenario, Wi-Fi connection is available, so the cost is not calculated. From the objective function results, we can see that the result indicates that service should be executed locally.

Scenario 2: Wi-Fi connected, Memory Priority – To make real-time decisions based on complex datasets, a self-driving car's AI system requires a constant, uninterrupted stream of data and instructions. This results in reduction of available memory. Hence, in the second scenario, we are considering that the memory is less than the threshold value. So, the priority is provided to weight value of Memory and decision is made based on that weight value.

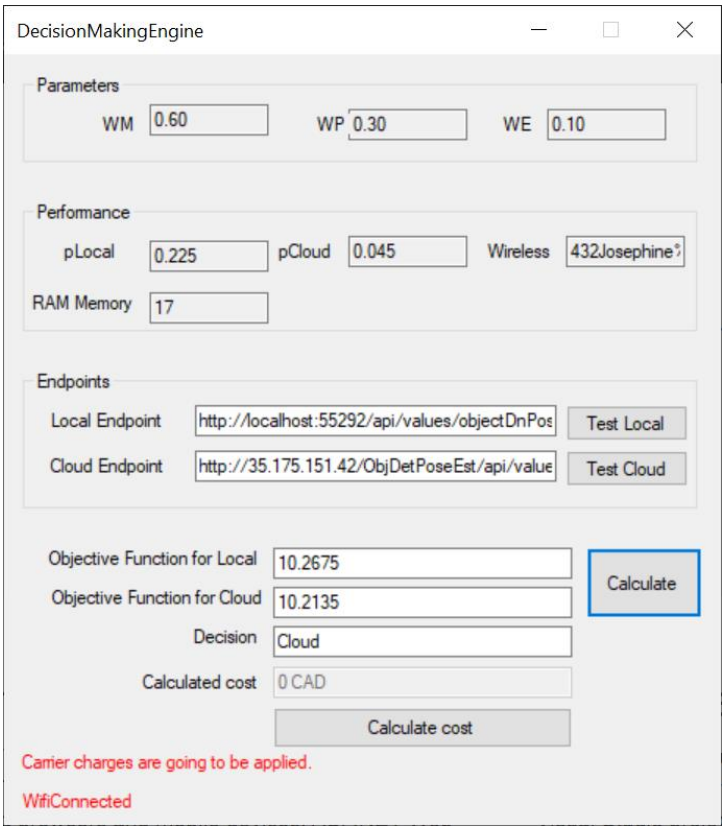

Figure 4.12: Decision Making Engine Scenario 2

Scenario 2 is shown in Figure 4.12, where Wi-Fi is connected, so cost will not be calculated. The value of Memory available is only 17% and as it is below the threshold (25%), the weight values are changed. In this case, the priority is given to WM and hence its value is 0.6 and the value of WP is 0.3. After the calculation of all the values, the result indicates that it is better to offload the service and execute it on the cloud.

Scenario 3: Wi-Fi not connected, use Mobile Data – Since self-driving cars travel through different regions, network connectivity is not always available. In a situation when the car is driving into a tunnel, connectivity will not be available for the system. Therefore, we must consider such situations and be prepared for the proper execution of services and for safety of users. In these situations, mobile data is used to run the services.

