CAPITAL UNIVERSITY OF SCIENCE AND TECHNOLOGY, ISLAMABAD

Cost and Time Efficient Resource Aware Scheduling in Fog Computing Environment

by

Jawad Usman Arshed

A dissertation submitted in partial fulfillment for the degree of Doctor of Philosophy

in the

Faculty of Computing
Department of Computer Science

2023

Cost and Time Efficient Resource Aware Scheduling in Fog Computing Environment

By

Jawad Usman Arshed

(DCS161007)

DEPARTMENT OF COMPUTER SCIENCE

CAPITAL UNIVERSITY OF SCIENCE AND TECHNOLOGY

ISLAMABAD

2023

Dedicated to the memory of my LATE Father

Muhammad Arshed

CAPITAL UNIVERSITY OF SCIENCE & TECHNOLOGY
ISLAMABAD

Expressway, Kahuta Road, Zone-V, Islamabad
Phone:+92-51-111-555-666 Fax: +92-51-4486705
Email: info@cust.edu.pk Website: https://www.cust.edu.pk

CERTIFICATE OF APPROVAL

This is to certify that the research work presented in the thesis, entitled "**Cost and Time Efficient Resource Aware Scheduling in Fog Computing Environment**" was conducted under the supervision of **Dr. Mohammad Masroor Ahmed**. No part of this thesis has been submitted anywhere else for any other degree. This thesis is submitted to the **Department of Computer Science, Capital University of Science and Technology** in partial fulfillment of the requirements for the degree of Doctor in Philosophy in the field of **Computer Science.** The open defence of the thesis was conducted on **June 02, 2023**.

Student Name :	Jawad Usman Arshed (DCS161007)	[illegible]

The Examination Committee unanimously agrees to award PhD degree in the mentioned field.

Examination Committee :

(a)	External Examiner 1:	Dr. Muhammad Zubair, Professor Riphah Int. University, Islamabad	[illegible]
(b)	External Examiner 2:	Dr. Ayyaz Hussain Professor QAU, Islamabad	[illegible]
(c)	Internal Examiner :	Dr. Nayyer Masood Professor CUST, Islamabad	[illegible]

Supervisor Name :	Dr. Mohammad Masroor Ahmed Associate Professor CUST, Islamabad	[illegible]
Name of HoD :	Dr. Abdul Basit Siddiqui Associate Professor CUST, Islamabad	[illegible]
Name of Dean :	Dr. Muhammad Abdul Qadir Professor CUST, Islamabad	[illegible]

AUTHOR'S DECLARATION

I, **Jawad Usman Arshed (Registration No. DCS161007)**, hereby state that my PhD thesis titled, **'Cost and Time Efficient Resource Aware Scheduling in Fog Computing Environment'** is my own work and has not been submitted previously by me for taking any degree from Capital University of Science and Technology, Islamabad or anywhere else in the country/ world.

At any time, if my statement is found to be incorrect even after my graduation, the University has the right to withdraw my PhD Degree.

[illegible]

(Jawad Usman Arshed)

Dated: 02 June, 2023

Registration No: DCS161007

PLAGIARISM UNDERTAKING

I solemnly declare that research work presented in the thesis titled "**Cost and Time Efficient Resource Aware Scheduling in Fog Computing Environment**" is solely my research work with no significant contribution from any other person. Small contribution/ help wherever taken has been duly acknowledged and that complete thesis has been written by me.

I understand the zero tolerance policy of the HEC and Capital University of Science and Technology towards plagiarism. Therefore, I as an author of the above titled thesis declare that no portion of my thesis has been plagiarized and any material used as reference is properly referred/ cited.

I undertake that if I am found guilty of any formal plagiarism in the above titled thesis even after award of PhD Degree, the University reserves the right to withdraw/ revoke my PhD degree and that HEC and the University have the right to publish my name on the HEC/ University Website on which names of students are placed who submitted plagiarized thesis.

[illegible]

(Jawad Usman Arshed)

Dated: 02 June, 2023 Registration No: DCS161007

List of Publications

It is certified that following publication(s) have been made out of the research work that has been carried out for this dissertation:-

1. **Arshed, J. U**, and Ahmed, M., RACE: Resource Aware Cost-Efficient Scheduler for Cloud Fog Environment, *IEEE Access, Volume 9, Pages: 65688-65701*, 2021.
   DOI: 10.1109/ACCESS.2021.3068817
2. **Arshed, J. U**, Ahmed, M., Muhammad, T, Afzal., M, Arif., M, & Bazezew., GA-IRACE:Genetic Algorithm-Based Improved Resource Aware Cost Efficient Scheduler for Cloud Fog Computing Environment, *Wireless Communications and Mobile Computing, Volume 2022.*
   https://doi.org/10.1155/2022/6355192

(Jawad Usman Arshed)
Registration No: DCS161007

Acknowledgement

First and foremost, I thank Almighty **ALLAH** for bestowing me the strength, knowledge and ability for completing this study. Secondly, I wish to pay my gratitude to my respected teachers and especially to my supervisor Dr. Mohammad Masroor Ahmed for his guidance, and moral support during the whole process. It would not be wrong to say that without him it was impossible to accomplish this task. Furthermore, I am thankful to Dr. Yasir Nauman for his continuous guidance and discussion in *(PCN)* research lab, from my synopsis defense to the paper writing. Lastly, I would like to express my deepest gratitude for the spirit of moral courage and support I received from my late father, mother, and wives in achieving this great milestone.

(Jawad Usman Arshed)

Abstract

With the growth of *(IoT)* devices, the amount of data produced daily has increased significantly. This growing amount of data demands Cloud computing services to process, store, and analyze it. Furthermore, real-time applications, for example, smart health, online gaming, and traffic management, cannot work with high latency. The use of Cloud services for such applications increases the execution delay with an increase in bandwidth usage from end users to the Cloud and the Cloud services cost. Fog computing is one of the emerging architectures designed to give quality service to applications by moving Cloud services closer to the edge of the network. The heterogeneous and distributed nature of resources at the fog layer makes it challenging to give an efficient scheduler in the fog layer, which reduces the latency for applications with minimum bandwidth and the monetary cost for these applications. This study is an attempt towards efficient scheduling at the fog layer. Broadly this study presents a novel idea of Resource Aware scheduling *(RACE)*, which organizes the application modules in a fog environment, taking into account both the computational capabilities of fog devices and the computational demands of the applications. The proposed approach consists of two parts: One part categorizes the incoming application modules based on their computation and bandwidth requirement and makes a prioritized list of modules for placement at the fog layer. The second part schedule the modules from fog to the Cloud layer as per the defined priority. To stabilize the idea, a large range of experiments is carried out using the *iFogSim* simulator, which shows that, in most circumstances, our method surpasses the traditional Cloud placement and baseline methodology. Increase in number of application modules affects the execution time of applications with an increase in bandwidth and monetary cost of applications. On the other hand the limited capacity at the network's edge, may increase the delay for the applications, whereas sending modules to the Cloud can increase the cost and bandwidth. Therefore, there is a need for schedulers that can provide a balanced solution for connecting parameters. The second part of the thesis proposes a modified nature-inspired approach *GA-IRACE* (Improved

Resource Aware Cost Efficient Scheduler) for Cloud fog environment. The optimized placement of application modules from fog to the Cloud layer improves the execution time of applications. The results demonstrate a 15-40% improvement in execution time, cost, and bandwidth usage when using the proposed improved *GA*-based approach.

Contents

List of Figures

List of Tables

Abbreviations

BT	Bandwidth Threshold
CFP	Cloud Fog Placement
CLOUD	CLOUD Datacenters
COP	Cloud Only Placement
CP	Computation Power
CR	Computation Requirement
CT	Comppputation Threshold
EFT	Estimated Finish Time
ET	Estimated Time
FD	Fog Device
FOP	Fog Only Placement
GA	Genetic Algorithm
GA-IRACE	Genetic Algorithm Based Improved Resource Aware Cost Efficient Scheduler
IOT	Internet of Things
MI	Millions of Instruction
MIPS	Millions of Instruction per second
RACE	Resource Aware Cost Efficient Scheduler
REM(M)	Remaining Module
ST	Start Time
TCP	Traditional Cloud Placement

Symbols

C_{cm}	Communication Cost
C_p	Processing Cost
$Max_{(g)}$	Maximum Generations
P	Population
M_R	Remaining Modules

Chapter 1

Introduction

The emergence of Internet of Things *(IoT)* has paved the way for new communities that bridge the gap between people and their technological accouterments. With its potential to improve metropolitan administration, e-health, and incident management, *IoT* is an intriguing opportunity for public investment to better people's quality of life. By 2025, the *IoT* might have an annual effect of about 11 trillion dollars, equal to 11% of the global economy [1]. As the *IoT* has expanded rapidly, the number of connected devices has skyrocketed [2].

According to a 2015 press release[3], there are now more internet-connected devices than people worldwide. This staggering figure is projected to reach 38.5 billion by 2020 [3], up to around five connected gadgets per individual. Increasing Internet-enabled gadgets have accumulated a massive amount of data, which requires substantial computing capacity to process.

In recent years, it has been proposed that Cloud computing [4] provides data service consumers with flexible and efficient services. The idea behind "the Cloud" is to centralize resources like servers and data repositories, linking super-fast computers with high-speed networks. As an infrastructure-level service, Cloud computing can accommodate the ever-increasing data storage and processing requirements of *IoT*-based devices and gadgets. *IoT* gadgets often benefit from cloud-based computing resources, but this is not always the case.

While cloud-based computing resources are often the best for IoT gadgets, this is not always the case. Most healthcare, emergency response, video streaming, and other geo-distributed IoT applications have stringent latency requirements that aren't effectively met by Cloud data centers [5]. In response to these issues, Cisco coined the concept of Fog [6] in 2012.

Fog is a kind of distributed computing that bridges the gap between central cloud infrastructure and edge devices used in the IoT. For Cloud-based services to be brought closer to IoT devices, fog provides storage and computation capabilities [7]. Figure 1.1 depicts a typical fog environment comprising of classic networking components such as routers, proxy servers, base stations, switches, etc.

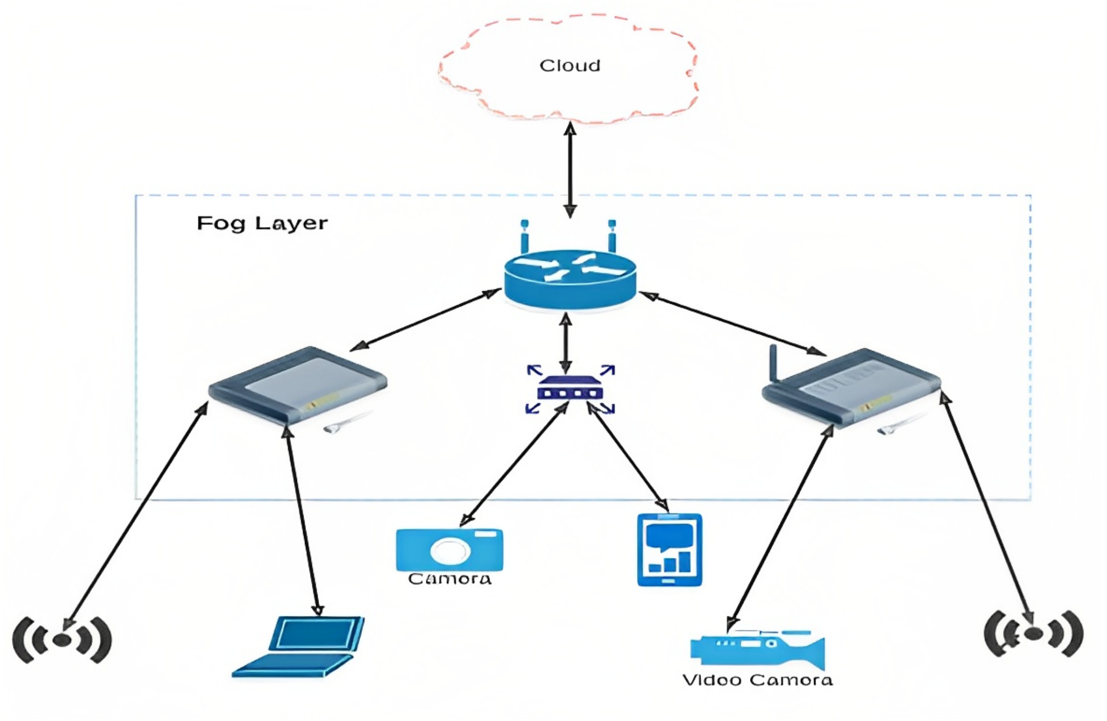

Figure 1.1: Fog Architecture

Fog computing provides the same computations, storage, and networking capabilities on a highly virtualized platform that sits between end devices and traditional Cloud servers. In the context of IoT, Fog computing is a compromise between distributed Cloud servers and centralized edge devices.

Fog computing delivers Cloud-based services closer to *IoT* devices by providing computation, networking, and storage services [7]. Fog computing's network capabilities allow Cloud services to be deployed across a large region. Fog computing will enable characteristics like situational cognizance, mobile support, instantaneous communications, adaptability, and integration [8]. Fog computing, in particular, makes it possible to perform computations close to the network's periphery [9]. Fog computing surpasses Cloud computing in meeting user expectations for *IoT* applications [10].

1.1 Limitations of Cloud Computing

The volume of Big data is increasing at an alarming rate because of how rapidly *IoT* technologies are being implemented. According to [11] there were around 30 billion linked devices all over the world by the year 2020 which may increased to almost 80 billion by the year 2025, almost three times as many in just five years. One prediction states that by the year 2025, around 152,000 brand-new devices will be connected to the internet each and every minute. International Data Corporation *IDC* has made a prediction that by the year 2020, around forty percent of data will be processed at the edge, and this prediction was published. It is difficult to send all of the data that is made to the cloud because of the costs of connectivity. Since this is the case, it is preferable to process the data locally, rather than transmitting it to the cloud, so that it may be more conveniently accessed by the users [11].

The *IoT* was developed to facilitate the opening of additional doors leading to opportunities that have the potential to contribute to a more desirable future by increasing profits and reducing expenses. This is accomplished by providing improved insights in situations in which massive amounts of data on their own are insufficient. In order for businesses to make the most of the *IoT's* revolution, they need to provide a platform that enables them to collect, manage, and analyze a large amount of data from sensors in an effective manner that is also cost-efficient [12].

1.1.1 Ineffective use of Network Resources

Because of its tremendous compute and processing resource capacity, the Cloud is the ideal location for processing large amounts of data. However, the volume of network traffic considerably grows if all of the created data is sent to the Cloud for processing. As a result, it is important to both mine data in order to detect the pattern at the edge level and reduce the volume of data that is present at the edge level. Certain applications, such as those used in video surveillance systems, produce a significant amount of data in the form of video.

There are two major problems that are caused by these types of systems. To begin, if upload all of the video data to the Cloud, it might potentially use up all of the bandwidth that is available before uploading them to the Cloud. Second, processing data at the edge of the network is difficult for the simple reason that it requires a significant amount of computing power. The Cloud is the ideal location for the processing of large amounts of data, as was indicated previously; however, the problem is that we do not have a limitless amount of bandwidth that is available to us. While video data are essential for the prevention of criminal activity and the establishment of an intelligent monitoring system in a smart city, the complexity of such systems will continue to grow over time. Therefore, dealing with applications that require to process video files via the Cloud is not the most comfortable approach to handle the situation [13].

1.1.2 Awareness of latency

When compared to other types of applications, those that utilize augmented reality, online gaming, smart homes, and smart traffic are especially vulnerable to the effects of latency. In the vast majority of situations, fog nodes are located one or two hops away from the user. As a consequence of this, the fog idea, when applied to the processing of Big data at the edge, is easily capable of supporting applications that are cognizant of their level of latency. The Cloud, on the other hand, is located at a multiple hop distance, which is quite a distance from the user and resulting in a higher latency than fog. This distance also makes the Cloud more difficult to access. In addition, latency has an impact on the enterprises that operate on the internet. For example, a delay of 100 milliseconds could result in a 1% loss in sales for Amazon, whereas a delay of 500 milliseconds would result in a 20% decrease in traffic for Google [14].

1.1.3 Location awareness

The vast majority of applications for the *IoT* are aware of their context, which indicates that, the way in which they process data is affected by the surrounding environment and the applications that are immediately adjacent to them. Sending all of these context-aware application requests to the Cloud is not only impractical but also not always possible. This is because bandwidth and delay limits make it impossible. There are many applications for the *IoT's* that require low processing delays of less than a specific amount of milliseconds.

Some examples of these applications include those used in the healthcare industry, automotive networks, applications for controlling drones, and emergency response systems. Processing of data in real time is essential to the functioning of these applications. These application services never use data from far-away places; rather, they always rely on the environmental data from their immediate surroundings. With the use of location awareness, which is also utilized in intelligent traffic applications, the pattern of the traffic, including things like roadwork, roadblocks, traffic congestion, and accidents can all be observed. The developers of these programmers promote the exchange of information between vehicles that are connected to one another in order to improve both the navigation of vehicles and the management of traffic [15].

An additional illustration of the intelligent surveillance application system, enables a law enforcement officer stationed at a local police station to view a video stream of potentially dangerous individuals in the vicinity of his assigned area, as well as to monitor their movements and take appropriate preventative measures. Processing and executing applications in the Cloud may not be as efficient as they may be under the parameters indicated above if the Cloud has high reaction times [16].

1.2 Challenges with Fog Computing

The Cloud is utilized for all phases of the data processing life cycle, including pre-processing, post-processing, and processing of extensive data, all of which are wholly reliant on the Cloud. On the other hand, when fog computing is used, the data streams are pre-processed locally through several spatially distributed mobile or stationary devices. It is possible to install these gadgets in a specific place permanently, or they can be moved around depending on the situation. After the data has been processed, it is sent to a cloud data center for any post-processing that may be necessary.

Pre-processing, transformation, and post-processing are the three stages that make up the life cycle of data processing in a fog environment. These phases are also called by their respective acronyms. The execution of applications data on fog poses several issues because fog devices can be mobile and have restricted resource capacities; as a result, many challenges need to be solved. Some of the key challenges that cover the key concerns raised regarding the execution of applications on fog are shown in Figure 1.2 and explained below: [17].

1.2.1 Selection of a Master Node

The possibility of many Fog computing devices being mobile may make it challenging to build a cluster of Fog computing devices. This is because mobile devices are frequently being moved, making it difficult for them to remain in one location. When working with huge volumes of data using the map-reduce processing paradigm, it is vital to select a master node, which is mainly responsible for keeping track of jobs. A master node prevents jobs from becoming lost or forgotten. Without a primary node, jobs may not be completed correctly. It is essential to rerun the whole operation if the master node is relocated to a different cluster. This is because it will have an impact on the way the job is managed. Selecting the master nodes for the Fog cluster can be a challenging endeavor. Choosing a master node from among the stationary nodes, on the other hand, is an effective method for solving this problem.