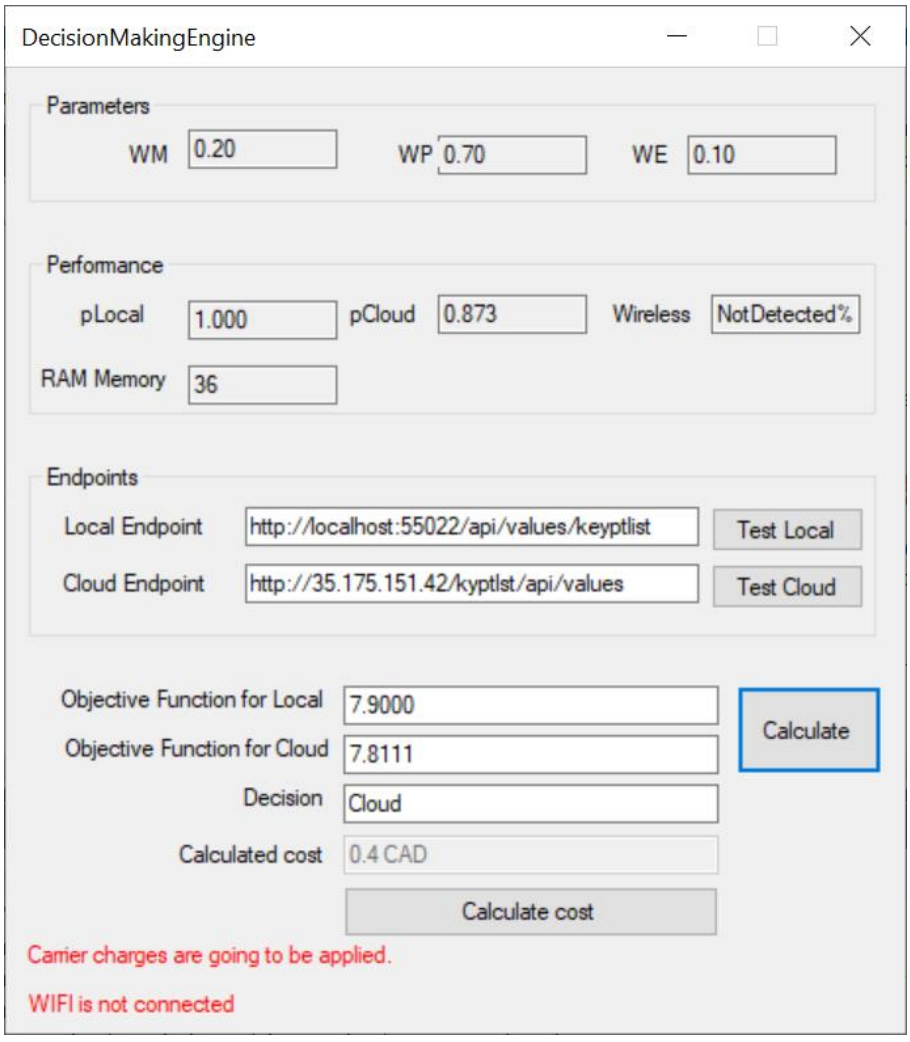

Figure 4.13: Decision Making Engine Scenario 3

In scenario 3 (Figure 4.13), the Memory available is above threshold so the weight values are not affected. However, the Wi-Fi connection is not available so cost for offloading is calculated. Based on the user's selection, we have a variable IsCarrierChargesAllowed which has value true or false. Here, the label 'Carrier charges are going to be applied' on bottom left corner indicates that the value of IsCarrierChargesAllowed = true. So, charges are calculated as per the code shown in Figure 4.14. According to the data used in MB, the cost calculated is 0.4 CAD and the decision for executing the service is cloud. Hence, the charges will be added for the user and service will be executed on cloud.

```
//START Cost calculation
originalCharge = totalUsedmb;
CalculateDataCost();
BeforeCharge= totalUsedmb;
HttpWebResponse response = (HttpWebResponse)request.GetResponse();
CalculateDataCost();
AfterCharge = totalUsedmb;
originalCharge = originalCharge + (AfterCharge - BeforeCharge);
totalUsedmb = originalCharge;
txtCalculatedCost.Text = Convert.ToString(totalUsedmb * dbWC) + " CAD ";
//END Cost calculation
```

Figure 4.14: Code for cost calculation

Scenario 4: Wi-Fi not connected, execute locally – As the services and important data are saved on car's on board unit, the services can be executed locally without the usage of internet even when the Wi-Fi connectivity is not available. In these cases, the performance is checked by the Decision Making Engine. If the performance is better on local, then service is executed locally, and mobile data will not be used.

Scenario 4 shown in Figure 4.15, is much similar to Scenario 3. The only difference is that the performance on local is better which indicates that the service will be executed locally. So, the charges will not be added for the user.

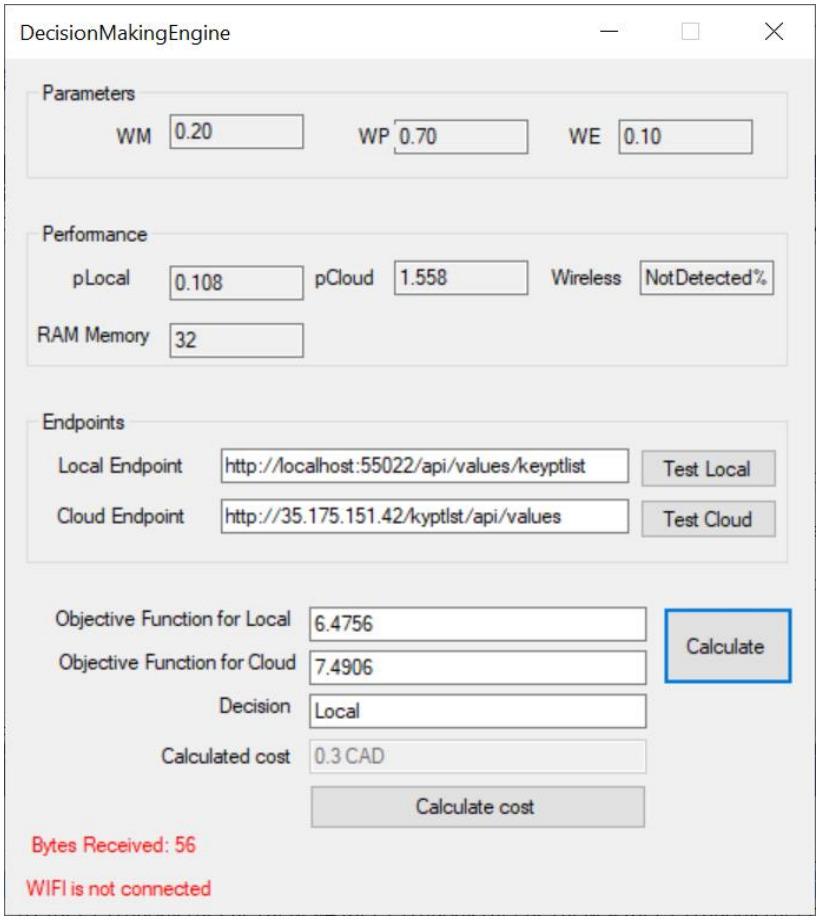

Figure 4.15: Decision Making Engine Scenario 4

Scenario 5: User declined addition of data charges – As mentioned earlier in this section that initially the user will be asked if they want to allow the addition or data charges in case of network disconnection. If the Wi-Fi or data is available then service can be executed on cloud (based on context information), but if it is not available and the user has declined the addition of charges then the service will be executed locally only.

In Scenario 5, the user declined for including additional charges, so the value of IsCarrierChargesAllowed is false. Hence, the label indicates that 'Carrier charges will not be applied' and the calculate cost button is disabled. Additionally, the Wi-Fi connection is not available. Based on both conditions, the service will be executed locally.