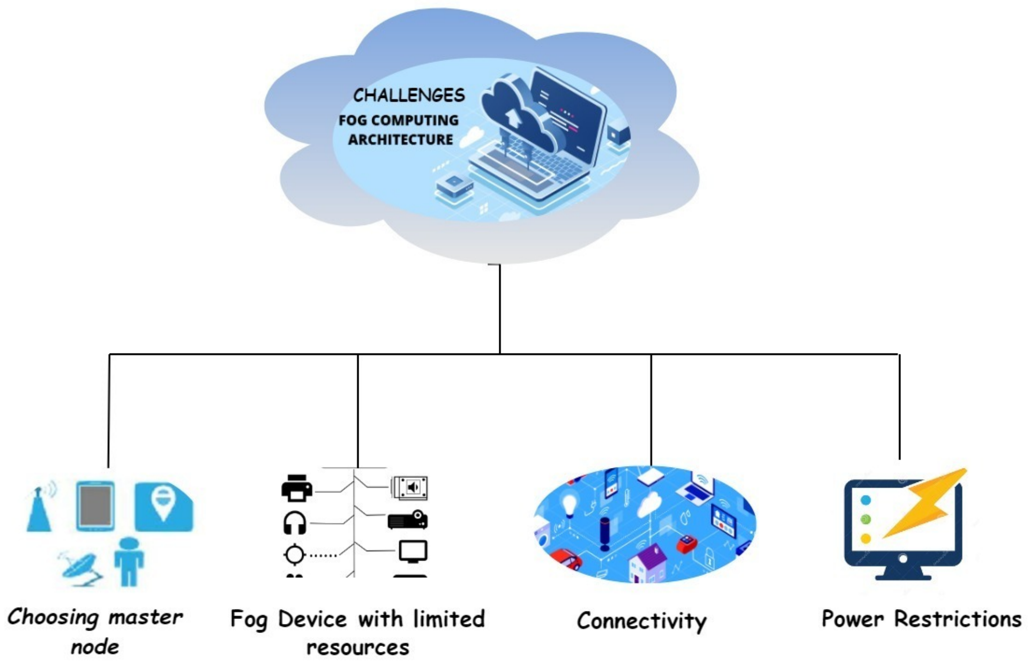

FIGURE 1.2: Challenges with Fog Computing

1.2.2 Limited Resources of Fog Devices

Computing devices that use fog have fewer resources available for processing data than traditional computing devices. Compared to those found in the Cloud, the resources that can be accessed through fog devices are extremely limited. Nevertheless, by assembling Fog clusters, data processing can be carried out on these devices. Any device that can perform computation can function as a fog device, including user devices and all network management devices. Fog devices can take on many different forms.

In most cases, these devices come equipped with their own operating systems and software programmer, which account for the majority of the resources utilized by the system. Running applications developed with fog is always the second priority on these devices. Therefore, an effective and well-thought-out policy for the distribution of jobs is required to safeguard a significant number of job failures.

1.2.3 Connectivity

Most devices in fog computing and the Internet of Things are connected via wireless communication, hampered by interference and fading. As a result of these problems, there are peaks and valleys in the amount of accessible bandwidth. Furthermore, there are many gadgets and sensors that are always communicating with one another in the fog's atmosphere.In addition, there will be a rise in the overall number of devices connected to the internet as time goes on.

To find a solution to this problem, it is important to come up with a good model that can estimate the number of connected devices in a fog and predict how many resources will be added before the system breaks down. After that, there will be a chance of finding a solution [18].

1.2.4 Command Restrictions

In the fog computing paradigm, a fog device can be any device that can process, store, and connect to a network. As a result, mobile devices powered by batteries, such as laptops, smartphones, and tablets, can function as fog devices. The device can turn off due to low battery levels while the computation is still in progress because the amount of energy these devices can provide is limited.

As a result, it will be necessary to reschedule any tasks being completed before the power goes out so that they can be carried out on another device. Consequently, the scheduler might know the total energy available to a mobile device before delegating a task to that device. Second, there is a wide variety of sensor devices, and the batteries that power these devices typically need to be recharged regularly. These batteries can become discharged over time if they are not appropriately maintained.

Because a malfunction caused by the battery may result in severe injury in some instances, this can be a complex problem to resolve. For example, body sensor networks are used to monitor patients. They need to employ energy-efficient methods by incorporating primary data analytics, which will reduce the amount of data that has to be stored and transferred, causing to save a significant amount of money.

1.3 Scheduling in Fog Computing

Scheduling refers to the process of efficiently allocating resources to application modules. It is up to fog schedulers to select the appropriate fog device or Cloud for the application. Reduced make-span, minimal monetary cost, increased throughput, decreased energy usage and low bandwidth usage, etc., are some of the primary goals of a fog scheduler. Poor resource allocation in a Fog computing environment may lead to slower application performance, more wait time, more costs, and missed Service Level Agreements *(SLAs)*.

1.4 Motivation

Cloud computing originated from the idea of distributed computing and has matured into an advanced computing infrastructure in recent years [4]. Users' tasks run in parallel on many virtual machines in the Cloud. With increased data expected to be kept in the Cloud, network disruptions and unavailability are exacerbated, making it impossible to store such data. Substantial amounts of digital data provide a security risk during storage. Maintaining low latency and allowing high mobility are only two of the upcoming issues that Cloud computing will have to overcome due to the explosion in data and the proliferation of connected devices. In response, the idea of Fog computing emerged to address these concerns. The term "fog" describes a distributed paradigm in which processing, storage, and networking resources are relocated to devices on the network's edge [19].

Since Fog computing enables processing closer to the data sources, it minimizes the latency and cost of transmitting data to a distant Cloud. The fog is a complementary technology to the Cloud that processes data at the network's edge. It consists of many nodes, or individual components, spread out over the globe. Fog's horizontal and hierarchical structure allows it to connect everything between the edge and the Cloud. Routers, switches, gateways and access points etc., are all examples of conventional network infrastructure under the umbrella term "fog network". As a result of fog computing, decision-making is decentralized, bandwidth is effectively utilized, operating costs are reduced, interoperability is achieved, and continuous operations can occur even when reliable network connectivity is difficult to establish.

The term "Fog computing" was coined in 2012 [19] by Cisco, the firm that originated the idea. The growth of electronic devices is to blame for the enormous amount of data generated. Processing the data generated by IoT devices is an everyday use case for Cloud computing environments. However, several challenges make uploading such large datasets to the Cloud arduous. Transferring this much information to the Cloud is wasteful and time-consuming for most applications, especially those extremely sensitive to latency. Consequently, assessing data at its source reduces latency, alleviates strain on the primary network, and keeps critical data secure by retaining it inside the network. Fog can process large amounts of data, providing strong backing for the IoT underlying architecture. Reduced latency, improved location awareness, and support for real-time applications are some ways fog be helpful.

1.5 Problem Statement

The Fog computing method shifts data processing, transmission, and storage to the edge of a network. The placement of application modules in fog environment is important to achieve the certain goals of reducing the execution time of applications with minimum Cloud resources cost and the efficient resource utilization. Scheduling is a NP-hard problem. In Fog computing, scheduling becomes harder due to the heterogeneous devices, user mobility, and varying application requirements. There are several issues related to fog scheduling such as latency-aware placement of application modules, high bandwidth usage, higher monetary cost and execution time of applications. To overcome the issues like high bandwidth usage, high execution time, high monetary cost, there is a need to develop a resource-aware scheduler for Fog computing environment which can schedule the incoming application modules in such a way to minimize the execution time of applications, reduce the monetary cost of application execution, and minimize bandwidth utilization by maximum utilization of fog devices.

1.6 Research Questions

1. How to reduce the execution time of applications with efficient resource utilization at fog layer?
2. How to develop a resource aware scheduler, so that the bandwidth and the monetary cost of applications should be minimized.

1.7 Research Methodology

This thesis uses the same research methodology as presented in [20]. We started our research with the goal of finding job placement solutions in heterogeneous environments. As part of this process, a literature review (Chapter 2) was conducted,

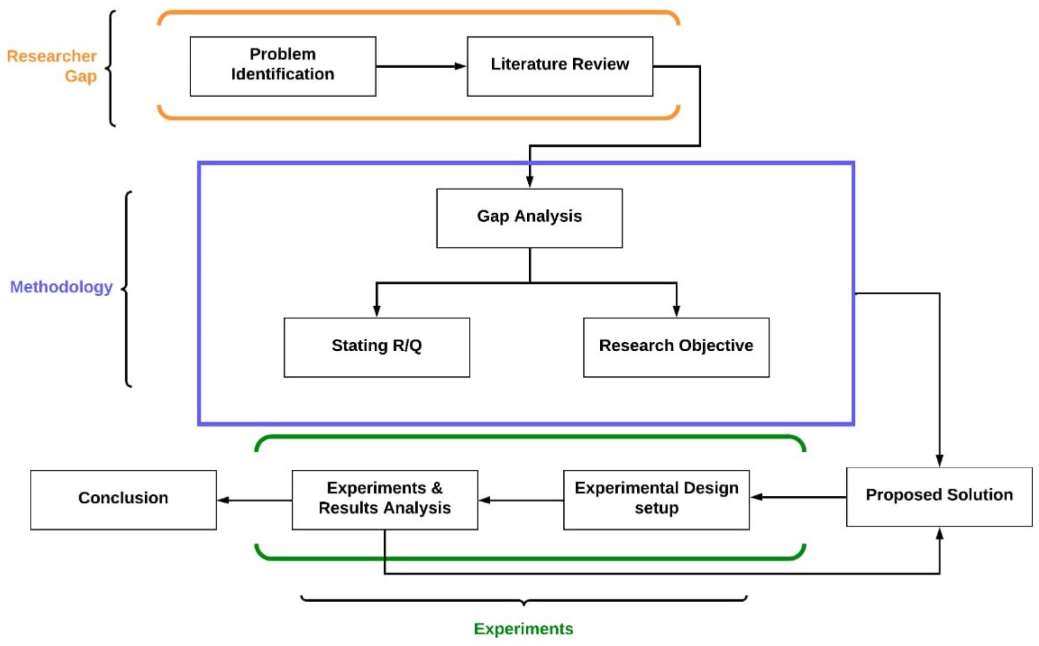

FIGURE 1.3: Research Methodology

including articles from top conferences and journals. From literature review it was observed that: In addition to bringing the benefits and power of Cloud computing closer to where data is created and acted upon, efficient scheduling at the fog layer reduces the amount of data that is sent to the Cloud, thus reducing bandwidth consumption and associated costs. Moreover, it also reduces latency and improves overall responsiveness because data processing takes place near the data. As a result of this observation, research questions were developed (Section 1.6). To answer these research questions comprehensive solutions were designed (Section 3.8, Section 4.5). An experimental plan was developed to test the validity of the proposed solution. At the end the experiments were performed to verify the viability of the proposed solution (Section 3.9, Section 4.6) in order to answer the research questions in Section 1.6. An overview of the adopted research methodology can be found in Figure 1.3.

1.8 Research Contribution

Following are the major contributions of this research work:

- In-depth critical analysis of the state of the art scheduling approaches, resource aware scheduling and heuristic based resource allocation approaches in a Cloud Fog computing environment to identify the strengths and limitations of these approaches.

- Empirical evaluation of the state-of-the-art heuristic-based tasks scheduling algorithms has been performed. Moreover, experimental evaluation of most prominent and state-of-the-art algorithms provides baseline for the resource aware cost efficient scheduler in Cloud Fog computing environment.

- In Cloud Fog computing environment, the application modules are relocated from the fog to the Cloud, our unique strategy help users save money while taking advantage of the speed and scalability of Cloud computing. We have introduced the priority based resource aware scheduling approach in Cloud fog computing environment. The incoming application modules are categorized on the basis of their computation and bandwidth requirement. The categorization of application modules helps us to make the prioritized list of the application modules, in which the incoming application modules are placed on the basis of the priority assigned by the scheduler. We have used the *iFogSim* simulator to test our approach under different network topologies of Cloud Fog computing environment and found better results as compared to other state of the art approaches [21].

- The increased in number of applications reduces the efficiency of the system in terms of the execution time, the band width and the monetary cost of application miodules. We have introduced the nature inspired approach *(GA-IRACE)* for job scheduling in Cloud Fog computing environment. The Genetic algorithm *(GA)* is used to iteratively improve the suggested solution based on how long it takes various applications to complete their tasks. The chromosome-based issue representation is first encoded in the proposed method by considering the application's attributes and the fog devices. The solutions evolved by changing various parameters in *GA* by employing the re-population operators.

GA is implemented to schedule the application components, and the results have been validated using the *iFogSim* simulator. A set of three unique application corpora and state-of-the-art techniques are used to evaluate *GA-IRACE*'s performance. The findings indicate that the suggested *GA*-based approach reduces execution time, cost, and bandwidth utilization by 15-40%. Additionally, compared to fundamental algorithms, the enhanced scheduling makes less use of Cloud resources, is faster, and requires less bandwidth [22].

1.9 Research Evaluation

The significance of the proposed approach is evaluated by using the *iFogSim* simulator. The *iFogSim* is open-source software that models and analyses the performance of Cloud fog settings and services. *iFogSim* architecture [23] is frequently used to represent many types of computing paradigms. Since the proposed method has few practical uses in the actual world, it would not be easy to test it in a real-world fog environment. In addition, research and development in a simulated environment is currently favored since this is a developing yet novel industry.

Different versions of this scenario have been built using three different network topologies, each with specific workloads. The term "workload" describes all the application modules deployed in Cloud fog architecture. The millions of instructions*(MI)* values for the application modules vary between 50 and 700. For experiments, three distinct datasets are used, one for each kind of power-hungry program.

We use simulation to examine the *RACE* algorithm's efficiency across three scenarios, looking at how it fares in terms of cost, usage of bandwidth, and execution time. The simulation parameters are the total number of modules, millions of instructions *(MI)* for each module, the total number of fog devices, and the CPU *MIPS* of fog devices and the Cloud.

For the *GA-IRACE* we also used the simulation using *iFogSim* simulator to evaluate the performance of optimized placement of application modules at fog layer. The simulation is tested under three different scenarios of using the different network topologies at the fog layer.

1.10 Thesis Organization

The thesis is divided into five further chapters. Chapter 2 dives into the specifics of Fog computing and resource allocation by examining the current state-of-the-art resource-aware scheduling for fog environment algorithms. Moreover the chapter also covers the kinds of scheduling algorithms, the goals of scheduling, the benefits and drawbacks of cutting-edge task scheduling algorithms. Chapter 2 also includes an overview and discussion of research gaps.

In Chapter 3 the discussion is about the development of a resource aware scheduler for the Cloud Fog computing environment The applications are placed in Cloud Fog Computing environmnent by considering the processing requirement of the application modules and the processing capabilities of the fog devices at fog layer. Meanwhile the bandwidth and the monetary cost of the Cloud resources was also under consideration.

Chapter 4 is about the development of improved nature inspired approach *GA-IRACE*. This chapter discusses the use of genetic algorithm for the optimized placement of application modules from fog to the Cloud.

The conclusion of the thesis is discussed in Chapter 5. In addition, a number of research directions are provided for researchers to explore in future research.

Chapter 2

Background and Related Work

This chapter provides the background information required to familiarize the reader with the study conducted in this thesis. Moreover, the prior research work related to job scheduling in Cloud Fog computing environment is also presented in this chapter.

2.1 Background

The fog computing system is a distributed computing system in which information, processing, storage, and applications are distributed between the data source and the cloud. [24]. Because of their similarity, Fog computing is frequently used interchangeably with Edge computing. This is used by a variety of enterprises to increase production, as well as to assure security and fulfill legal obligations [25]. While Cloud computing is suitable for resource-intensive, long-term analytic, fog networking is a valuable complement since it enables shorter-term analytic to be conducted at the edge. Even while edge devices and sensors are the sources of data production and gathering, they may not necessarily have the resources necessary to perform advanced analytics. In most cases, cloud servers are physically located too far away from the end user to provide an accurate analysis and prompt response to data in real-time [26].

2.1.1 Edge Computing

The main components of an Edge computing system are the end devices, the edge devices, servers, and so on. These parts can be geared up with the chops needed to enable computing at the edge. Since it is a decentralized kind of computing that operates at the network's edge, Edge computing can respond to user requests for computational services much more quickly. The Edge computing system usually refuses to provide the raw data in large quantities to the central network. On the other hand, Edge computing focuses on the end devices rather than the Cloud or Cloud-based services [27]. There have been several paradigms established in the area of Edge and Cloud computing. One possible future development is Mobile Edge Computing *(MEC)* followed by Fog Computing *(FC)* and Mobile Cloud Computing *(MCC)*.

2.1.2 Mobile Edge Computing *(MEC)*

It is widely agreed that Mobile Edge Computing *(MEC)* is a necessary component of the current generation of cellular base stations. It allows for the co-location of cellular base stations and edge servers [28]. To improve network performance, *MEC* strives to provide clients with adaptive, instantaneous access to cellular services. There have been recent advancements in *MEC* that make possible for it to enable 5G communications. In addition, it seeks adaptable entry to radio network data for content distribution and application creation [29].

2.1.3 Mobile Cloud Computing *(MCC)*

In comparison to running these kinds of high-computation applications on the mobile devices themselves, it is far more practical to use Mobile Cloud Computing *(MCC)*. The processing resources needed for offloading and local execution of mobile applications near their end users are made available by *MCC* [30]. The edge network in *MCC* often houses lightweight Cloud servers known as cloudlets [31].

Cloudlets provide a three-tiered hierarchical application deployment architecture for mobile applications, which includes end-user mobile devices and Cloud data centers. Briefly expressed, *MCC* is an approach to improve mobile customers Quality of Experience *(QoE)* by integrating Cloud computing, mobile computing, and wireless connectivity. *MCC* opens up new markets for both traditional network service providers and Cloud providers.

2.1.4 Fog Computing*(FC)*

In the same way, as *MEC* and *MCC* permit computation at the edge, Fog computing can do the same. It is important to note, though, that fog computing is not restricted to edge networks, but can also be implemented within the core network as well [8]. Fog computing relies on both central and edge networking components to perform its computations.

2.2 Architecture of Fog Computing

Fog architecture refers to the practice of computing, storing, and processing data by utilizing the services offered by end devices such as switches, routers, and multiplexers. The software, hardware, and logical components of the network, in addition to the physical features of the network, that collectively make up the fog computing architecture join together to create a massive network that is made up of a large number of devices that are all connected. After that, several operations can be carried out by utilizing this network.

The most important architectural features are the physical and geographical distribution of fog nodes, the fog architecture's protocols, and the topology of the fog architecture. Distributing functions across many different layers, the types and quantities of protocols used, and the restrictions imposed at each layer is called "fog architecture." This term refers to all of these aspects of the practice. Furthermore, it relates to the practice as a whole [32].

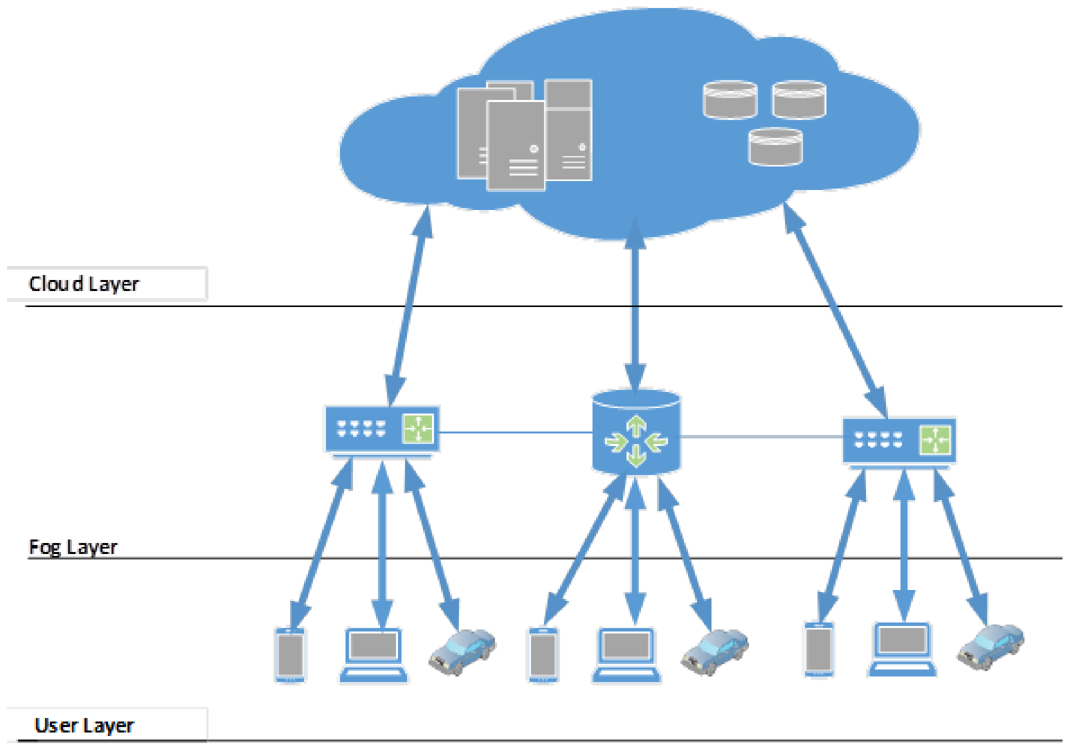

FIGURE 2.1: Fog Computing Environment

In the past few years, several studies proposed different architectures for Fog computing [6, 33–40]. According to these studies, all proposed Fog computing architectures characterize the concept of data processing at the edge and the usage of fog devices for temporary storage. In contrast, Cloud infrastructure will be used for long-term storage. Since Fog computing is much less centralized than traditional forms of computing, Bonomi et al. [6] first proposed architecture for Fog computing consisted of three layers.

The middle layer created as a result of Fog computing and is positioned between the end devices and the Cloud is referred to as the "fog layer,." The term "fog layer" is used to characterize this layer. The fog layer delivers Cloud services to the network's periphery, where users can access them [41]. Figure 2.1 illustrates the three tier hierarchical architecture of Fog computing environment and is explained as follows:

2.2.1 Terminal/User Layer of Fog Computing Architecture

- The terminal layer, also called the user layer in the hierarchical architecture of Fog computing, is the foundation of fog architecture and comprises various endpoints such as smartphones, sensors, intelligent cars, readers, and smart cards.

- The sensing and recording apparatus are located within this layer. The devices can be found dispersed across a large area, frequently at a significant distance from one another.

- The sensing and information collection functions are the primary focuses of this layer. This layer covers a wide range of devices, including those built on different platforms and with different architectural designs.

- Devices can function in environments that contain a variety of different types of technology and communications.

2.2.2 Fog Layer of Fog Computing Architecture

- Routers, gateways, and other kinds of fog servers are some examples of the kinds of devices that can be discovered at the fog layer. In addition, the fog layer is sometimes referred to as the fog nodes as a collective word. Other devices, such as access points, base stations, and specific fog servers, are some devices that could potentially be located in the fog layer.

- Fog nodes are distinguishable from other nodes in a network due to their location on the network's periphery and their name. In certain situations, an edge could be situated one hop away from an end device.

- The fog nodes are located right in the midst of the network, between the cloud data centers and the end devices.

- Fog nodes can either be permanently installed, like the ones that are found in a bus terminal or a coffee shop, or they can be mobile, like the ones that are installed inside a moving vehicle. The provisioning of services to end devices is the responsibility of fog nodes. The fog nodes can temporarily compute, transfer, and store the data.

- The provisioning of services to the end devices is the responsibility of fog nodes. The data can be temporarily computed, transferred, and stored by the fog nodes.

- IP core networks make it possible to connect fog nodes to Cloud data centers, paving the way for interaction and collaboration with the Cloud to enhance processing and storage capacities. This is done to improve the capabilities of both processing and storage.

2.2.3 Cloud Layer of Fog Computing Architecture

- In this layer, computation and analysis are carried out, and data are persisted for subsequent user access and backup purposes.

- Cloud tier is made up of a variety of storage devices, including powerful computers, that can provide a significant amount of space for storing information (servers)

- For processing power and storage capacity, this layer is unsurpassed.

- Cloud computing is carried out across a distributed network of enormous data centers. With exceptional processing power levels, users are provided with all of the benefits that contribute to the overall attractiveness of Cloud computing by data centers. The data centers offer scalability and on-demand access to the available computing resources.

- Clouds, which comprise the highest layer of the fog structure, can be found somewhere in the atmosphere. In fog architecture, it is used to permanently store data and acts as a backup at the same time. Cloud storage is typically utilized for user data that the user does not currently require.

Aside from this, several researchers have proposed other architectures, such as hierarchical, layered, and network-based [6]. Aazam et al. [33] proposed a layered architecture comprised of the three-layers, as shown in Figure 2.1. Another architecture was proposed by Atlam et al. and H. & V et al. [42, 43] comprises of six layers: the physical and virtualization layer, the monitoring layer, the pre-processing layer, the temporary storage layer, the security layer, and the transport layer. These six layers are shown in the Figure 2.2 and are explained below:

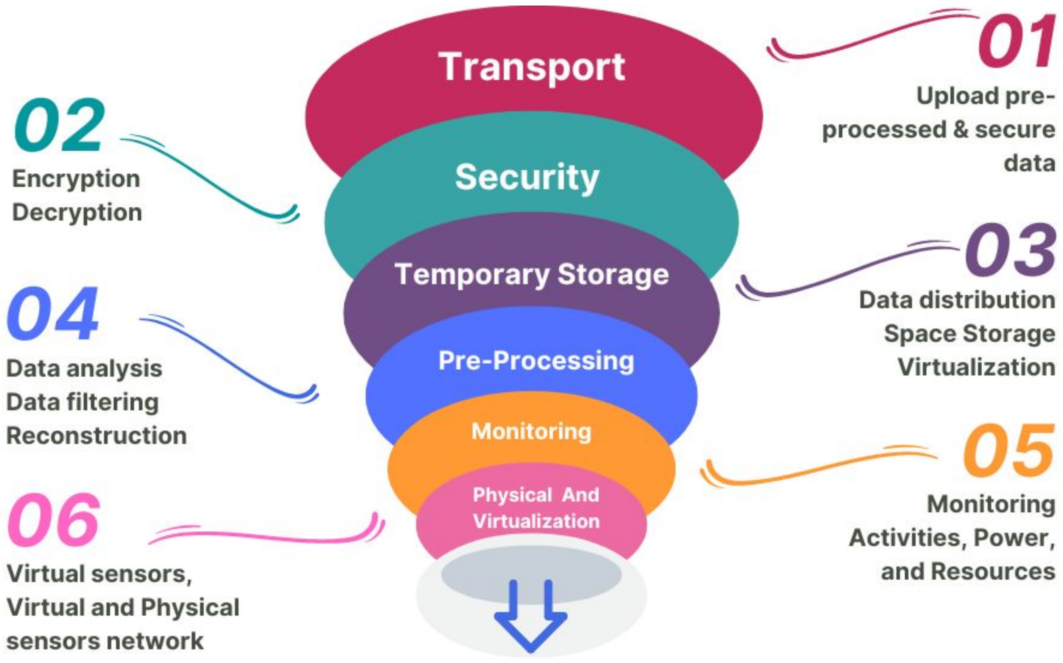
Transport
Security
Temporary Storage
Pre-Processing
Monitoring
Physical And Virtualization
01
Upload pre-processed & secure data
02
Encryption Decryption
03
Data distribution Space Storage Virtualization
04
Data analysis Data filtering Reconstruction
05
Monitoring Activities, Power, and Resources
06
Virtual sensors, Virtual and Physical sensors network

FIGURE 2.2: The Layered Architecture of the Fog Environment

2.2.4 Virtualization Layer and Physical Layer

This layer is made up of physical and virtual nodes. The primary function of the nodes, which are situated in various locations, is data collection. Nodes typically use sensing technology to compile information regarding the environment in which they are located. A node can refer to a device that runs on its own, such as a mobile phone, or it can refer to a component of a larger device, such as a temperature sensor that is mounted inside of a vehicle.

Regardless of whether a node is standalone or a part of a larger piece of hardware, data is gathered from the environment around this node by the sensors employed here, which are then sent to the upper layers via gateways to be processed further [44].

2.2.5 Monitoring or Observation Layer

The monitoring or observation layer monitors the nodes connected with various activities and responsibilities. Several metrics may be monitored, such as the amount of time that nodes work, the temperature, the maximum battery life of the device, and any other physical properties they possess. In the other metrics, this layer specifies when and which node must perform the job. It also takes care of the requirements for any additional tasks that must be carried out to accomplish the primary task [45].

2.2.6 Pre-processing Layer

The pre-processing layer is responsible for a wide variety of data operations, most of which are related to analysis. After the data has been passed through the pre-processing layer, it has been cleaned, and any remaining undesired data may be found using a search. The process of gleaning valuable and pertinent information from the copious amounts of data gathered by the end devices is an essential component of the data analysis carried out at this layer. When using data for a given purpose, one of the most vital considerations is whether it has been previously evaluated [46].

2.2.7 Temporary Storage

The temporary storage layer is responsible for the distribution and replication of data in a manner that is not permanent. This layer makes use of storage virtualization technologies such as virtual storage area network *(VSAN)*. After the data have been transferred to the Cloud from this layer, the temporary layer will be emptied of its contents, and the data will be deleted [46].

2.2.8 Security Layer

The security layer is in charge of ensuring the confidentiality of the data, maintaining the data's integrity, and encrypting and decrypting the data, in addition to its other responsibilities. When working with fog computing data, one's identity can be protected in several ways. Among these ways is the protection of an individual's privacy about the use of the system, an individual's data, and an individual's location. Even after being sent to the fog nodes, the data's confidentiality will be maintained with a security layer. The security layer prevents unauthorized parties from gaining access to the information and ensures that it cannot be viewed by anyone else. The security layer will maintain the data's confidentiality even after the data has been sent to the fog nodes [47].

2.2.9 Transport Layer

The fundamental objective of the transport layer is to guarantee the secure transmission of data that has been fine-grained and partially processed to the cloud layer, where it can subsequently be kept forever. This layer also fine-tunes the data before it is transferred. It has been determined that collecting and uploading the relevant portion of the data will help improve operational efficiency. Before being uploaded onto the Cloud, the data travels through intelligent gateways. Because fog computing has limited resources, the communication protocols employed have been intended to be as lightweight and efficient as possible [47].

2.3 The Products for Fog Computing

The term "Fog computing" refers to a technique that moves some of the operations of a data center to the outside of the network. The fog provides a small amount of computing, storage, and networking services decentralized between the end devices and the traditional cloud computing data centers. The primary goal of Fog computing is to provide low and consistent latency for time-sensitive *IoT* applications. Some of the products at the fog later are as follows:

2.3.1 Cisco IOx

Integrating sensors connected to the Internet of Things and the Cloud, the Cisco IOx application platform makes it possible to carry out processing in the fog close to the end devices. The *IOS* and Linux operating systems, fog director, the Software Development Kit *(SDK)* and development tools, and fog applications are the four parts that come together to form Cisco IOx. Linux is an open-source network operating system *(NOS)* that can be modified.

In contrast, Cisco IOS is the most popular network operating system *(NOS)* that provides safe network connectivity. This platform was created specifically for running applications on the infrastructure of the Cisco *IoT* network. With fog director's assistance, the applications being run on the Cisco IOx environment can simplify their administration for greater efficiency. The *SDK* and the development tools supply the developers with a selection of tools, a command line utility, and web-based applications. Applications considered to be in the "fog" category can be deployed relatively quickly and run on infrastructure fueled by Cisco IOx [48].

2.3.2 Local Grid-Based Platform for Fog Computing

The platform for fog computing is provided by the local grid, an embedded software package that can be installed on any peripheral devices connected to the network. The local grid can transform various communication protocols into an open standard computing platform designed for use in fog. Every sensor, switch, router, machine, and edge device that needs secure access can have the embedded software installed on them, and it will run on the infrastructure IOx enables [49].

In addition to this, it facilitates peer-to-peer *(P2P)* communication across numerous edge devices, which enables real-time coordination and control even in the absence of a central server. The acronym *P2P* often refers to the communication of this kind. Through the incorporation of intelligence network communication, the local grid also features a communication infrastructure connected to the Cloud. This adds to a reduction in the amount of latency that occurs, a reduction in the amount of bandwidth consumed, and an increase in the level of security achieved. With the support of this platform, distributed data processing and decision making may be carried out locally on edge devices [49].

2.3.3 Gateways and Fog-Based Equipment

Regarding fog devices and gateways, also referred to as computer on modules, in addition to the proprietary products currently on the market, various other options can be used instead. Alternatively, these choices may be utilized in place of the products. Devices such as the *Intel Edison, Arduino, Odroid-XU4 Raspberry* P_i*, and Asus Tinker Board* fall under this category. Most people have adopted the Raspberry P_i group [50]. On the other hand, there is a possibility to have vendors who produce devices that are more powerful than the Raspberry P_i at a price comparable to what Raspberry P_i charges for its products.

Most of these devices are compatible with the Android operating system and can run on other operating systems such as windows, RISC operating system, Ubuntu MATE, Caspian, and OpenELEC. The Android operating system is compatible with the vast majority of these devices. Throughout the past few years, a significant number of research studies utilizing these computers on various modules have been carried out to implement the fog environment for processing applications data [51]

2.4 Fog Computing Applications

For Cloud computing, data is stored and processed in off-site data centers. Because of the dispersed nature of the Fog computing architecture, smart devices on the network's edge exert some control over the applications' operations, despite the fact that most Cloud-based software is managed automatically. Fog computing is a dispersed architecture; part of the application's operations will be controlled by the smart devices located at the network's edge, however, other applications continue to be administered remotely.

A simple explanation would be to use smart devices on the network's edge to do all the tasks that would otherwise be performed in the Cloud. For the most effective processing of data, fog serves are there as an intermediate layer between the Cloud and hardware. Therefore, there will be a reduction in the amount of data that must be sent to the Cloud. In addition to improving Cloud security, Fog computing also serves as a firewall. A wide range of applications as shown in Figure 2.3 might be improved by using the Fog computing paradigm. Some of the most important applications are listed below.

- **Real-Time Video Analytics:**
  For the smart city to provide traffic management and associated services, widespread deployment of cameras in the town or along the road is necessary. Tasks like item tracking/identification, etc., that need a lot of storage space and processing power are ideal for Fog computing. The next phase directly delivers alerts, events, video descriptions, or video summaries to users, databases, or central servers. It is possible for fog to process massive volumes of video streams and low-bandwidth output data in real time [52]. Fog servers can be equipped with privacy-preserving features to reduce the risk of a government-run surveillance program leaking data into the public domain.

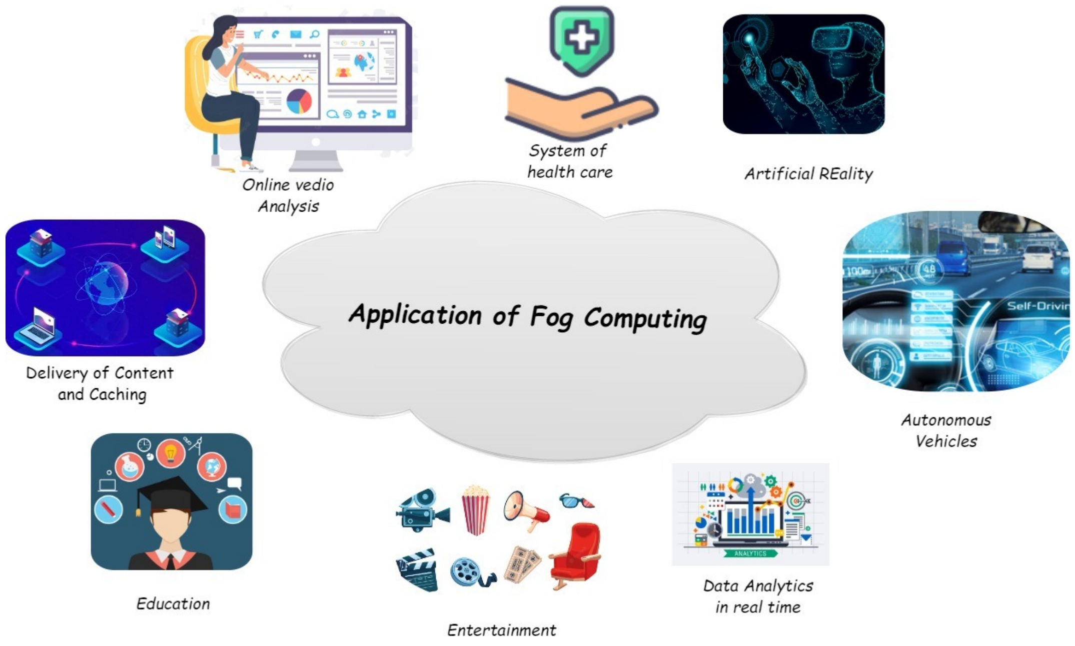

FIGURE 2.3: Applications of Fog Computing

- **Health Care System:**

  As *IoT* devices are becoming pervasive, they can be used to track various healthcare processes. Ubiquitous computing's architectural components are beginning to include Fog computing. Due to increased network latency, Cloud-based e-healthcare solutions are not suitable for use during medical emergencies.

  Many healthcare computer tasks are handled by neighboring fog nodes, reducing wait times and increasing accessibility, thanks to Fog computing. Some applications in smart health care must be completed immediately, while others may tolerate some wait. Healthcare providers, for example, record and store patient data so they can analyze it later. This method of storing and retrieving data is not time-sensitive and can tolerate some delay. However, quicker data processing is needed to produce emergency alarms in other situations, such as when a patient's life is in danger. Due to the potentially fatal consequences of a delayed reaction to an emergency warning, such tasks are more latency important [53].

- **Augmented Reality *(AR):***

  Augmented reality smartphones, tablets, and smart glasses are popular platforms for applications that add contextual information to the user's perspective of the environment. Microsoft hololens, Google glass and Sony smart eyeglass are just three examples of primary products and initiatives from recent years. High computer power to analyze video streaming and high bandwidth for data transfer are typical requirements for augmented reality applications.

  Ordinary augmented reality applications, for instance, must analyze video frames in real-time using a computer vision algorithm, in addition to handling other inputs like speech and sensors, before finally displaying relevant data to users. Delays in a sequence of successive exchanges, however, have a profound effect on humans. Users who endure processing delays of several milliseconds or more get frustrated and leave critical comments. Wearable cognitive aid based on Google glass and cloudlet, powered by an augmented reality *(AR)* system enabled by Fog computing, provides the user with tips on engaging with others via real-time scene analysis. Offloading computation-intensive tasks to a neighboring cloudlet allows the system to meet stringent end-to-end latency constraints. Services are automatically degraded in the event of a network outage or when remote cloudlets are unavailable [54].

- **Self-Driving Vehicles:**

  By exchanging valuable information, vehicular-ad-hoc networks *(VANETs)* ensure traffic efficiency, driving safety, and convenience in Intelligent Transportation Systems *(ITS)* [55, 56]. *VANETs* and their related applications such as self-driving have developed dramatically in capability and data transfer over the past few years. Thus, higher data storage, computation, and communication levels are required. As a result of these requirements, a new computing paradigm i.e., Vehicular Fog Computing *(VFC)* has been proposed. Vehicular Fog Computing *(VFC)*, which combines mobility, location awareness, and low latency to satisfy the demands of the application [55, 57, 58].

As part of *VFC*, end-user clients or near-user edge devices are collaboratively used to carry out substantial amounts of computation and communication [58]. *VFC* differs from existing techniques in that it has dense geographical distribution, supports mobility, and is close to end users [58], in addition to traditional Cloud characteristics, such as computing, storage, and applications.

- **Content Delivery and Caching:**
  Once server-side optimizations have been made, traditional online content delivery systems will no longer be able to respond to changes in user demand. To improve web performance, however, we need specific data only available at or near the client's network. When thinking about web optimization [59], considers this novel angle, the setting of Fog computing. The fog server provides dynamic, adaptable optimization based on information gathered from clients' devices and the surrounding network. Also, the fog server's proximity to the client allows it to get insight into the client's system and user history, allowing for more refined web page rendering. Additional bandwidth can be saved, and delivery delay can be decreased by optimizing caching inside fog nodes.

- **Real-Time Data Analytics:**
  Data can be moved from its point of creation to other locations via Fog computing infrastructure, enabling real-time analytics. Manufacturing systems' data is sent to financial institutions' real-time data systems through Fog computing for real-time analytics. Most effectively exemplifying grid computing is the modern electric power grid. The electrical network of today is intelligent and adaptive, requiring less manufacturing and electricity, yet still being responsive. With Fog computing, data doesn't need to be sent to a central place in order to be processed, making it especially useful in situations when data is being created at a distance. For the sake of not overwhelming the Cloud, some of the data that may be produced from a single sensor or a collection of sensors can be handled locally. Some of the most prevalent examples of this are electric meters [60].