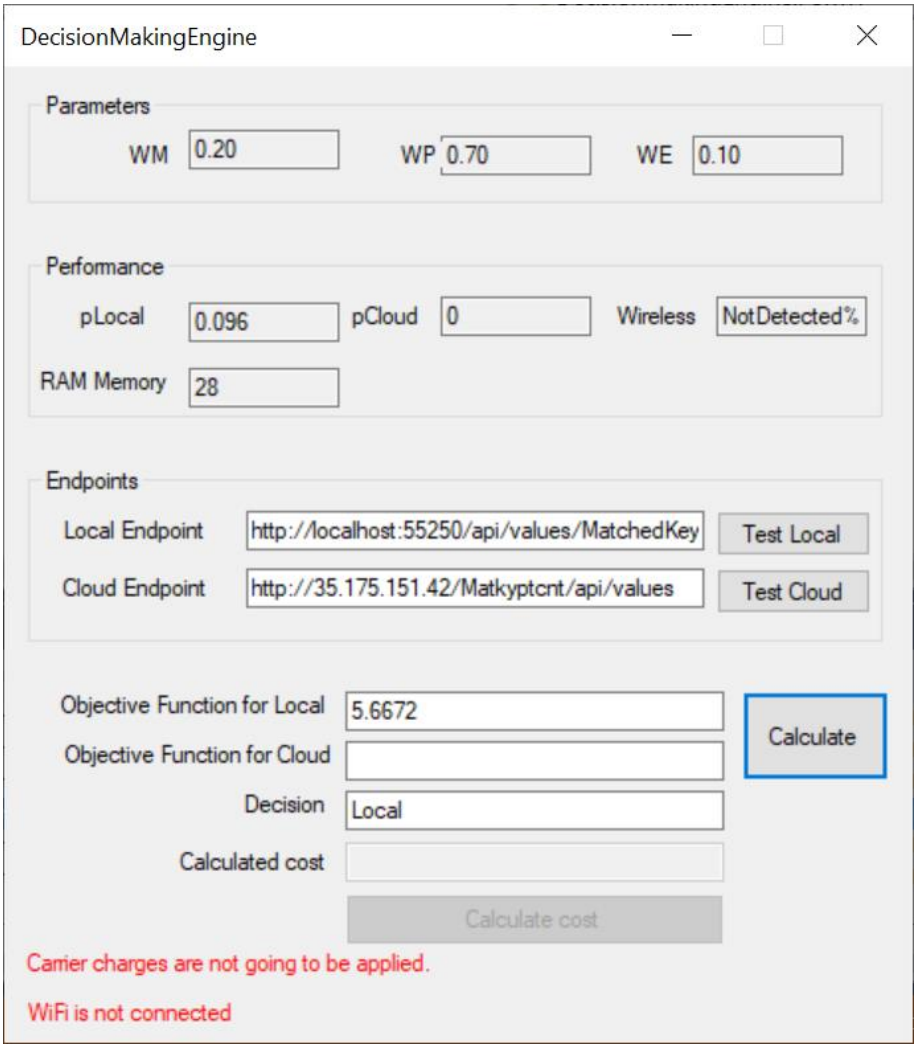

Figure 4.16: Decision Making Engine Scenario 5

These were some of the scenarios that can occur during the execution. The next section describes the process that will take place if the result of the Decision Making Engine is to execute the service on cloud.

4.6 Implementation of Communication Manager and Offloading Planner

For offloading of the service, a cloud resource Elastic Compute Cloud (EC2) of AWS is used. It provides a resizable compute capacity by allowing users to create server on the cloud. An AWS Educate account was created for this thesis, since AWS Educate is free for a year for students. A Windows AMI was used to create an EC2 instance as shown in Figure 4.17.

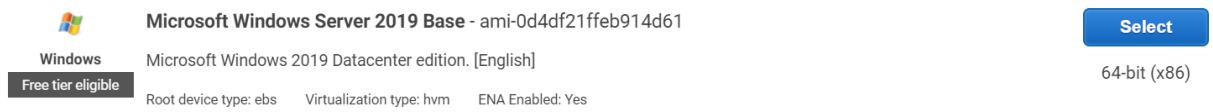

Figure 4.17: AWS EC2 Windows Server Creation

IIS (Internet Information Services) is installed and configured so that the services can be executed on the EC2 instance. For security in AWS, no incoming requests (except RDP) to this server are allowed by default. RDP is allowed for the connection to the server so that modifications can be made when required. Hence, after the creation of instance, Security Groups are needed to be configured. As shown in Figure 4.18, HTTP and HTTPS ports are added so that the APIs can be called using their URLs.

Description | **Inbound** | Outbound | Tags

Edit

Type	Protocol	Port Range	Source
HTTP	TCP	80	0.0.0.0/0
HTTP	TCP	80	::/0
RDP	TCP	3389	0.0.0.0/0
HTTPS	TCP	443	0.0.0.0/0
HTTPS	TCP	443	::/0

Figure 4.18: Security Groups Configuration

The Communication Manager is responsible for mobile-cloud interactions through a TCP socket connection over Wi-Fi or cellular network. After the information is obtained from the Decision Making Engine, if the service is needed to be offloaded to the cloud then communication takes place between the device and the cloud. Then the Offloading Planner selects the service that is needed to be offloaded.

4.7 Comparison and Discussion

Many modern researches on computational offloading for mobile cloud computing focuses on the techniques of code partitioning and offloading, however, when the self-driving car is moving, the

context information like location, available cloud resources and the network conditions changes. Therefore, it is crucial for these vehicles to be able to dynamically offload the applications and, hence perform in an effective and safe manner. Adiththan et al. [50] presents an approach for adaptive offloading for automated-driving. The author has considered network conditions and application requirements to leverage remote resources. The parameters considered in this method are the network characteristics, data transfer requirements and computation requirements. However, the important factor, i.e. Monetary Cost is not taken into account in this paper. Since the offloading of service is performed to the cloud, it is important to consider cost to understand its effect on Quality of Experience (QoE) for the user. There are various other interesting literatures [51,52], which use learning methods like reinforcement learning or adaptive learning, but in case of autonomous cars, there are new situations occurring on the roads constantly. Hence, it is not always safe to use these learning techniques. Overall, from the experimental results and comparison it can be deduced that our framework is a robust way for providing computational offloading for self-driving car applications.