- **Education:**
  Since the advent of Covid-19, educational institutions have been forced to adapt to the changing technological landscape. The whole industry evolved to rely significantly on computers, and many workers eager to further their careers turned to distance learning. The Fog computing platform simplifies collaboration and ensures the continued operation of data storage and management networks. It enhances scalability, elasticity, and redundancy in educational systems to keep data private, flexible, and secure. Maintaining records in the education sector is vital since information in the systems may be both organized and informal. Fog/Edge computing helps with data collection, processing, and application enhancement. As Fog computing becomes more widely used, it is increasingly being included in the academic system. This allows for technical progress in a very ancient field, enhancing data storage and instruction quality [61].

- **Entertainment:**
  The entertainment industry has made tremendous progress during the last several decades. Producer and consumer interest have been quite strong. Just consider the vast sports broadcasting industry, which includes networks such as *NBC* sports and *ESPN* news channels etc., which regularly broadcast coverage of events held in large arenas. Many fans desire minute-by-minute updates on their favorite sports. Captured footage is stored in the Cloud, where video processing services and algorithms process it and send it through streaming to users' devices. The enhanced gaming application results from Fog computing's decreased latency, increased bandwidth utilization, and protected user privacy[62].

2.5 Aspects of Fog Computing

The Fog computing process, transmission, and storage capabilities are provided by equipment that is geographically near the end user. Being in such proximity to consumers is great for honing service abilities and is also its main advantage over more traditional computer designs, including the following benefits and features.

- **Real-Time Communication with Minimal Delay:**
  In a local area network, fog nodes near the network's periphery collect data from sensors and other devices, then analyze and store that information. It provides fast, high-quality localized services powered by endpoints while significantly reducing data flow over the internet. It demonstrates that Fog computing is ideal for time- or latency-sensitive applications because it allows instant communication [58]. Fog computing's service latency is much lower than Cloud computing's, as proved by [5], which theoretically modeled the architecture of fog. After applying fog to the face identification and resolution problem, the experimental findings of [63] reveal that it took less time than Cloud computing.

- **Facilitation of Mobility:**
  Given the wide variety of endpoints commonly used in fog situations, the terminal layer is frequently in motion while the other terminals, such as traffic cameras, do not move. Both mobile and stationary data processing equipment may function as fog nodes in the fog layer. Having no restrictions on where it may be used, it can be used in the airport, coffee shops, trains, buses, etc., [64, 65]. The development of smartphones and other devices gives the user the option of whether or not to upload their information to a central server, as Fog computing is the direct communication of mobile and other devices for storage and processing. Mobile data analysis occurs for the massive data volume need to be processed by either the final device or an intermediary device. Inherent routing, communication, and addressing mechanisms allow fog applications to interact directly with people and mobile devices. Locator/ID separation protocol for mobile nodes may simplify mobility techniques by separating host and location identifiers [66, 67].

- **Data Dissemination and Decentralization:**
  In contrast to their cloud-based counterparts, services and applications for

Fog computing are best suited for decentralized network deployment. The system needs a lot of well-placed nodes to watch the endpoints and figure out where they are, which is necessary if it is to be mobile. Due to the decentralized nature of Fog computing, data is processed closer to the edge than at a centralized location. This feature could improve location-based services, real-time decisions, and large amounts of information processing. All pervasive devices can only reach their full potential if they are part of a thriving *IoT* and ubiquitous computing ecosystem. Fog computing applications in the Internet of Vehicles *(IoV)*, cars and vehicular access points can connect and work together to provide a wide range of services [55].

- **Data Protection:**
  Because Fog computing works in close proximity to its users, it allows privacy and protecting personal space to be maintained. The first line of defense might be to encrypt and separate data used by fog nodes as measures like isolation, integrity check access control policy, and encryption to keep private information safe. Second, Fog computing may prevent problems from occurring while the software is being updated. The update of firmware on a remote device may lose contact with the server; a common issue could be resolved with fog as for Fog computing systems, only algorithms and micro-applications are updated rather than firmware [68].

- **Energy-Efficient:**
  The fog nodes of a Fog computing architecture can be located anywhere. Due to the low concentration, there is no need for an extra cooling system. Also, it has been shown that this connection can use less energy if the communication range is shorter and mobile nodes have better energy management policies [69]. Fog computing is a much better choice when it comes to the environment. Researchers in [5] used modeling to show that Fog computing is a more environmentally friendly platform with the usage of almost 48% less energy than traditional Cloud computing.

2.6 Resource Allocation

Due to the availability of different Quality of Service *QoS* criteria, including CPU, memory, speed, and stability, resource allocation [25] in Fog computing differs from the conventional and distributed computing environment. In resource management, several requests may be stacked up in a queue. The Cloud's biggest problems are the time it takes to transfer data from end user to the Cloud for required operations and then send back to the end user causing delays, bandwidth utilization [26]. These problems are addressed by Fog computing's deployment of incoming application modules to devices near the network's edge with a load-balancing mechanism focused from edge of the network to the Cloud.

2.7 Resource Aware Scheduling for Fog Environment

Users of fog services may be in several places at once, making mobility a difficult obstacle to overcome. Fog computing's user mobility makes resource allocation a challenging task. It is possible for a particular device to become overloaded when users with different types of applications, such as latency-tolerant and latency sensitive(cpu, storage, and bandwidth), are in range of it at the same time. Luiz F. et al. [70] presented three methods for the mobility problem: First Come First Serve *(FCFS)*, delay-priority and concurrent.

When a considerable number of application modules are scheduled concurrently, system resources will inevitably become overburdened in a concurrent system. By holding off on downloading Cloud-based application modules until the client device has adequate processing power, the *FCFS* approach avoids resource congestion as latency-sensitive or latency-tolerant. While the suggested method effectively prevents resource contention across fog devices, it comes at the cost of increased network latency for time-critical programs.

When determining which devices should host which application modules, the delay priority technique presented in [70] gives preference to devices closest to the user and those with the least amount of processing power. The unused portion of a fog device's processing power is uploaded to the Cloud when it reaches its limit. As the number of application modules grows, the suggested technique helps minimize latency for latency-sensitive applications, but at the cost of increasing network usage and financial demands on Cloud resources.

A resource-aware scheduling strategy was designed by [24] to help increase productivity in a Cloud Fog Computing environment. The resource-aware scheduling technique considers the device's CPU, RAM, and network throughput when determining which application modules to execute on fog devices. Prioritization is determined by the CPU requirements and capabilities of newly incoming application modules or devices i.e., fog or Cloud. Recursive binary search is used to narrow down the pool of potential fog devices until one is found that meets the module's requirements for processing power, memory, and data transfer rate. The proposed fix is wasteful and fails to consider the cost of using the Cloud services.

The authors calculated the cost and time it will take to implement their solution on fog or cloud based on Estimated Start Time *(EST)* and Estimated Finish Time *(EFT)*, as well as the mapping cost of each application module. Modules are distributed among nodes and are determined by calculating the best compromise between cost and makespan (Fog device or Cloud). With the proposed approach, programs run slower while using less resources. For Big data processing framework V. Cardellini et al. [26] suggested a distributed Quality of Service *QoS* aware scheduling algorithm. Apache storm was used to perform the research on a cluster of eight worker nodes. The authors have updated the Storm distributed scheduler to increase its awareness of quality-of-service requirements. Experiments indicate that the suggested scheduler improves application execution performance compared to Storm's default scheduler.

L. Ni, J. Zhang, et al. [71] created a pricing synchronized petri net-based resource management technique for Fog computing. The basic idea is that the recipient may choose the ones that will offer them the most excellent satisfaction among various options. The proposed method considers the cost and time required to complete application modules. A user with a large credit limit may use secure tools to help them finish their work. Credit evaluations for users and resources have some wiggle space under this system. In addition, the proposed plan does not account for how those resources would be used. In [72], D. Rahbari et al. propose employing symbiotic organism search *(SOS)* to enhance the efficiency of the knapsack scheduling method. CPU utilization and virtual machine throughput are considered when determining the fitness function. The author used the *iFogsim* simulator to gauge the improvement in energy usage and network utilization. The proposed algorithm was compared to the baseline *FCFS* algorithm, the knapsack algorithm, and the knapsack *SOS* technique. However, the proposed approach is careless since it does not consider the resource utilization.

Latency-aware application module management using fog layer clusters was suggested by R. Mahmud et al. in [14]. Two developed algorithms support the proposed technique for application management. The first method correlates delays across an application's components to arrive at an overall delay. Those parts of the application that can function with a more significant latency are kept higher in the stack, while the remaining parts are handled by the fog layer. Regarding service punctuality, the suggested method satisfies the standards of *QOS*. This method does not include the cost of using Cloud resources with application modules that can better handle service delays.

A strategy for optimizing the fog layer's effects was presented by R. Mahmud et al. in [14]. The suggested technique uses a heuristic in which modules are transferred from a vacant fog node to an occupied one. In this manner, energy can be saved, and resources can be better utilized when fog nodes are fully utilized. However, communication delays may occur after a module migration because data will first reach the source node and then be relayed to the host node.

In order to find a middle ground between application up-time and Cloud resource costs, X. Pham et al. [73] suggested an approach. The authors save cost by using

Cloud resources and shorten the time needed to build interdependent modules. In mapping procedure the over all CPU is estimated in terms of time, money, and effort. To minimize both costs and delays, jobs are distributed among available nodes. The technique leads to inequitable load distribution and disregards resource efficiency. S. Agarwal et al. [74] used virtualization technology to provide a framework and technique for efficient resource allocation in a Fog computing environment. At the fog layer, the suggested architecture places fog servers *(FS)* and a fog service manager *(FSM)*. Every customer makes a separate request to *FS*. If there is enough processing power, *FSM* will take care of it by partitioning the module into smaller pieces and distributing them across the available *FS*. The modules will be delayed if the fog layer processing resources are utilized. The main disadvantage of this approach is the lengthened time to completion whenever modules are required to wait for the availability of resources.

The priority-based scheduling strategy proposed by T. Choudhari et al. [75] aims to reduce expenses without sacrificing response times. Fog nodes (also known as fog servers) from the fog layer architecture described in [76] were employed in this study. The fog layer's processors are under the watchful eye of the fog server's. Time limits have been set for submitting the various application sections. Tasks are prioritized according to their impact on maintaining an elevated service quality level and tracked using the due module dates. A module's priority determines which processors will process it. The proposed procedure does not think about making the most of the limited resources. In [77], Sun et al. proposed a two-level resource scheduling algorithm in which the fog layer is divided into clusters. Each resource is assigned to a different fog cluster or fog node within a given cluster. The researchers claim that they could reduce delays and improve job execution solidity.

In [25], Xuan-Qui et al. proposed a task scheduling technique for dependent tasks on each other. The author used the Directed Acyclic Graph *(DAG)* to represent tasks. The growth of these intertwined efforts requires constant and open lines of communication. A priority is created by traversing the *DAG* and assigning the prioritized jobs to the fog nodes. The assignment of modules is based on the fog devices' resource limitations. The proposed technique has a low resource utilization rate and increases the application's execution time.

A job scheduling and the image placement strategy on a storage server was proposed by Zeng et al. [78]. The embeded clients and the fog nodes perform the computations by using the shared storage. To ensure the timely completion of tasks without sacrificing quality, deadlines are set to expire as soon as feasible. Each step is meticulously planned to ensure the best possible user experience. J. Zhang et al. [71] proposed a dynamic resource allocation technique to improve the use of existing resources and, by extension, the value provided to users. Priced Timed Petri Nets *(PTPNs)* consider both the time it takes to complete the task and the trustworthiness of the fog nodes when choosing how to allocate the available resources.

A deadline-based resource provisioning and allocation method considering changing user needs has been presented by Naha et al. [79]. Time-sensitive applications cannot be run without addressing the issue of dynamic resource allocation in the fog Cloud environment. Their efforts aimed to allocate resources to decrease processing time while allowing them to complete their jobs by their due dates. Power, speed, and bandwidth are used to rank the available resources. The requests are processed in order of their fog layer priority. The Cloud-based assignments are sent to the cloud if fog cannot complete it. The authors tested their algorithm using CloudSim tool. The author claimed that 12% of the processing time was saved. Aside from that, they haven't done anything to improve efficiency in managing resources, responding to requests, or prioritizing resources.

Another scheduling and allocation approach was introduced by Abdelmoneem, R. M. et al. [80] named mobility-aware heuristic based scheduling and allocation approach *(MobMBAR)*. The system was developed for use in healthcare, making it mobile and adaptable to various settings. Patients' whereabouts have become the center of attention. Firstly, they have outlined a transfer procedure that aids in transporting mobile sink devices from one gateway to another. Consistent with the goals of this policy, the change in network status is indicated by a decrease in the intensity of the incoming signal. The maximum number of devices that may communicate with gateway is also considered in determining this policy. The organizing and allocating approach has two distinct halves.

One is the ranking stage which comes first, followed by the scheduling stage. After determining which tasks are most urgent, the ranking process begins. For this purpose the Weighted Sum Method *(WSM)* was employed in their deliberations whereby relative importance may be placed on each job which keeps an eye on the users' geographic location shift. *iFogSim* has been used to analyze how well the suggested process works. In terms of efficiency, the algorithm was superior to make-span compared to every other way. The outcomes were a shorter make-span, a smaller network the pressure, and faster reaction times. However, the researchers have no project scheduling and due dates regarding resources.

The Hosseinioun et al.[81] proposed an approach to reduce the make-span while increasing the resource usage. The dynamic voltage and frequency scaling *(DVFS)* algorithm was developed to reduce energy usage. A hybrid method, the Invasive Weed Optimization and Culture Algorithm *(IWO-CA)* is used to build valid sequences of tasks. There are two stages to the suggested approach. To begin, the author constructs a task queue with the optimum priority for scheduling purposes. Second, the fog servers can access the power and frequency needed. The authors claim that the HEFT method efficiently allocated jobs to processors. There are three phases of Invasive Weed Optimization *(IWO)*. The first stage is initiation, reproduction, geographical spread, and competitive extinction.

The jobs are assigned at startup using a height distribution approach, which considers the lowest and most incredible heights. These parameters are used to generate the first population. In the next stage, reproduction, the population reproduces by making new individuals based on the solutions that have been chosen with the least fitness. The generated seeds in discrete solution space are dispersed with the help of the significant code distance approach. However, short code distance is used to provide local optimizations. The authors claims while comparing the proposed method with *(EES)*, *(HEFT-T)*, *(DEWTS)*, and *(HEFT-B)* that the algorithm did well by reducing the make-span and maximized resource consumption. However, the author don't consider the reaction time and resource utilization in their efforts. Task scheduling and resource allocation are two significant obstacles to the fog-Cloud computing paradigm.

Priority Aware Genetic Method *(PGA)*, introduced by Hoseiny et al. [82], was a new scheduling algorithm designed to solve these problems. Their goal is to maximize the proportion of jobs finished before the due date while minimizing the total amount of time spent on computation. Their method prioritizes tasks and identifies the best node to carry them out. The suggested solution is broken up into two stages.

In the first stage, tasks are categorized according to factors such as due dates, while in the second stage, the *PGA* is applied to the organized tasks. After that, two to-do lists, fog-list and Cloud-list, are compiled. Dates due on tasks and the percentage of instructions given are used to compile these lists. The two lists are then processed by *PGA* simultaneously. The second step involves the employment of a binary masking approach for crossover. A roulette wheel selection technique often chooses the first parent in a crossover. Then, a new mask vector and second parent are chosen. The kids of a binary mask operator retain their parents' genetic makeup. Mask vector bits may be 0 or 1.

When the bit's value is one, gene swapping occurs between the parents, but genes have been copied without being swapped when it is zero. They have looked at how their method stacks up against Ant-Mating Optimization *(AMO)* and Power Of 2 Choices *(PO2C)*. The algorithm did better with reduction in processing time, however, they do not have a very varied datasets to work with. Further, resource prioritizing has not been considered.

2.8 Heuristic Based Resource Allocation

Resource allocation heuristics based on ”penalty and incentive schemes” were suggested by Z.Pooranian et al. [83]. The problem was seen as a ”bin packing penalty-aware” problem to allocate resources using fog servers considered as bins instead of containers. The *VMs*, on the other hand, are packs that must be served according to time and frequency constraints. For some iterations following a penalty, the server remains unallocated. D. T. Hoang [84] developed a heuristic-based job scheduling mechanism to optimize performance and get low latency in Cloud fog architecture. The author suggests an area-based fog architecture for task distribution in fog and Cloud environments.

A two-tiered resource scheduling method was presented by Sun et al. in [85], which partitions the fog layer into clusters. A fog cluster or fog node is first assigned responsibility for a particular resource. The author used the idea of genetic algorithm with non-dominated sorting for resource scheduling to optimize for many objectives concurrently across all fog nodes for a specific cluster. According to the researcher, this strategy reduced wasted time and boosted faith in the project's eventual success.

A recursive binary search approach was used to help the author choose the best possible instrument. However, the suggested method of transferring modules from the fog layer to the Cloud does not account for the price of Cloud services. A deadline-aware job scheduling system for tiered-based IoT architecture was described by Jianhua Fan et al. [86]. To determine the location of module placement, it is essential to put the service provider's requirements first. An empty or one-filled knapsack was the source of the issue. The primary objective was to maximize their long-term net profit with an optimization strategy based on Ant Colony Optimization *(ACO)*. Other algorithms, such as *FCFS* and Min-min, are compared to the proposed technique. Time and material lags are not taken into consideration in their assessment.

Zenith, developed by Jinlai Xu et al. [87], is the optimal method for allocating resources in edge computing. Service providers and edge infrastructure providers may be able to reach a legally binding agreement via resource allocation. Edge servers agree to the specific requirements to ensure that the offloaded application modules are scheduled on time and that the allotted computing resources are accessible. The authors introduced an independent variable resource allocation model. The proposed approach distributes edge computing resources independently of the provider's infrastructure. The cost of using Cloud services is not included in the suggested method. The objectives of Fog computing is to accommodate several programs with different latency requirements and delay thresholds. By considering the latency requirements the Liu et al. [88] proposed a strategy that combines the quickest feasible time to accomplish a job with the mining of association rules for minimum execution time. Rules for assigning tasks to Fog nodes are generated using the priori algorithm.

X. Pham, et.al. [89] proposed an algorithm to achieve a balance between application execution time and cost of using Cloud resources. The authors try to minimize the make-span of the interdependent modules while reducing the monetary cost of using Cloud resources. The estimated starting time, estimated finishing time, and monetary cost of mapping each for all processors is calculated. The make-span and monetary cost trade-off values are used for the assignment of tasks to the nodes. This approach results in load imbalance with poor resource utilization.

Gupta et al. [90] introduced a novel technique of task scheduling for latency-sensitive applications based on experimental data demonstrating the superiority of modules hosted near the network's edge over those hosted in the Cloud. To calculate the energy consumption, network utilization, and loop delay the author used the EEG Tractor Beam game by using the *FCFS* strategy. Fog computing's resource allocation was investigated by L. F. Bittencourt et al. [70], who compared the performance of latency-tolerant versus latency-sensitive jobs. The application's parts were structured using a contemporary *FCFS* and delay-priority approach. The concurrent scheduling method encounters difficulties when there are several scheduled modules and a limited number of shared resources. By shifting processing from the fog to the Cloud, *FCFS* slows down latency-sensitive applications and increases network traffic. To deal with latency-sensitive applications, the author devised a delay priority strategy that involves transferring them from fog to the Cloud in a hierarchical order of delay sensitivity.

However, the suggested technique starts to break down when dealing with many application modules, leading to more time spent in network traffic, longer execution times, and higher overall costs associated with using Cloud resources. Prioritization-based work scheduling was developed by T. Choudhari et al. [75] to reduce response times and costs. In the fog layer, jobs are received only when they are the least urgent. As assignments come in, the clock starts ticking and deadlines become more prominent. Several small data centers are spread across each fog layer and linked by a network of fog nodes. When the resources in a fog layer are exhausted, the work is transferred to the Cloud.

The suggested method aims to compromise small memory footprints and fast processing speeds. In addition, other important goals are overlooked, such as using the network efficiently. The S. Bitam et al. [91] proposed a scheduling algorithm drew inspired from the behavior of bee swarms. The suggested approach was developed with balancing memory use and computational performance as its key goals.

Reinforcement learning was utilized by K. Gai et al. [92] to enhance the quality of the user experience and maximize the use of available resources. The author discussed two reinforcing methods that may be used to improve the accuracy of cost-mapping tables and the distribution of resources. Each user request required its unique configuration of fog nodes to achieve a specific objective, such as lowering latency. The double deep Q-learning model *(DDQ)* was introduced by Zhang et al. [93] for use in edge computing to reduce energy consumption. Several techniques for dynamically adjusting voltage and frequency scaling *(DVFS)* were evaluated using the *(DDQ)* technique. The author used linear rectified units as activation functions rather than sigmoid ones to prevent gradient vanishing.

Rahbari et al. [94] used the greedy knapsack-based scheduling *(GKS)* approach for scheduling the tasks in Cloud fog environment. The researchers claims that the suggested method achieves better energy savings and execution cost results than concurrent scheduling, *FCFS* scheduling, and delay priority scheduling.

Mai et al.[95] used deep reinforcement learning for real-time job scheduling in their work. The researchers used a reinforcement learning approach and evolutionary strategies to train a neural network. Consequently, only a handful of algorithms that make good use of available resources have been examined since Fog computing is still in its infancy. Despite fog computing's infancy, relatively few studies have examined Big data applications in fog environments So far, most of the efforts have been focused on improving the healthcare system. For the purpose of managing Big data in Fog computing for telehealth applications, a service-oriented architecture has been suggested as a potential solution [96].

The fog layer of the network was augmented with low-powered embedded computers so that this architecture could be produced. The raw data that these embedded computers have access to be processed through data mining and analytics by the embedded computers. The use cases of speech motor disorder and cardiovascular detection were both considered for this research. The fog layer was responsible for the processing of raw data for both of the use cases that were considered. Once the raw data from speech motor disorder sensors or cardiovascular detection sensors have been processed, all detected patterns are saved, and the one that is most unique is uploaded to the Cloud for further analysis.

During the preliminary research, it was determined that the smartwatch was the one responsible for sending speech data to the fog device. The fog device is responsible for carrying out the feature extraction, pattern mining, and compression stages that follow the previous step in the processing sequence. The fundamental frequency and loudness characteristics of speech are altered in this manner so that they are transformed into an average form. After that, the data representing the speech is further compressed before being uploaded to the Cloud. The Cloud is responsible for carrying out the necessary processing to convert the average fundamental frequency and loudness that it has received into the time series of the original speech. This conversion is carried out in real time. For the purpose of this specific case study, the Dynamic Time Warping *(DTW)* method was utilized for the purpose of voice data mining in order to extract information. This was done in order to better understand the data. In addition, the Clinical Speech Processing Chain *(CLIP)* algorithm was added in order to compute relevant clinical metrics such as fundamental frequency and loudness. The mining of voice data was made possible by utilizing both of these algorithms in conjunction with one another. *DTW* was utilized in the other case study, which centered on *ECG* monitoring, with the objective of pattern recognition. Following the discovery of the pattern, the data that had been previously processed were transferred to the Cloud so that they could be subjected to additional analysis [96].

Kabirzadeh et al.[97] presented a hyper-heuristic approach to fog network schedules, focusing on the difficulty of scheduling workflows in a Fog computing setting. Various nodes in a sensor network serve as receivers, soaking up data about their local surroundings. When the nodes are in the communication range of the gateways, they will send the information to the fog machines. The most efficient way of assigning virtual machines *(VMs)* to particular modules in fog-based systems was determined using meta-heuristic techniques. A fitness function examines each algorithm's results to determine whether the modules are utilizing their resources efficiently. They propose picking a heuristic that works well for the current task as part of their strategy. By shrinking the size of a heuristic algorithm, it could improve its decision-making ability for distributing resources with specific constraints to users following the sort of workflows while reducing simulation time and energy usage. Algorithmically, the authors consider the specific workflow that was selected using the best available heuristic algorithm's inputs. Although current efforts in Fog computing such as resource management are designed for non-VM applications such as task scheduling, virtual machine containers can be utilized for *VM*s since they are quicker and require fewer resources. It takes considerable time to boot and set up containers or virtual machines.

The delay in execution is a significant problem for time-sensitive applications. For example, a virtual machine must be established on the fog node before a request can be processed. A *VM* may still run on fog nodes if the power goes off. The *VM* might begin processing as soon as a request is received. However, resource waste will be unavoidable if these conditions are placed on the use of the virtual machine. In addition to transmitting image files, virtual machines may be assigned tasks and resources without worrying about where they will be located.

Yin et al.[98] propose methods that might shorten the time required to complete jobs. The authors made significant contributions by researching optimal work schedules and developing intelligent manufacturing processes. Their scheduling strategies are based on the container concept, which is at the heart of the author's model of containers and task processing. They have broken down the tasks at hand into manageable chunks, each of which must be reviewed, authorized, and then assigned to the fog, a single computer, or an abstract network. The work is considered finished if the amount of time it took to do is less than the maximum time permitted.

The container resource allocations may be altered without adding extra expenses to the system, which is not the case with virtual machines. Because of this, the number of jobs assigned to a fog node ensures that all available resources are used effectively. It is expected that the reassignment strategy would also significantly reduce the delays. According to their findings, the scheduling algorithm and the reallocation mechanism can potentially increase the number of tasks accepted by 5%. A possible 10% reduction in work time is possible using this method.

When simplifying the service model, they brush over the inevitable limitations of Cloud-based assets in favor of removing the related computing Cloud. S. Bitam et al. [76] used the Bee's swarms to optimize the scheduling of tasks in a Fog Computing. For effectively allocating modules to fog devices, the authors proposed a bio-inspired approach based on the Bee's life algorithm. The suggested method aims to find the best balance between processing speed and memory use. Particle Swarm Optimization *(PSO)* and a Genetic algorithm *(GA)* are contrasted with the proposed method based on a simulation of Bee's life to determine which is more successful and efficient. However, the suggested approach disregards the significance of resource allocation and network bandwidth in optimizing the processing performance of the underlying Fog computing environment.

To guarantee the consistent scheduling of scientific operations in the Cloud, Rodriguez, et al. [99] created a workflow responsive resource provisioning and scheduling *WRPS* method. Its purpose is to save the user time and money by facilitating the completion of Cloud-based activities. The proposed approach is adaptable to delays and varying conditions common in Cloud computing. To solve this problem in pseudo-polynomial time, the authors break the workflow into bundles of similar tasks and pipelines with the same deadline.

The main goal of the proposed approach is to complete the task within the user-specified deadline while utilizing as few resources as possible. Their simulation findings show that it is robust and sensitive to the performance fluctuation of the Cloud, can scale concerning the number of activities in a workflow, and can deliver solutions of better quality than those supplied by state-of-the-art algorithms. A heuristic-based system for scheduling tasks has been presented for use in a hybrid fog Cloud environment by Ali et al. [100].

The authors state that determining whether or not to carry out operations at the fog or cloud device is a significant problem. The suggested method aims to increase the tasks success rate while decreasing their make-span and average turnaround time. When a task is received, the fog layer determines whether it can be processed locally or must be sent to the Cloud by sorting their to-dos in increasing order of urgency to ensure they get things done on time even while the clock is ticking. Once that is done, the least busy virtual machine *Vm* is given the job based on the task's computational needs.

The authors used the *iFogSim* simulator to test their approach and compared the results to those obtained using the Real Time Task Processing *(RTTP)*, the First-in, First-out *(FIFO)*, and the Shortest-Job-First *(SJF)* algorithms. To reduce the load on the fog layer's resources the proposed approach sends requests with flexible due dates to the Cloud.

Table 2.1: Summary of Literature Review Techniques

References	Response Time	Make Span	Monetary Cost	Resource Utilization
Luiz F. et al. [70]	✗	✗	✓	✓
Taneja et al. [24]	✓	✗	✓	✓
Xuan-Qui et al. [25]	✓	✓	✓	✗
V.Cardelli et al. [26]	✗	✗	✓	✓
D. Rahbari et al. [72]	✗	✗	✓	✓
R. Mahmud et al. in [14]	✗	✗	✓	✓
X. Pham et al. [89]	✗	✗	✓	✓
S. Agarwal et al. [74]	✓	✗	✓	✓
T. Choudhari et al. [75]	✓	✗	✓	✗
S. Bitam et al. [76]	✗	✗	✓	✗
J. Zhang et al. [71]	✓	✓	✓	✓
Naha et al. [79]	✓	✗	✓	✗
Abdelmoneem et al. [80]	✓	✓	✓	✗
Hosseinioun et al. [81]	✗	✓	✗	✓
Z.Pooranian et al. [83]	✓	✗	✓	✗
D. T. Hoang [84]	✓	✗	✗	✓
Sun et al. [77]	✗	✗	✓	✓
Jianhua Fan et al. [86]	✓	✗	✓	✗
Jinlai Xu et al. [87]	✓	✗	✗	✗
Liu et al. [88]	✗	✗	✓	✓
K. Gai et al. [92]	✓	✗	✓	✓
Rahbari et al. [94]	✗	✗	✓	✓
Mai et al. [95]	✗	✗	✗	✓
Kabirzadeh et al. [97]	✗	✗	✓	✓

Yin et al. [98]	✓	✗	✗	✓
Rodriguez et al. [99]	✗	✓	✓	✗
Ali et al. [100]	✓	✓	✗	✓
Xu et al. [101]	✓	✓	✗	✗

When allocating resources, the authors fail to take into account the relative importance of various tasks.

Xu, J., et al. [101] proposed a work scheduling technique *(LBP-ACS)* that combines laxity-based priority algorithms with ant colony systems. The deadline for each work is included in the overall priority ranking. A laxity-based algorithm is combined with an ant colony system to reduce overall energy usage and establish a strategy for scheduling jobs. Each task's laxity is determined using the laxity-based priority algorithm *(LBA)*, which is then used to determine the relative importance of smaller jobs.

After that, the jobs in the Directed Acyclic Graph*(DAG)* are organized and sequenced. The sensitivity of jobs is represented by the shortest amount of time that may be postponed before the deadline. For a given task, a lower laxity number suggests more importance. The ant colony-based constrained optimization method *(COA-ACS)* is utilized to delegate jobs which take a list of jobs and *VM*s as input. Following the determination of the first pheromone, the appropriate assets may be chosen.

After finding the both regional and international pheromone values the final decision is based on the least draining route's energy requirements. The effectiveness of the suggested method is measured against those of greedy for energy *(GFE)*, and heterogeneous earliest finish time *(HEFT)*. The authors claim that *LBP-ACS* scored better than its competitors in terms of failure rate because it considers due dates when determining which activities to prioritize. Prioritizing resources and prompt responses have not been taken into account. The summary of the literature review is given in Table 2.1.

2.9 Gap Analysis

From the literature, it has been concluded that different factors including application type [70], monetary cost [76, 86, 89], makespan [25, 73], latency [14, 87, 102], network usage [24, 70, 72] and execution time [74, 75] affect the placement of application modules in Fog computing environment.

To deal with latency a number of scheduling approaches have been proposed in [14, 70, 87, 102]. Latency oriented scheduling heuristics results in reduced network usage. However, these scheduling schemes cause longer execution time of applications and resource contention at fog layer. The scheduling approaches in [24, 25, 73–75] achieve a balance between application execution time and monetary cost of using Cloud resources. However, these approaches suffer from poor resource utilization at fog layer with increase in bandwidth usage. In order to meet *QoS* in terms of user-defined deadline, the proposed approaches [14, 86, 102], reduce the probability of deadline violation to a reasonable level; however, results in decreased resource utilization and increase network usage. To address these issues and achieve maximum resource utilization at fog layer with minimum execution time of the incoming applications, reduced the network usage and the monetary cost of the applications there is a need to have a scheduling technique which can reduce the execution time of the applications with less bandwidth consumption and the monetary cost of using Cloud resources.

Chapter 3

Resource Aware Cost-Efficient Scheduler for Cloud Fog Environment

3.1 Introduction

The Internet of Things *(IoT)* facilitates the development of novel social relationships between people and their technological devices. The government has invested heavily in the *IoT* because of its potential benefits in life and different fields of science like municipal management, e-health, disaster recovery, etc. The number of devices communicating with one another has skyrocketed as the *IoT* has grown [2]. A 2015 press release [103] on the *IoT* shows that the total number of internet-connected devices has already reached $13.4 billion, surpassing the human population. It is estimated that by 2020 [103], there will be 38.5 billion of these devices, or around five for every person on the planet. It is predicted that by 2025, the *IoTs* might affect the global economy by around $11 trillion annually [1], as much as 11 percent of the whole economy.

The growing volumes of data created by the many *IoT* devices need substantial computing power. To fulfill the demands of computing resources, a great number of data centers have been set up. According to the conventional view of Cloud computing, customers of data services would benefit from efficient and flexible services [4]. There are two leading players in the traditional Cloud computing model:

1. Consumers who use Cloud resources

2. Cloud service providers

Companies that acquire and make available different Cloud resources (such as data storage, data processing, etc.) to end users are known as "Cloud service providers" *(CSP)*s. The Cloud service provider provides its customers with a selection of pay-as-you-go resources [104]. The *CSP* typically provides a web-based interface via which the Cloud user can access the leased resources.

The task scheduling heuristic [105] receives jobs from clients through tasks/-cloudlets from the Cloud task manager. A user's computationally difficult task is received and then distributed to the virtual machine *(VM)* that is best suited to do it. The task scheduling heuristic can collect information about *VM*s computing power, current workload, etc., by exchanging information with resource management. A resource manager oversees and controls the Cloud's virtual and physical resources. The availability and settings of a pooled set of resources may be adjusted on the fly. Cloud computing stores and processes data in remote data centers connected by supercomputers and lightning-fast networks.

For the processing and data storage needs of *IoT* devices, Cloud computing can offer scalable infrastructure-level services; however, it is not necessarily the best practice for *IoT* devices to use Cloud computing. Cloud data centers can not adequately meet the latency needs of most globally scattered *IoT* applications like healthcare, emergency response, video streaming, etc., resulting in sluggish distribution, clogged networks, and low Quality of Service *(QoS)* [5].

Depending on the amount of time and effort required for data transfer from the user to Cloud resources, a delay may occur [106]. The Fog computing that Cisco helped create in 2012 [19] aims to solve these problems by working with the Cloud. Fog computing is a network-peripheral implementation of Cloud computing [6, 7]. Standard pieces of networking hardware like routers, hubs, and switches are typically used in Fog computing architectures. It is possible to put these network hardware devices closer to IoT as presented in Figure 3.1.

One other way these devices boost the efficiency with which customers' applications may be operated is the capacity to store data locally or in the Cloud and connect to other devices and the internet. In addition to facilitating location awareness, mobile support, real-time communication, compatibility, and scalability, Fog computing is a critical component of these technologies [8]. One of the main ideas behind Fog computing is that data processing power should be kept as near as possible to the original data creation point [107].

As a result, Fog computing is beneficial in several ways, including reduced network traffic, improved service latency, and lower overall costs associated with using Cloud resources. Thus, instead of simply relying on Cloud computing, IoT applications may use Fog computing to meet user needs [10] .

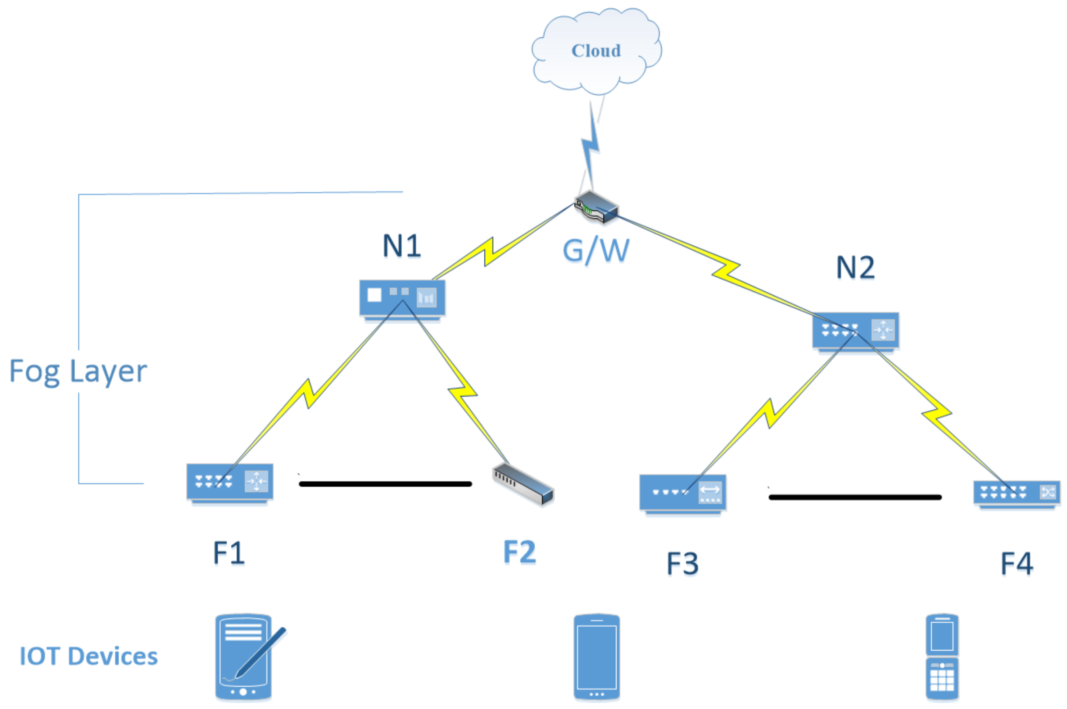

Figure 3.1: Cloud Fog Architecture

3.2 Architecture of Fog Computing Environment

The Fog computing concept introduces a new computational paradigm in which services like computation, communication, and storage are at the network's edge. The core concept of Fog computing is to place data storage and computation close to the end-user (i.e., at the network's edge). Cloud computing and Fog computing are not detached from each other as Fog computing can utilize the Cloud resources for application demanding large scale processing and storage requirements. This provides a novel type of applications and services [8]. Cloud-Fog architecture along with end-users collectively formulates three-tier architecture [108] of the Fog computing environment as revealed in Figure 3.1.

3.3 The Three-Tier Architecture of Fog Computing Environment

Most of the architectures suggested for Fog computing in recent years come from the same three-layer framework. Fog computing introduces a layer between end devices and Cloud called the fog layer, which enlarges Cloud services to the network edge [109]. The three-tier hierarchical architecture consists of the following three layers:

3.3.1 User Layer

The user layer as shown in Figure 3.2 is the nearest layer to the user and consists of IoT devices like sensors, smart cards, readers, and mobile phones. Although mobile phones have computation power, yet these devices are often used as smart sensing devices in the Fog computing environment. IoT devices are geographically dispersed and are accountable to sense data from physical objects or environments. IoT devices transferred the data which is sensed, to the fog layer as depicted in Figure 3.2 for necessary storage and processing.

3.3.2 Fog Layer

Fog layer as presented in Figure 3.2 is situated at the network edge and is the fundamental component of the Fog computing environment. Fog layer is formed of traditional networking gadgets (i.e., switches, routers, set-top boxes, proxy servers, and base station *BS*, etc.,). These devices may be static at some fixed locations like parks, streets, cafes, shopping centers, and bus terminals, etc., or movable with a moving carrier and are widely distributed among Cloud and end devices. These devices may be resource-poor (low computation power, memory, etc.,) such as access points, routers [110], or have resource-rich devices such as cloudlets [23].