CHAPTER 5

CONCLUSION AND FUTURE WORK

5.1 Conclusion

There is a great increase in the use of mobile devices like smartphones, tablets and laptops in various domains like gaming, information systems, e-learning and health-care [12]. However, there are various limitations on such devices such as processor potential, battery life, memory capacity and network connection. Even though mobile devices nowadays are having large memory and fast processors, those are not enough to perform computation and resource intensive tasks such as image recognition, natural language processing and object detection. Hence, Mobile Cloud Computing is used to bridge the gap between the mobile device limitations and the requirements of applications. MCC is an infrastructure that can extend the storage and computing capabilities of mobile devices by employing required cloud resources. MCC can be achieved using Computational Offloading, which allows the execution of computation and resource intensive tasks on remote server or cloud resources. Although there are various frameworks addressing computational offloading, some approaches use static offloading, some lack the standard architecture, and some are inefficient because of cloning and high management overhead.

Hence, the aim of the proposed system is to have a wholistic architecture using Service-Oriented Architecture in which individual applications' computing and data-intensive tasks are offloaded from a mobile device to a cloud server. In this method, tasks are remotely executed as an individual service on the cloud server. Additionally, a framework is designed for making offloading decisions based on the context of the device. The implementation of the application tasks as services is performed on AWS EC2 instance which can be used by any application on the client. Factors like Memory, Performance, Economic Cost and Cost are considered in this system. However, other frameworks consider some of these factors but not all in the same system.

Sections 4.4, 4.5 and 4.6 clearly depicts how different scenarios can be handled by using the proposed formula.

5.2 Future work

This research work provides some more possibilities for further improvement:

1. As it is an autonomous car system, for better performance and safety of the individuals, optimization can be performed on the system.
2. A Service Level Agreement can be negotiated between service provider and consumer, so that the Quality of Service can be improved.

REFERENCES

[1] Rahimi, M. R., Ren, J., Liu, C. H., Vasilakos, A. V., & Venkatasubramanian, N. (2014). Mobile cloud computing: A survey, state of art and future directions. *Mobile Networks and Applications*, *19*(2), 133-143.

[2] Bhattacharya, A., & De, P. (2017). A survey of adaptation techniques in computation offloading. *Journal of Network and Computer Applications*, *78*, 97-115.

[3] Kumar, R., & Rajalakshmi, S. (2013, December). Mobile cloud computing: Standard approach to protecting and securing of mobile cloud ecosystems. In *2013 International Conference on Computer Sciences and Applications* (pp. 663-669). IEEE.

[4] Khandelwal, G. Introduction to Service Oriented Architecture. Retrieved from https://www.c-sharpcorner.com/UploadFile/govind77/introduction-to-service-oriented-architecture/

[5] Ashok, A., Steenkiste, P., & Bai, F. (2015, September). Enabling vehicular applications using cloud services through adaptive computation offloading. In *Proceedings of the 6th International Workshop on Mobile Cloud Computing and Services* (pp. 1-7). ACM.

[6] Cuervo, E., Balasubramanian, A., Cho, D. K., Wolman, A., Saroiu, S., Chandra, R., & Bahl, P. (2010, June). MAUI: making smartphones last longer with code offload. In *Proceedings of the 8th international conference on Mobile systems, applications, and services* (pp. 49-62). ACM.

[7] Kosta, S., Aucinas, A., Hui, P., Mortier, R., & Zhang, X. (2012, March). Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading. In *2012 Proceedings IEEE Infocom* (pp. 945-953). IEEE.

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

[9] Chun, B. G., Ihm, S., Maniatis, P., Naik, M., & Patti, A. (2011, April). Clonecloud: elastic execution between mobile device and cloud. In *Proceedings of the sixth conference on Computer systems* (pp. 301-314). ACM.

[10] Gordon, M. S., Hong, D. K., Chen, P. M., Flinn, J., Mahlke, S., & Mao, Z. M. (2015). Tango: accelerating mobile applications through flip-flop replication. *GetMobile: Mobile Computing and Communications*, *19*(3), 10-13.

[11] Mahmoodi, S. E., Uma, R. N., & Subbalakshmi, K. P. (2016). Optimal joint scheduling and cloud offloading for mobile applications. *IEEE Transactions on Cloud Computing*.

[12] Akherfi, K., Gerndt, M., & Harroud, H. (2018). Mobile cloud computing for computation offloading: Issues and challenges. *Applied computing and informatics*, *14*(1), 1-16.