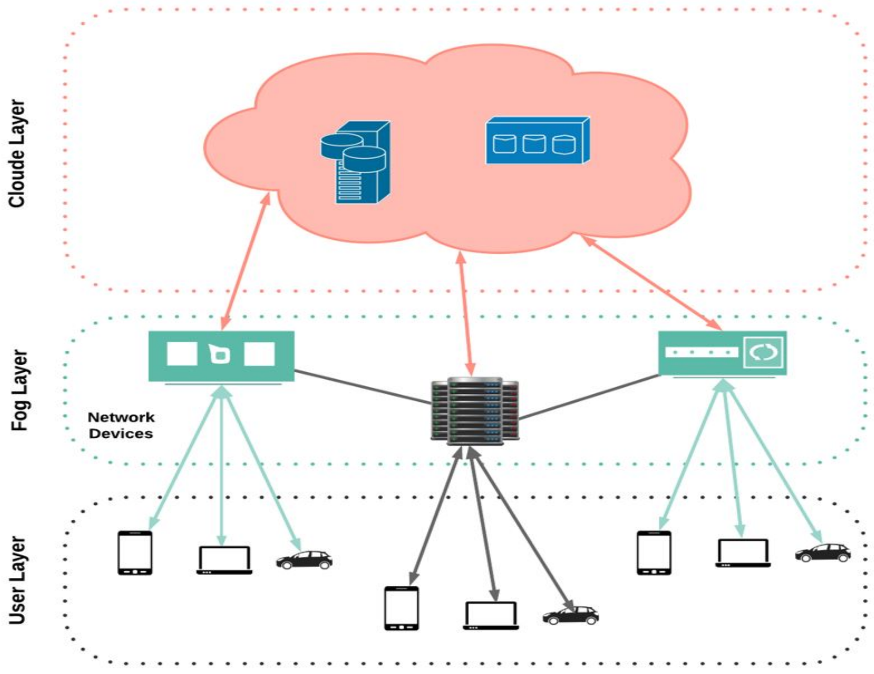

FIGURE 3.2: The Three-Tier Architecture of Fog Computing [111]

3.3.3 Cloud Layer

Cloud layer as shown in Figure 3.2 comprises of several high capacity servers and storage devices delivering various application services. It can process and store a huge amount of data. Though, like traditional Cloud computing, the Cloud layer does not have to perform all storage and computations [110]. Cloud resources are used for application demanding large scale processing and storage requirements.

3.4 Scheduling in Fog Computing

An application module must be mapped with resources (such as CPU, memory, bandwidth, etc.) to make the best use of available resources via a process called scheduling. A fog scheduler's primary responsibility is to determine which stage of an application is next to run based on the availability of suitable modules. Reduced execution time of applications, minimal monetary cost, increased throughput, less bandwidth consumption, etc., are some of the primary goals of a fog scheduler.

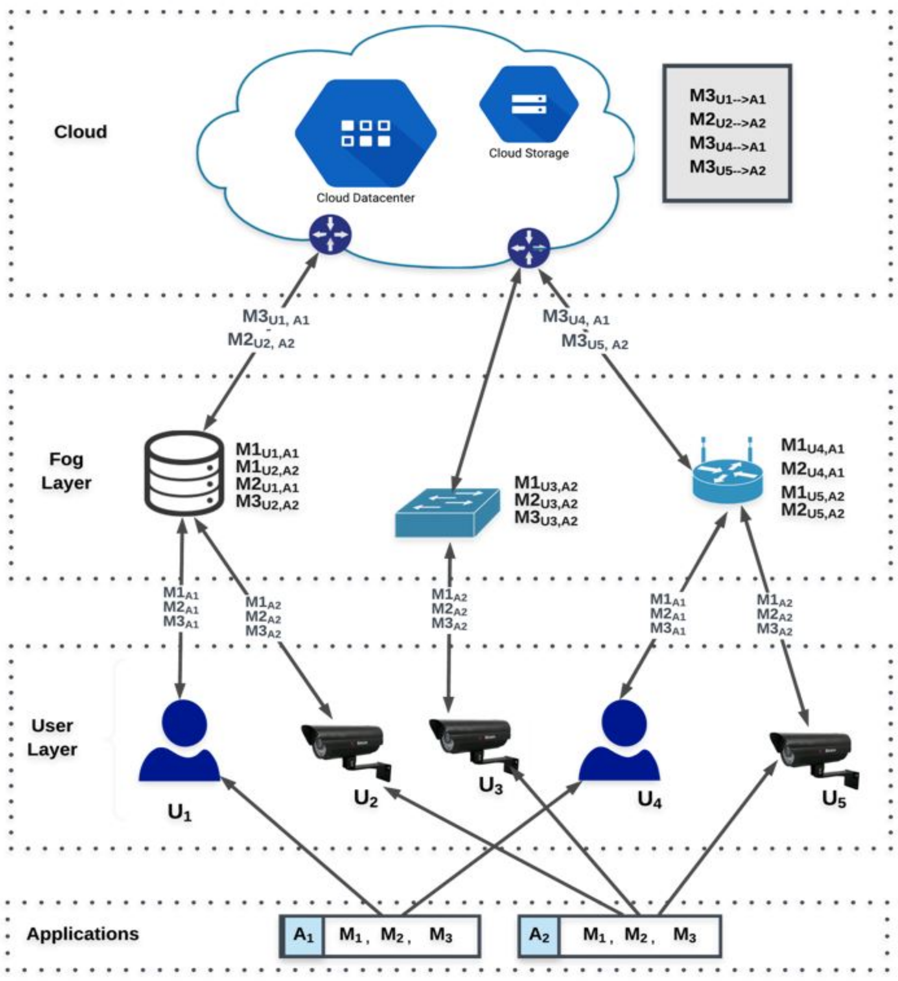

Figure 3.3: Scheduling in Cloud Fog Computing Environment

The inappropriate allocation of the resources in the Fog computing environment degrades the execution time of application, increases latency, increases in monetary cost, and failure to meet the service level agreements. A scheduling example in the Fog computing environment is presented in Figure 3.3 that shows a hierarchical, bi-directional computing infrastructure. Edge devices can communicate with cloudlets and other network devices at the fog layer. Five users (U_1, U_2, U_3, U_4, and U_5) are presented in Figure 3.3 at the user layer.

Every user is running an application either A_1 or A_2. Each application comprises three modules (M_1, M_2 and M_3). The scheduler at the fog layer, based on defined scheduling policies, decides which application modules will be placed at the fog layer and which modules will be executed at the upper layer i.e., Cloud. As shown in Figure 3.3, modules M_1 and M_2 of A_1 for U_1 and M_1 and M_3 of A_2 for U_2 are placed at fog layer while M_3 of A_1 for U_1 and M_2 of A_2 for U_2 are placed at Cloud. For U_3 all application modules (i.e., M_1, M_2, and M_3) are successfully mapped at fog device in the range of U_3. Similarly, for U_4 and U_5, modules M_1 and M_2 of A_1 for U_4 and M_1 and M_2 of A_2 for U_5 are placed at fog layer while M_3 of A_1 and A_2 for U_4 and U_5 are placed at Cloud.

This study introduces a resource-aware scheduler to allocate the arriving application modules to fog devices. The proposed innovative technique for moving application modules from the fog to the Cloud layer allows for more efficient use of Cloud resources at a lesser expense of execution time, network bandwidth, and the Cloud resources cost. Following is a summary of the research's most significant findings:

- We provide a Resource Aware scheduler for moving application modules between the fog and the Cloud layers, which helps to make the most efficient use of available resources.
- We have defined the priorities for the application module placement to reduce data transfer, run time, and cost of using the Cloud.
- The resource aware policy has been implemented using *iFogSim*'s simulator and compared to conventional Cloud placement methods and fog placement strategies. As the findings reveal, our approach has led to a dramatic increase in efficiency. The abbreviations used in the paper are listed in 3.1.

TABLE 3.1: Preliminary Notation Used in RACE System and Performance Model

Notations	Descriptions
$\boldsymbol{RACE}$	Resource Aware Cost-Efficient Scheduler
$\boldsymbol{M_i}$	i^{th} Module
$\boldsymbol{MI}$	Millions of Instruction
$\boldsymbol{MIPS}$	Millions of Instruction per second
$\boldsymbol{FD}$	Fog Device
Cloud	Cloud Datacenters
Pri	Prioritized List
$\boldsymbol{CT}$	Computation Threshold
$\boldsymbol{BT}$	Bandwidth Threshold
$\boldsymbol{CR}$	Computation Requirement
$\boldsymbol{BR}$	Bandwidth Requirement
$\boldsymbol{FOP}$	Fog only Placement
$\boldsymbol{CFP}$	Cloud Fog Placement
$\boldsymbol{TCP}$	Traditional Cloud Placement

The rest of the chapter is organized as follows: The problem formulation is described in section 3.5 with the application model for the proposed algorithm is explained in section 3.6. The architecture and the algorithms of $RACE$ are explained in section 3.7 and 3.8 respectively. The experimental evaluation is discussed in 3.9 with the discussion of simulation results in 3.10. Finally the chapter is concluded in section 3.11.

3.5 System Model and Problem Formulation

This section explores a thorough overview of the projected scheduling heuristic Resource Aware Cost-Efficient Scheduler *(RACE)*. The system architecture, application model, algorithm, and overhead analysis are disclosed in this segment. By combining the advantages of edge and Cloud computing, the fog Cloud solution facilitates low-latency communication across a large geographical area, thus enabling for a computation, networking, and storage resources distribution framework between the unified Cloud and the remote end-users. In reality, it encourages versatility as well as resource and device flexibility.

The user layer consists of end devices, while the fog layer has small servers, routers, access points, gateways, etc., which can host application modules [112]. The devices at the fog layer have limited processing power because of the hardware constraints of the resources each one possesses. In contrast, the Cloud contains the same resources produced by several *VMs*, each of which may be set to a unique configuration that reveals the *VMs* computational power. Because the Cloud as a whole has so much more computational and resource capacity than a typical system in the fog layer, the Cloud seems limitless to the fog devices.

Hence, the complete range of resources comprises of fog devices and the Cloud *VMs*. The network node's computing power (or resource capacity) is constrained by a common set of limited constraints and may be defined as a series of three simple narrow down parameters, termed as RAM, CPU, and Bandwidth. Similarly, applications with their modules may be intensive in computation or intensive in bandwidth [30, 90, 113, 114] as required by their computation and bandwidth requirements. Thus if M_i represents an application module and CR shows the computation requirement for the application module in module set then it can be represented as follows:

Let $M = \{M_1,\ M_2,\ M_3\ ....\ M_n\}$ be the set of modules of user input jobs with their computation and bandwidth requirement as follows:

Let $C = \{CR_{M1},\ CR_{M2},\ CR_{M3}\ ....\ CR_{Mn}\}$ be the computation requirement of each module in module set M.

Let $B = \{BR_{M1},\ BR_{M2},\ BR_{M3}\ ....\ BR_{Mn}\}$ be the bandwidth requirement of each module in module set M.

Let $F = \{FD_1,\ FD_2,\ FD_3\ ....\ FD_n\}$ be the list of fog devices are arranged in descending order of their computation power.

$$M_i = \{CR_{Mi}, BR_{Mi}\} \tag{3.1}$$

Where i = 1 to n. Equation 3.1 shows the computation requirement CR_{Mi} and the bandwidth requirement BR_{Mi} of each module in a consolidated way. In order to find threshold for the computation and bandwidth requirement of application modules we used the median of the given values. As the mean is the average of the given values, while a mode is the frequency of occurrences of a given number. Since we want to separate the upper half from the lower half, therefore we use the median.

$$CT = median(C) \tag{3.2}$$

$$BT = median(B) \tag{3.3}$$

Where $\boldsymbol{CT}$ is the computation threshold and $\boldsymbol{BT}$ is the bandwidth threshold for the module M_i in module set $\boldsymbol{M}$. The input modules may be compute-intensive, bandwidth-intensive, or both the compute and bandwidth-intensive. In order to address either of these three scenarios, this research defines a priority list $Pri(M)$ comprising of four different categories C_1, C_2, C_3, and C_4 w.r.t their computation and bandwidth requirement. These categories are combined from C_1 to C_4 respectively and make another list of application modules as $Pri(M)$, the prioritized list of modules.

$$Pri(M) \begin{cases} CR_{Mi} > CT \&\& BR_{Mi} > BT \rightarrow C_1 \\ CR_{Mi} < CT \&\& BR_{Mi} > BT \rightarrow C_2 \\ CR_{Mi} < CT \&\& BR_{Mi} < BT \rightarrow C_3 \\ CR_{Mi} > CT \&\& BR_{Mi} < BT \rightarrow C_4 \\ \quad \text{where} \quad i = 1 \quad to \quad n \end{cases} \tag{3.4}$$

The compute-intensive modules from $Pri(M)$ are assigned to the available heavy/powerful fog devices. After these modules have been assigned, the next assignment is for the modules with less compute but bandwidth intensive as in C_2, the remaining modules in the priority list.

These modules are mapped to the fog devices as moving such modules to the Cloud consumes the bandwidth of the network with time delay for the output increases the execution time of the application. After the successful mapping of these modules, the next assignment is for the C_3 the modules with less compute-intensive and less bandwidth-intensive. These modules are mapped to the available fog devices; however, in case, all fog devices are full and cannot accommodate more modules, then the modules will be assigned to the Cloud.

After the selection of modules from $Pri(M)$ we have two different scenarios for assigning modules to the available fog devices. If the number of modules are less than or equal to the number of fog devices, then the first module is given to the first fog device and the second module is assigned to the second fog device and so on until all modules are mapped to the available fog devices. The module mapping for the scenario is represented in equation 3.5 given below.

$$M : Pri(M_i) \rightarrow FD_j \quad \text{where} \quad i \leq j \tag{3.5}$$

In other scenario when number of modules are greater than the number of fog devices, the module mapping for the prioritized list $Pri(M)$ of modules is presented in equation 3.6.

$$\begin{aligned} M : Pri(M_i) \rightarrow FD_j \quad \text{where} \quad i \geq j \\ \text{if} \quad MI(M_i) \leq MIPS(FD_j) \end{aligned} \tag{3.6}$$

After each successful assignment of M_i to FD_j, MIPS of FD_j will be updated as presented in equation 3.7 given below.

$$FD_{j*} = MIPS(FD_j) - MI(M_i) \tag{3.7}$$

If the condition in equation 3.6 get fails (assignment unsuccessful), then the fog device will be removed from the available list of fog devices F and the same is repeated for the updated list of fog devices F_* (equation 3.8).

$$F_* = F \setminus FD_j \tag{3.8}$$

If the F_* becomes empty and the $Pri(M)$ still have unmapped modules, then all the remaining modules will be assigned to the Cloud (equation 3.9).

$$Pri(M) \rightarrow Cloud \tag{3.9}$$

To improve the execution time of the application modules with minimum bandwidth and less monetary cost of using Cloud resources, we also considered the execution time and finish time of the application modules mapped at the fog layer. The modules are assigned to the fog devices on the basis of their arrival in the $Pri(M)$, the first module mapped to the first fog device available in the list, the second module mapped to the second fog device, and so on. The module mapping is shown in equation 3.10 as follows.

$$M : Pri(M) \rightarrow FD_j \quad \text{where} \quad i = j \tag{3.10}$$

After the successful mapping of modules from $Pri(M)$ to FD_j equal to the number of fog devices, remaining modules from $Pri(M)$ will be mapped as follows in Equations (3.11) to (3.15). In equation 3.11 ET computes the execution time of the i^{th} module M_i on j^{th} fog device FD_j. The computed execution time is used in equation 3.12 to find the execution time of all application modules placed at the fog device FD_j.

$$ET(M_i) = \frac{MI(M_i)}{MIPS(FD_j)} \tag{3.11}$$

$$X = \sum_{i=1}^{n} ET(M_i) \rightarrow FD_j \tag{3.12}$$

In equation 3.13 k is any arbitrary module from 1 to n which is going to be map on any arbitrary fog device FD_j where j varies from 1 to n.

$$Y = ET(M_k) \rightarrow FD_j \tag{3.13}$$

The estimated finish time of the k^{th} module on j^{th} fog device is the sum of the execution time of all application modules placed on the j^{th} fog device along with the execution time of k^{th} module going to be placed on the j^{th} fog device. Equation

3.14 compute the estimated finish time of the k^{th} module on j^{th} fog device.

$$EFT(M_k) = X + Y \tag{3.14}$$

After finding the EFT of the k^{th} module on all the available fog devices the k^{th} module will be placed on the fog device, for which the k^{th} module has the minimum finish time as computed using the equation 3.15.

$$M_k \rightarrow (min(EFT(FD_j), M_k)) \quad \text{where} \quad 1 \leq j \geq n \tag{3.15}$$

3.6 Application Model

On the basis of aforesaid scenarios and data-operations, we have deliberated the application model given below for the experiments. Figure 3.4 shows the application with different application modules. Each application comprises a number of modules with bandwidth requirements and computation requirements.

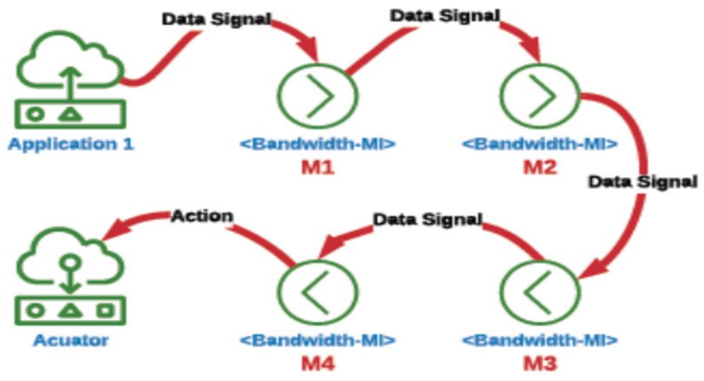

FIGURE 3.4: Application Model

3.7 RACE System Architecture

RACE is based on the scheduling of application modules on the basis of their computation and bandwidth requirements. *RACE* comprises of two sub schedulers

Module categorization and Module mapping. The system architecture of *RACE* is depicted in Figure 3.5.

The first step is to sort the fog devices and application modules in ascending order as per their computation power and computation requirements respectively. The sorted list of application modules is passed to *ModuleCat* function. The *ModuleCat* (shown in Figure 3.5) classifies the application components as per their computational and data transfer needs and provides a ranked list of those modules (PL).

The next step is to check the PL for the availability of application modules. If PL is non-empty, then a list of fog devices LF_d is checked for the availability of a fog device F_i for module mapping. The modules are mapped to the selected fog device F_i until its CPU capacity (i.e., remaining CPU capacity is greater than the module computing requirement) is full. When the CPU capacity of F_i is full, F_i is removed from the LF_d. When LF_d becomes empty and PL is not empty then all remaining modules from PL are mapped to the Cloud.