[13] Chen, X., Chen, S., Zeng, X., Zheng, X., Zhang, Y., & Rong, C. (2017). Framework for context-aware computation offloading in mobile cloud computing. *Journal of Cloud Computing*, *6*(1), 1.

[14] Boukerche, A., Guan, S., & Grande, R. E. D. (2019). Sustainable Offloading in Mobile Cloud Computing: Algorithmic Design and Implementation. *ACM Computing Surveys (CSUR)*, *52*(1), 11.

[15] Zhou, B., & Buyya, R. (2018). Augmentation techniques for mobile cloud computing: A taxonomy, survey, and future directions. *ACM Computing Surveys (CSUR)*, *51*(1), 13.

[16] Chen, X., Jiao, L., Li, W., & Fu, X. (2015). Efficient multi-user computation offloading for mobile-edge cloud computing. *IEEE/ACM Transactions on Networking*, *24*(5), 2795-2808.

[17] Deng, S., Huang, L., Taheri, J., & Zomaya, A. Y. (2014). Computation offloading for service workflow in mobile cloud computing. *IEEE Transactions on Parallel and Distributed Systems*, *26*(12), 3317-3329.

[18] Hani, Q. B., & Dichter, J. P. (2017). Energy-efficient service-oriented architecture for mobile cloud handover. *Journal of Cloud Computing*, *6*(1), 9.

[19] Parsa, S., & Ghods, L. (2008, December). A new approach to wrap legacy programs into web services. In *2008 11th International Conference on Computer and Information Technology* (pp. 442-447). IEEE.

[20] Kovachev, D., Yu, T., & Klamma, R. (2012, July). Adaptive computation offloading from mobile devices into the cloud. In *2012 IEEE 10th International Symposium on Parallel and Distributed Processing with Applications* (pp. 784-791). IEEE.

[21] How Do Genetic Algorithms Work? | Two Minute Papers #32 .... https://www.youtube.com/watch?v=ziMHaGQJuSI

[22] Akherfi, K., Gerndt, M., & Harroud, H. (2018). Mobile cloud computing for computation offloading: Issues and challenges. *Applied computing and informatics*, *14*(1), 1-16.

[23] Wikipedia contributors. (2019, September 24). Service-oriented architecture. In *Wikipedia, The Free Encyclopedia*. Retrieved 11:53, October 26, 2019, from https://en.wikipedia.org/w/index.php?title=Serviceoriented_architecture&oldid=917681137

[24] Kodali, R. R. (2005, June 13). What is service-oriented architecture? Retrieved from https://www.javaworld.com/article/2071889/soa/what-is-service-oriented-architecture.html

[25] Chang, A. X., Funkhouser, T., Guibas, L., Hanrahan, P., Huang, Q., Li, Z., ... & Xiao, J. (2015). Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*.

[26] Alp Güler, R., Neverova, N., & Kokkinos, I. (2018). Densepose: Dense human pose estimation in the wild. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 7297-7306).

[27] Goetz, M.; Zipf, A. (2012). Towards defining a framework for the automatic derivation of 3D CityGML models from volunteered geographic information. *Int.*

[28] Over, M., Schilling, A., Neubauer, S. & Zipf, A. (2010). Generating web-based 3D City Models from OpenStreetMap: The current situation in Germany. Computer Environment and Urban System (CEUS), vol. 34(6), pp. 496–507.

[29] Ashok, A., Steenkiste, P., & Bai, F. (2018). Vehicular cloud computing through dynamic computation offloading. *Computer Communications*, *120*, 125-137.

[30] Chen, L., Mislove, A., & Wilson, C. (2015, October). Peeking beneath the hood of uber. In *Proceedings of the 2015 Internet Measurement Conference* (pp. 495-508). ACM.

[31] Clement, S. J., McKee, D. W., & Xu, J. (2017, April). Service-oriented reference architecture for smart cities. In *2017 IEEE symposium on service-oriented system engineering (SOSE)*(pp. 81-85). IEEE.

[32] Helmlin, M. (2017). Service-oriented Architectures and Ethernet in Vehicles: Towards Data Centers on Wheels with Model-based Methods. *PREEvision technical Article*.

[33] MicroVision. (2016, November 28). MEMS and Sensors in Automotive Applications on the Road to Autonomous... Retrieved from https://www.slideshare.net/MicroVision/mems-and-sensors-in-automotive-applications-on-the-road-to-autonomous-vehicles-hud-and-adas

[34] https://developer.uber.com/docs/riders/references/api

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

[36] Yeshodara, N. S., Nagojappa, N. S., & Kishore, N. (2014). Cloud Based Self Driving Cars. *2014 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM)*.

[37] Bajpai, A., & Nigam, S. (2017). A Study on the Techniques of Computational Offloading from Mobile Devices to Cloud. *Advances in Computational Sciences and Technology*, *10*(7), 2037-2060.

[38] Long, C. A. I., Kunasekaran, K. K. H., Ramakrishnan, V., & Rajan, P. Task Offloading to the Cloud by Using Cuckoo Model for Minimizing Energy Cost.

[39] Vinnamala, S. (2018). Challenges and Future Research Directions in Mobile Cloud Computing. *International Journal of Technical Innovation in Modern Engineering & Science (IJTIMES)*.

[40] Noor, T. H., Zeadally, S., Alfazi, A., & Sheng, Q. Z. (2018). Mobile cloud computing: Challenges and future research directions. *Journal of Network and Computer Applications*, *115*, 70-85.

[41] Mohalik, S. K., D'Souza, M., & Jayaraman, M. B. (2017, February). Developmental aspects of Intelligent Adaptive Systems (DIAS). In *Proceedings of the 10th Innovations in Software Engineering Conference* (pp. 221-222). ACM.

[42] Feljan, A. V., Mohalik, S. K., Jayaraman, M. B., & Badrinath, R. (2015, December). SOA-PE: a service-oriented architecture for planning and execution in cyber-physical systems. In *2015 International Conference on Smart Sensors and Systems (IC-SSS)* (pp. 1-6). IEEE.

[43] Yogesh Dumbare, Kahate(2017). Computational Offloading Framework for Mobile Cloud Computing. *International Journal of Current Research*

[44] Nawrocki, P., & Sniezynski, B. (2017). Autonomous Context-Based Service Optimization in Mobile Cloud Computing. *Journal of Grid computing*, *15*(3), 343-356.

[45] Xu, H., Lin, J., & Yu, W. (2017). Smart Transportation Systems: Architecture, Enabling Technologies, and Open Issues. *SpringerBriefs in Computer Science Secure and Trustworthy Transportation Cyber-Physical Systems,* 23-49.

[46] Kumar, K., Liu, J., Lu, Y. H., & Bhargava, B. (2013). A survey of computation offloading for mobile systems. *Mobile Networks and Applications*, *18*(1), 129-140.

[47] Shiraz, M., & Gani, A. (2014). A lightweight active service migration framework for computational offloading in mobile cloud computing. *The Journal of Supercomputing*, *68*(2), 978-995.

[48] Helmlin, M. (2017). Service-oriented Architectures and Ethernet in Vehicles: Towards Data Centers on Wheels with Model-based Methods. *PREEvision technical Article*.

[49] Pierre, S. (2001). Mobile computing and ubiquitous networking: concepts, technologies and challenges. *Telematics and Informatics*, *18*(2-3), 109-131.

[50] Adiththan, A., Ramesh, S., & Samii, S. (2018, March). Cloud-assisted control of ground vehicles using adaptive computation offloading techniques. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)* (pp. 589-592). IEEE.

[51] Ning, Z., Dong, P., Wang, X., Rodrigues, J. J., & Xia, F. (2019). Deep reinforcement learning for vehicular edge computing: An intelligent offloading system. *ACM Transactions on Intelligent Systems and Technology (TIST)*, *10*(6), 60.

[52] Sun, Y., Guo, X., Song, J., Zhou, S., Jiang, Z., Liu, X., & Niu, Z. (2019). Adaptive Learning-Based Task Offloading for Vehicular Edge Computing Systems. *IEEE Transactions on Vehicular Technology*, *68*(4), 3061-3074.

[53] K. Yang, S. Ou, H.-H. Chen, On effective offloading services for resource-constrained mobile devices running heavier mobile internet applications, Commun. Mag. IEEE 46 (1) (2008) 56–63.

VITA AUCTORIS

NAME:	Rajasi Dhiman Upadhyay
PLACE OF BIRTH:	Mumbai, India
YEAR OF BIRTH:	1993
EDUCATION:	Bachelor of Engineering, 2011-2015 Gujarat Technological University, Ahmedabad, Gujarat, India Master of Science in Computer Science, co-op, 2017-2019 University of Windsor, Windsor, ON