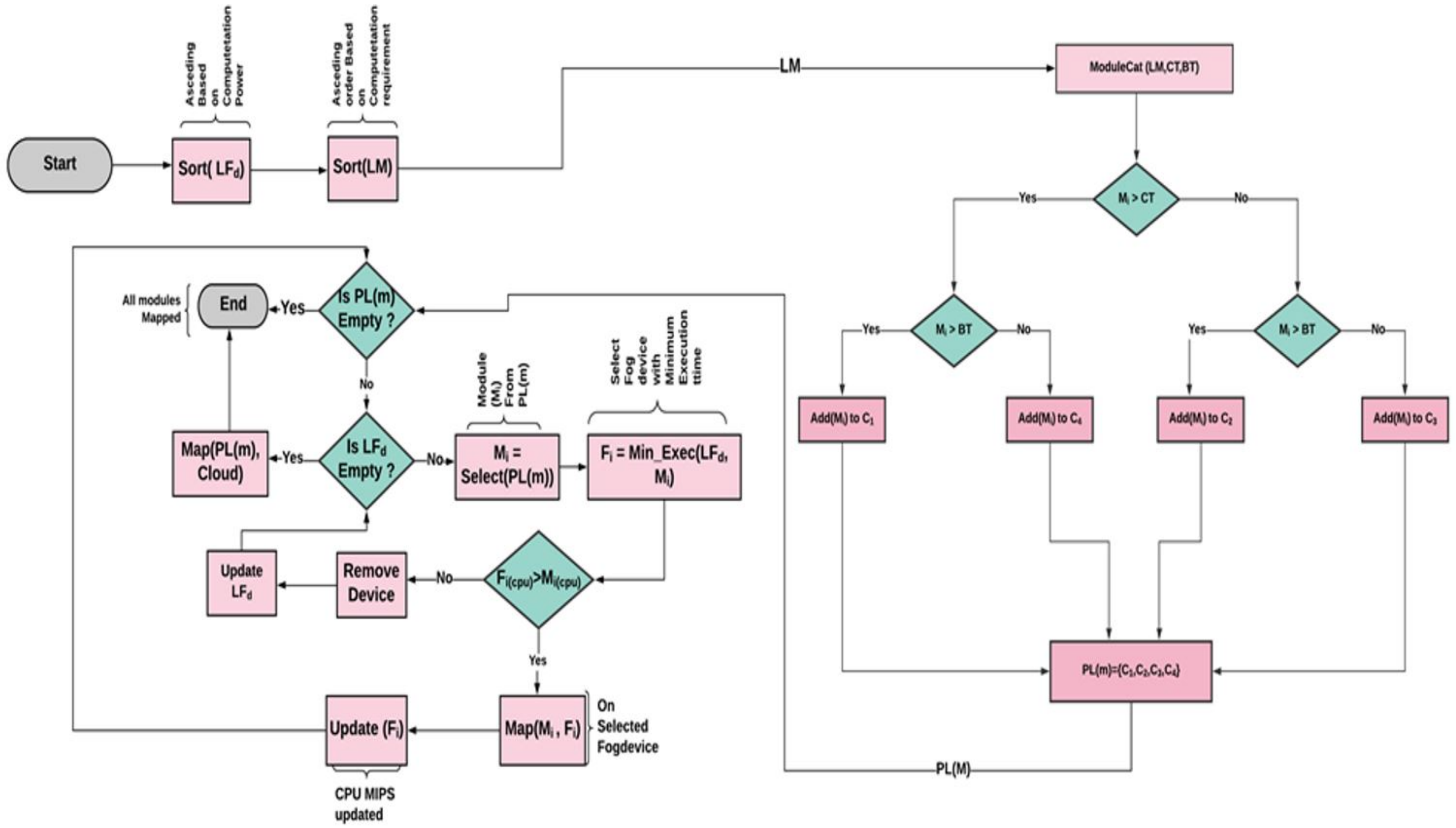

FIGURE 3.5: RACE System Architecture

3.8 RACE Algorithm

This section explains the proposed $RACE$ algorithm (shown in Algorithm 1) with its two primary modules i.e., Module Scheduler and Compare Module. List of modules (LM) with module computation requirement in million instructions MI is provided as an input to Algorithm 1 for mapping modules at fog devices. The Algorithm 1 first sorts the LF_d and LM in ascending order as per their computation power and computation requirement respectively (lines 2-3). The for loop from line 3 to line 16 classify the modules into four categories i.e.,
(1) High computation High bandwidth
(2) Low computation High bandwidth
(3) Low Computation Low bandwidth
(4) High computation Low bandwidth
as per their computation requirement and bandwidth requirement. Line 17 concatenates these four categories and make the PL. The for loop from line 16 to line 24 selects each M_i from PL and check the appropriate F_i for the placement of M_i. The process of selecting F_i from LF_d for the placement of M_i is elaborated in Algorithm 2 and is explained below. LF_d and M_i are provided as an input to Algorithm 2. The while loop starts at line 2 and repeats until LF_d becomes empty or all modules have successfully mapped at fog devices. Line 3 checks if the LF_d is empty, it means all devices at the fog layer are full, and control moves back to Algorithm 1. In case the fog devices are available at fog layer then loop from line 6 to line 15 verifies the availability of the F_i. At line 7 F_i is selected with minimum execution time. After selecting the F_i at Line 7, the selected F_i is checked, if the available F_i has enough CPU capacity to accommodate the incoming module, will return to Algorithm 1. In case the selected F_i could not accommodate the module (line 11), the selected F_i will be removed from the LF_d and the same is repeated for the new LF_d. After the selection of F_i for the placement of M_i, line 20 in Algorithm 1 updates the capacity of the selected fog device. If no appropriate fog device is found for placement, then the remaining modules are mapped to the Cloud (line 22 in Algorithm 1). In this way, the proposed strategy iterates from

```
Input: i) List of Fog devices $LF_d$ with their Processing Power ($MIPS$)
ii) List of Modules $LM$ with their computation requirement $CR$ and
bandwidth requirement ($BR$)
Output: modulemap[][]: Mapping of modules $LM$ on $LF_d$
MODULEMAP (List of Fog devices $LF_d$, List of modules $LM$)
begin:
Sorting of Fog devices according to their computation power in ascending
 order
Sorting of modules according to their computation requirement in ascending
 order
Compute $CT$ = Median of the $CR$ of all modules in the list
Compute $BT$ = Median of the $BR$ of all modules in the list
for $i = 0$ to $L_M.size$ do
    if $LM_{i_{CR}} > CT$ and $LM_{i_{BR}} > BT$ then
        add $LM_i$ to category 1
    end
    else if $LM_{i_{CR}} < CT$ and $LM_{i_{BR}} > BT$ then
        add $L_{Mi}$ to category 2
    end
    else if $LM_{i_{CR}} < CT$ and $LM_{i_{BR}} < BT$ then
        add $L_{Mi}$ to category 3
    end
    else if $LM_{i_{CR}} > CT$ and $LM_{i_{BR}} < BT$ then
        add $L_{Mi}$ to category 4
    end
end
Concatenate all categories to make Prioritized list ($PL$) of modules
for $i = 0$ to $PL.size$ do
    $d = compare(LF_d, PL[i])$;
    if $d! = -1$ then
        $Taskmap.insert(LF_d[d], PL[i])$;
        update Cap($LF_d[d]$);
    end
    else
        place all remaining task to cloud :
    end
    return modulemap;
end
end
```

Algorithm 1: ModuleScheduler

fog devices to the Cloud with maximum utilization of fog devices by assigning priority to modules based on their computation requirement and bandwidth usage improves the execution time with reducing cost and network usage.

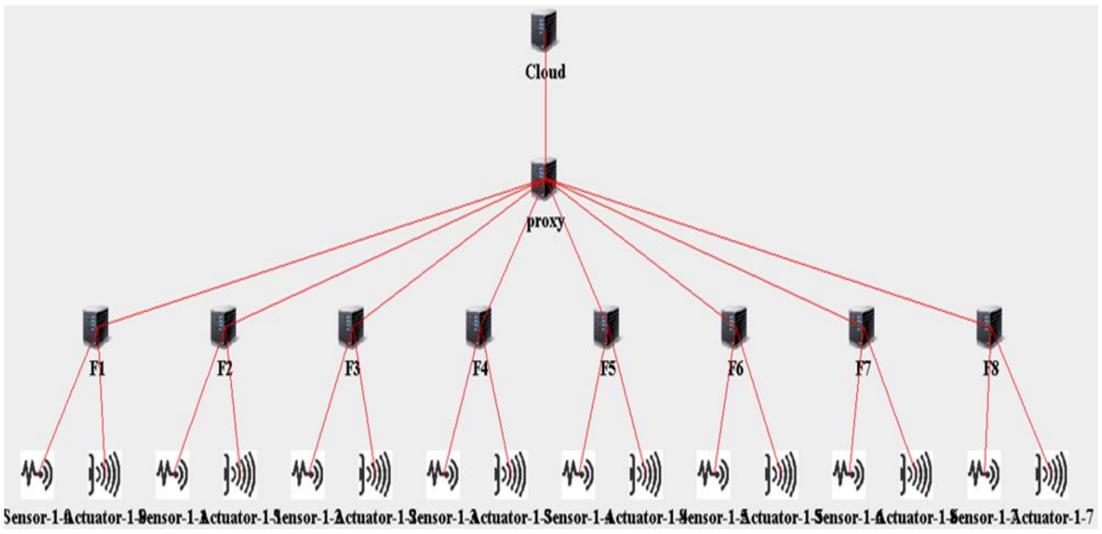
Cloud
proxy
F1
F2
F3
F4
F5
F6
F7
F8
Sensor-1-Actuator-1-Sensor-1-Actuator-1-Sensor-1-Actuator-1-Sensor-1-Actuator-1-Sensor-1-Actuator-1-Sensor-1-Actuator-1-Sensor-1-Actuator-1-Sensor-1-Actuator-1-7

FIGURE 3.6: One of the Network Topology used in Simulation [90]

3.9 Evaluation and Analysis of Results

To determine the efficacy of the proposed approach, *iFogSim* [90] simulates a fog system. The CloudSim architecture [115] is the foundation of *iFogSim*, widely used to model many computing environments [116, 117]. Table 3.2 shows the simulation parameters. Three distinct network topologies and workloads are used to create this scenario's variations. Figure 3.6 is a graphic representation of one of these topologies, as produced by *iFogSim*. The workload is defined as the number of applications with different modules for the placement at Cloud fog architecture. The *MI's* of application modules vary from the $50MI$ to $700MI$ as shown in Table 3.2. The applications may be compute-intensive or bandwidth intensive. For experiments we have the datasets of three different applications Application1 (A_1) mixed (Compute bandwidth intensive), Application2 (A_2) compute-intensive, and Application3 (A_3) bandwidth intensive. With these applications, we run the

```
Input: i) List of Fog devices ($LF_d$ with their Processing Power MIPS)
ii) Modules $M_i$ from list of modules $LM$
Output: Selected Fog Device for mapping
COMPARE List of Fog devices $LF_d$, $PL(M_i)$
begin:
while true do
    if isEmpty($LF_d$) then
        reurn -1
    end
    else
        for $i = 0$ to $LF_d.size$ do
            $SF_d$[i]=device with minimum execution time
            if $SF_d[i]cpu >$ $PL(M_i)$ then
                return i
            end
            else
                remove $SF_d$ from $LF_d$
            end
            $LF_d$=updated $LF_d$
            if isEmpty($LF_d$) then
                return -1
            end
        end
    end
end
end
```

Algorithm 2: CompareModule

simulation for three different scenarios and analyze the performance of the *RACE* algorithm in the context of execution time, bandwidth consumption, and cost for each application.

Scenario 1: The bandwidth, cost and execution time of applications when placed only at the Cloud.

Scenario 2: What will be the effect on execution time, bandwidth usage, and cost when only fog devices are part of the simulation.

Scenario 3: Applications are placed in the Cloud fog environment, by placing application modules from fog devices to the Cloud.

TABLE 3.2: Experimental Setup for the Simulation of RACE

Number of Applications = 3			
Application Type	Mixed(Compute + Bandwidth Intensive) (Application-1)	Compute Intensive (Application-2)	Bandwidth Intensive (Application-3)
No of Modules	10	15	15
MI in modules	20-200	40-500	100-900
No of Fog Devices	10		
CPU(MIPS) of Fog Devices	200-3000		
CPU(MIPS) of Cloud	42800		

We make use of *iFogSim* [90], which is inspired from CloudSim [115]. The suggested approach's efficacy was evaluated with the use of this open-source framework, which models and analyses the performance of Cloud fog environments and services. The Intel Core i3-4030U Quad-core CPU (1.9 GHz clock speed) and two gigabytes of main memory are used in this research for experimentation. Table 3.2 illustrates the configuration detail for the employed simulation environment. In this work, we evaluate RACE's efficacy in comparison to that of the classic Cloud placement and Taneja. et al. [24].

3.10 Simulation Results

Execution time, bandwidth utilization, and the cost of Cloud resources are evaluated between the suggested fog-Cloud placement strategy (*RACE* Scheduling Algorithm) and the standard Cloud-based placement approach. Figures 3.7 to 3.15 show the simulation results, which show that the suggested placement technique has a positive effect on execution time, bandwidth utilization, and cost across all three network topologies.

The results of the simulation, Figures 3.7 to 3.15 show an immensely favorable impact on the monetary cost when using Cloud resources, execution time and bandwidth in the proposed positioning approach on all the three network topologies used. Figure 3.7 depicts that in the case of Application-1 (A_1), the *RACE(CFP)* improves the execution time to 20%, 8%, and 3% compared to the traditional

Cloud placement or Cloud only placement *(COP)*, the *RACE(FOP)*, and the algorithm proposed by Taneja. et al. [24], respectively. The *RACE(FOP)* and Taneja. et al. [24] also improved the execution time of A_1 to 12% and 17%, respectively, compared with the *COP* strategy. It can be seen that for A_1, the *RACE(CFP)* outperforms in comparison with all three placement strategies.

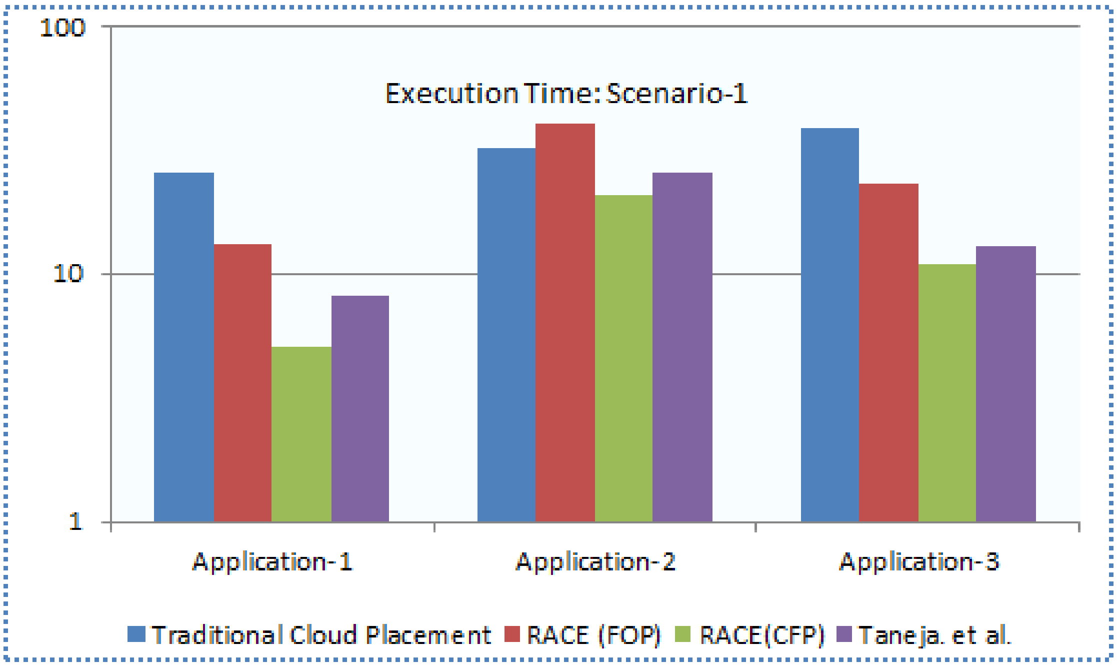

FIGURE 3.7: Execution Time of Three Applications Under Network Topology of 4 FDs

For Application-2 (A_2), the compute-intensive application, although the execution time of *COP* is comparatively less than *RACE(FOP)* because all the placements are at the CLOUD, which has high processing resources but still the *RACE(CFP)* improves the execution time to 12% and 20% as compared to the *COP* and *RACE(FOP)* respectively. The *RACE(CFP)* also improves the execution time to 5% compared to the Taneja. et al. [24]. For Application-3 (A_3), the bandwidth-intensive application, the *RACE(FOP)*, has better execution time than *COP* because the modules in A_3 are less compute-intensive, and their placement at the fog layer has little impact on their execution time while placing them in the Cloud.

In fog to cloud placement of modules, our proposed *RACE(CFP)* algorithm performed better. The *RACE(CFP)* strategy improved the execution time by 15%, 12%, and 3%, respectively, compared to the *COP*, *RACE(FOP)*, and Taneja. et al. [24]. The *RACE(FOP)* and Taneja. et al. [24] also improved the execution time to 15% and 26% respectively for A_3 compared to the *COP*.

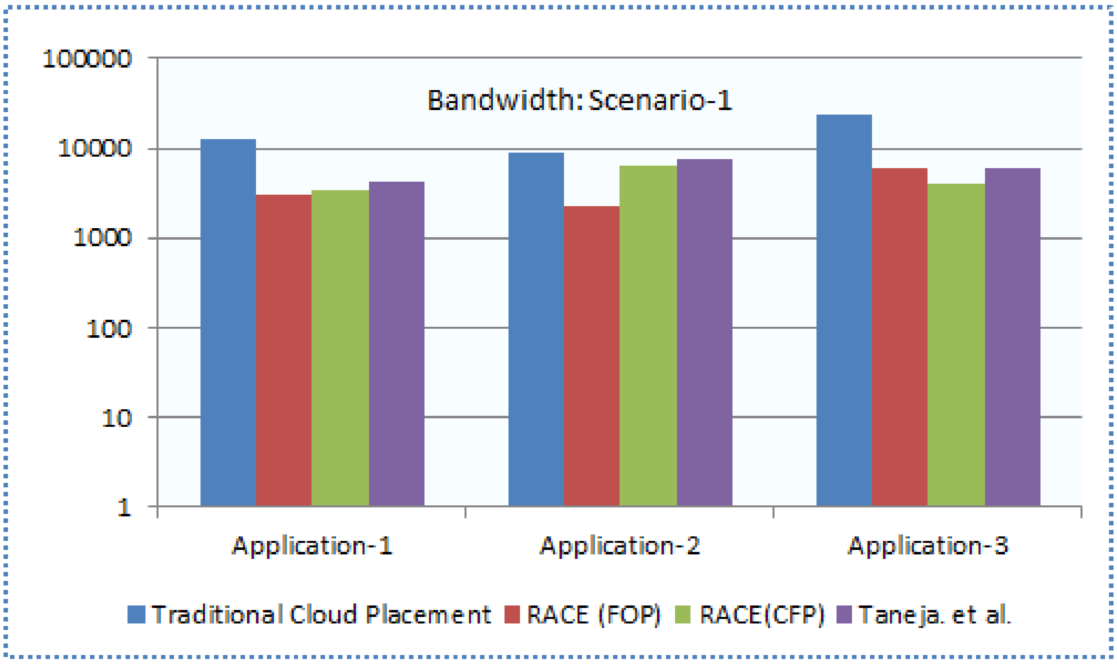

FIGURE 3.8: Bandwidth Consumption of Three Applications Under Network Topology of 4 FDs

The results shown in Figure 3.7 concludes that the *RACE(CFP)* has better results in execution time for all three applications in comparison to the *COP*, *RACE(FOP)* and Taneja. et al. [24] The bandwidth consumption of the three applications when using the network topology having four fog devices is shown in Figure 3.8. For A_1, the mixed application w.r.t computation and bandwidth requirement, the *COP* have the maximum bandwidth consumption as all the modules are placed at the Cloud layer causing maximum bandwidth utilization.

In the case of *RACE(FOP)*, as all modules are placed at the fog layer, the very one layer to the user, the bandwidth consumption reduces to 40% as compared to the *COP*. The *RACE(CFP)* also reduces the bandwidth consumption to 38% when compared with the*COP*. Similarly, the Taneja. et al.[24] also improves the bandwidth consumption to 34% compared to the *COP*. For A_2, the compute-intensive application, the *COP* has the maximum bandwidth consumption of 37% as all the modules are placed at the Cloud layer.

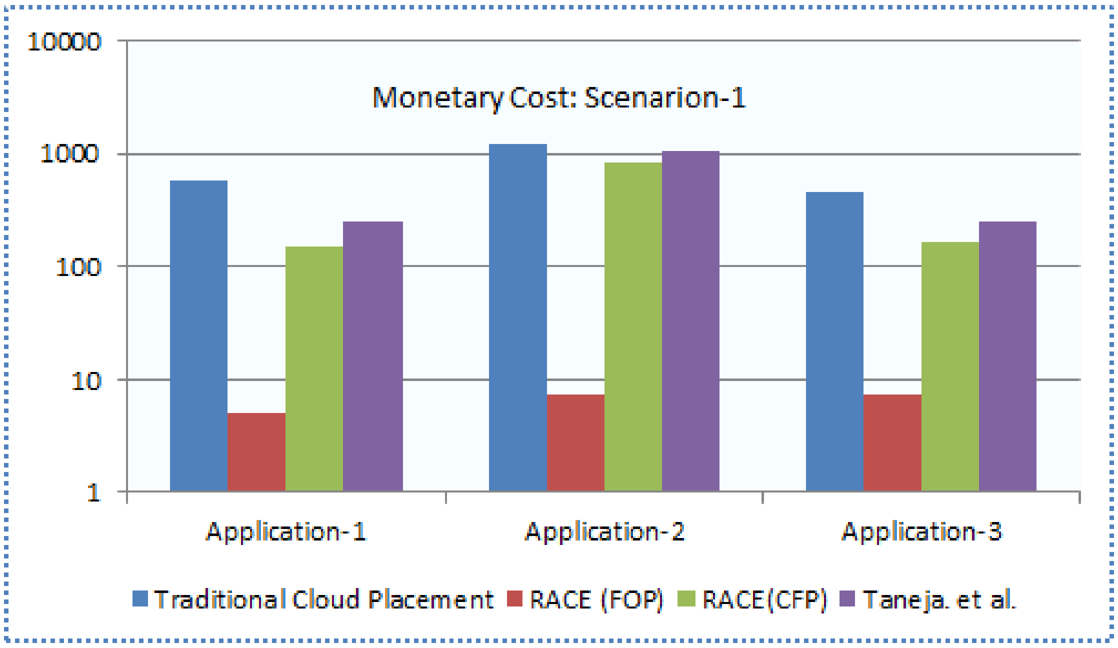

FIGURE 3.9: Monetary Cost of Three Applications Under Network Topology of 4 FDs

In the case of *RACE(FOP)*, as the modules are placed only at the Fog layer has only 9% of bandwidth consumption.The *RACE(CFP)* and the Taneja. et al. [24] has a bandwidth consumption of 25% and 29%, respectively, as both approaches use the Cloud resources and the Fog layer. For A_3, the bandwidth-intensive application, the *COP* has the maximum bandwidth consumption of 60% as all the modules are placed at the Cloud layer. The *RACE(CFP)* strategy improves the bandwidth consumption to almost 5% as compared to the Taneja. et al. [24].

It can be seen that the overall performance of *RACE(CFP)* is better than all the approaches except the *RACE(FOP)*. The monetary cost of using Cloud resources for the three applications is shown in Figure 3.9. It can be seen that *RACE(FOP)* has a lower monetary cost than *RACE(CFP)*, *COP*, and Taneja et al. [24] because the *RACE(FOP)* approach uses all fog layer resources, which are less expensive than cloud resources.

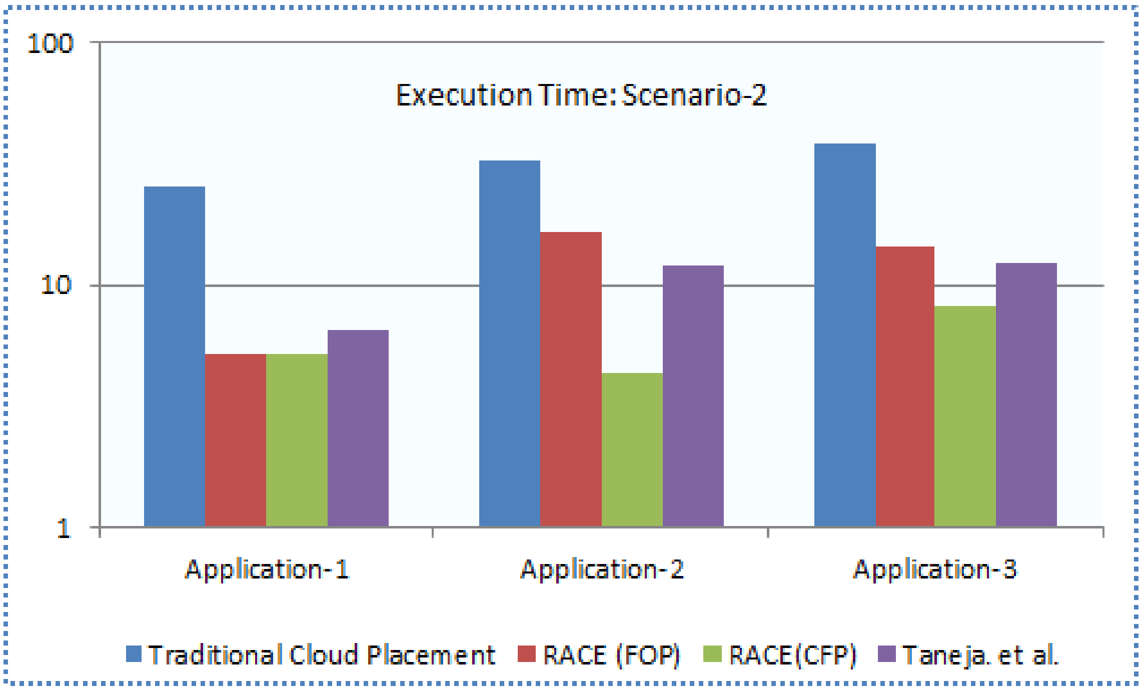

FIGURE 3.10: Execution Time of Three Applications Under Network Topology of 8 FDs

For A_1, when using the *COP* strategy, the monetary cost reaches 58%, whereas the Taneja. et al.[24], and *RACE(CFP)* reduces the cost to 25% and 16%, respectively. Similarly, for A_2, the *COP* has the maximum cost of 38% compared with the Taneja. et al.[24], which reduces the cost to 34%. The *RACE(CFP)* has better results by reducing the cost to the 28%. For A_3, the bandwidth intensive application the *COP* has a cost of 52%, which reduces to 28% by Taneja. et al.[24].

For the *RACE(CFP)*, as the application modules are placed from fog to the Cloud by considering their intensity, the approach reduces the cost to 19%. It can be seen in Figure 3.9 that although *RACE(FOP)* has less cost for all three applications, the *RACE(CFP)* also reduces the cost than *COP* and Taneja. et al.[24] when considering the Cloud fog layer for scheduling application modules in a Cloud fog environment. Figure 3.10 depicts the execution time of the three applications when using the eight fog devices. It can be seen that the execution time of A_1

reduces by the three approaches, i.e., *RACE(CFP)*, *RACE(FOP)*, and Taneja.et al.[24]

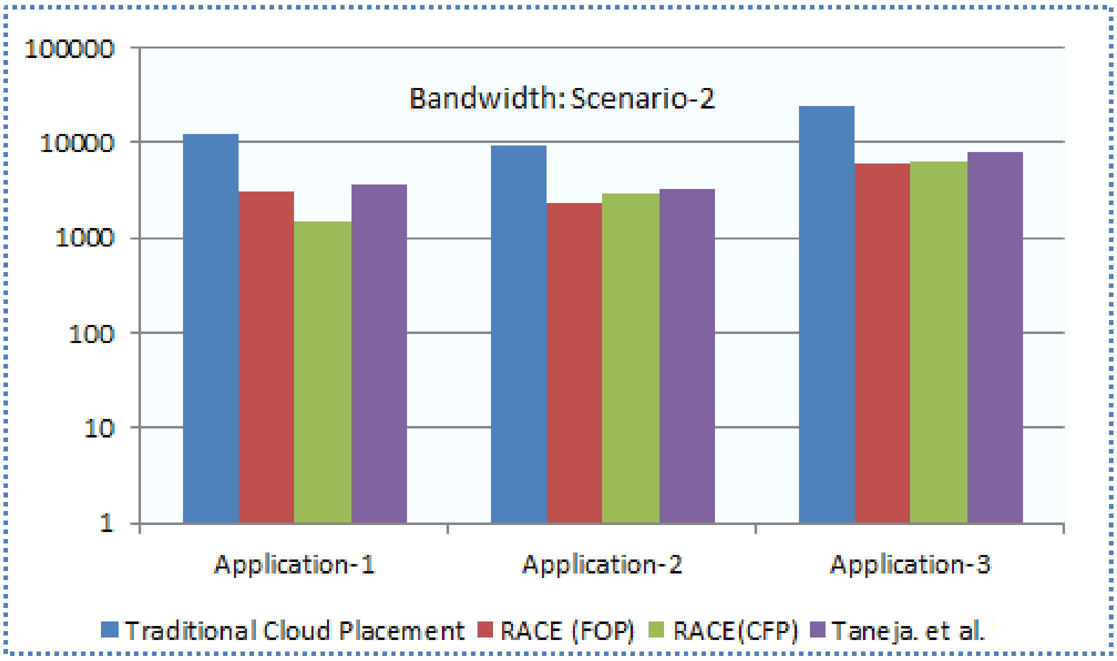

Figure 3.11: Bandwidth Consumption of Three Applications Under Network Topology of 8 FDs

because increasing the number of fog devices ensures that maximum modules can be accommodated at the fog layer. For the execution time of A_1 the *RACE(CFP)* and *RACE(FOP)* have almost the same results. However, the execution time improves from the *COP* and Taneja. et al.[24] by 5% and 48%, respectively.

For A_2, the compute-intensive applications the *RACE(CFP)* improves the execution time by 43% from *COP*, 18% from *RACE(FOP)* and 11% from Taneja. et al.[24]. It can be seen that an increase in the number of fog devices affects the execution time for all approaches. However, our scheduling policy has overall good performance from *COP*, *RACE(FOP)*, and Taneja. et al.[24].

Similarly, the increase in the number of fog devices also affects the execution time of A_3, as seen in Figure 3.10. The*RACE(CFP)* improves the execution time by 41%, 9% and 6% from *COP*, *RACE(FOP)* and Taneja.et al.[24] respectively. Figure 3.11 presents the bandwidth consumption for three applications with network topology of eight fog devices. It can be found in the figure that in the case of *COP*, the bandwidth consumption for all three applications is higher as all the modules are placed in the Cloud, which increases the bandwidth consumption. For A_1, *RACE(FOP)* improves the bandwidth by 45% and 3% compared to *COP* and Taneja. et al.[24].

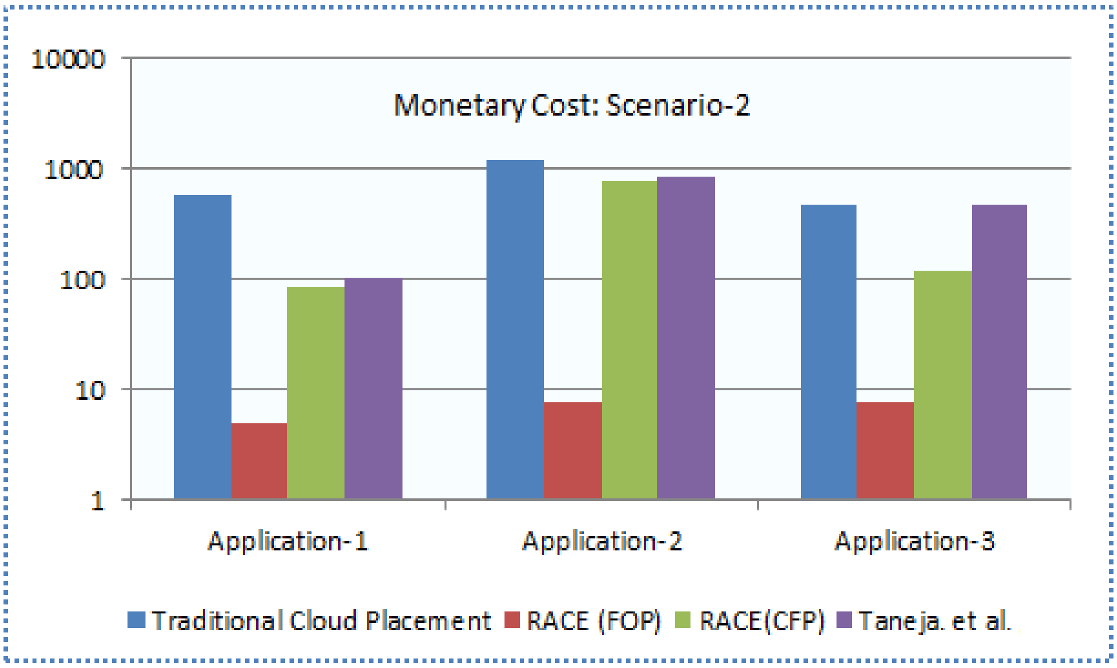

FIGURE 3.12: Monetary Cost of Three Applications Under Network Topology of 8 FDs

For A_2 the *RACE(CFP)* improves the bandwidth consumption by 35% and 3% in comparison to the *COP* and Taneja. et al.[24]. Compared to the *RACE(CFP)*, the *RACE(FOP)* has 4% better results because all modules are placed at the fog layer. For A_3, the bandwidth-intensive application the *RACE(FOP)* has less bandwidth consumption from all three approaches i.e., *RACE(CFP)*, *COP* and Taneja. et al. [24]. The *RACE(CFP)* has consumed 37% and 3% less bandwidth as compared to *COP* and Taneja. et al.[24]. Figure 3.12 presents the monetary cost of using Cloud resources for the three different applications, i.e., A_1, A_2, and A_3 having the network topology of eight fog devices.

The figure shows that *RACE(FOP)* performs well for A_1, A_2, and A_3 as all modules are placed at the fog layer, decreasing the Cloud resources cost. For A_1, the

RACE(CFP) has better performance with less monetary cost of 63% and 4% as compared to the *COP* and Taneja.et al.[24]. Similarly, it can be seen in Figure 3.12 that the *RACE(CFP)* has 15% and 4% less monetary cost for A_2 when compared with the *COP* and Taneja.et al.[24]. For A_3 the *RACE(CFP)* also has 33% and 23% less monetary cost from the *RACE(CFP)* and Taneja.et al.[24].

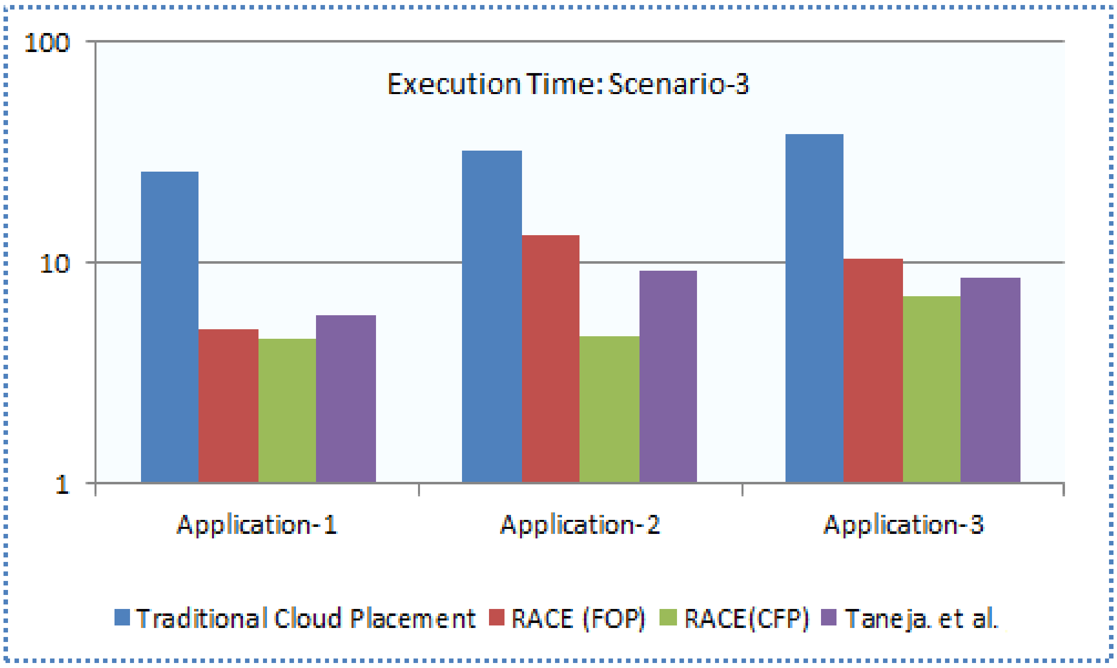

FIGURE 3.13: Execution Time of Three Applications Under Network Topology of 10 FDs

Figures 3.13 to 3.15 display the simulation results of execution time, bandwidth consumed, and the Cloud resources cost for three applications A_1, A_2, A_3, while using the network topology of ten fog devices. The favorable impact of *RACE(CFP)* can be seen in the execution time of three applications in Figure 3.13. As the number of fog devices increases, the execution times of all three applications improved. For A_1 the *RACE(CFP)* reduces the execution time to 1%, 3% and 52% respectively as compared to the *RACE(FOP)*, *COP*, and Taneja. et al.[24].

As the *RACE(CFP)* uses both the fog and the Cloud layer for processing the execution time for A_2, improves with 12%, 5%, and 46% respectively when compared with the *RACE(FOP)*, *COP*, and Taneja. et al.[24]. The same improvement can be seen with A_3, the *RACE(CFP)* reduces the execution time with 5%, 2%, and 49% respectively in comparison to the *RACE(FOP)*, *COP*, and Taneja. et al.[24].

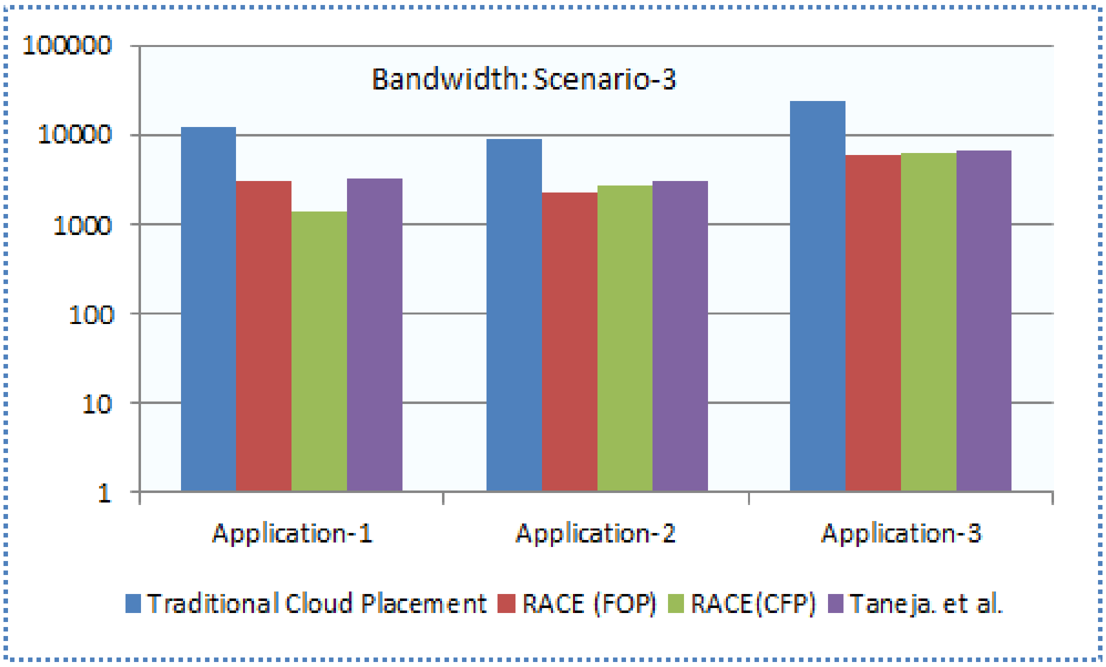

FIGURE 3.14: Bandwidth Consumption of Three Applications Under Network Topology of 10 FDs

Figure 3.14 shows the bandwidth consumption of three applications when using the network topology of ten fog devices. For A_1 the bandwidth consumption when using the *COP* is higher as 61% because all modules are placed at the *CLOUD*. The *RACE(FOP)* has better performance than the *COP* and Taneja. et al.[24] with 15% bandwidth consumption improving 46% and 2%, respectively. For A_2, the *RACE(CFP)* consumes only 3% more bandwidth as compared to the *RACE(FOP)*; however, it improves the bandwidth consumption as 2% and 37% from *COP* and Taneja. et al.[24]. The same results can be seen for A_3 for which the *RACE(CFP)* and *RACE(FOP)* have almost the same bandwidth consumption with an improvement of 41% and 2% from *COP* and Taneja. et al.[24].

The graph in Figure 3.15 displays the monetary cost of using Cloud resources. It can be observed in the Figure 3.15 that the *RACE(FOP)* has better results for all three applications because all modules are placed at the fog layer, which reduces the overall cost. For A_1 the *RACE(CFP)* reduces the cost to 51% and 9% from *COP* and Taneja. et al.[24].

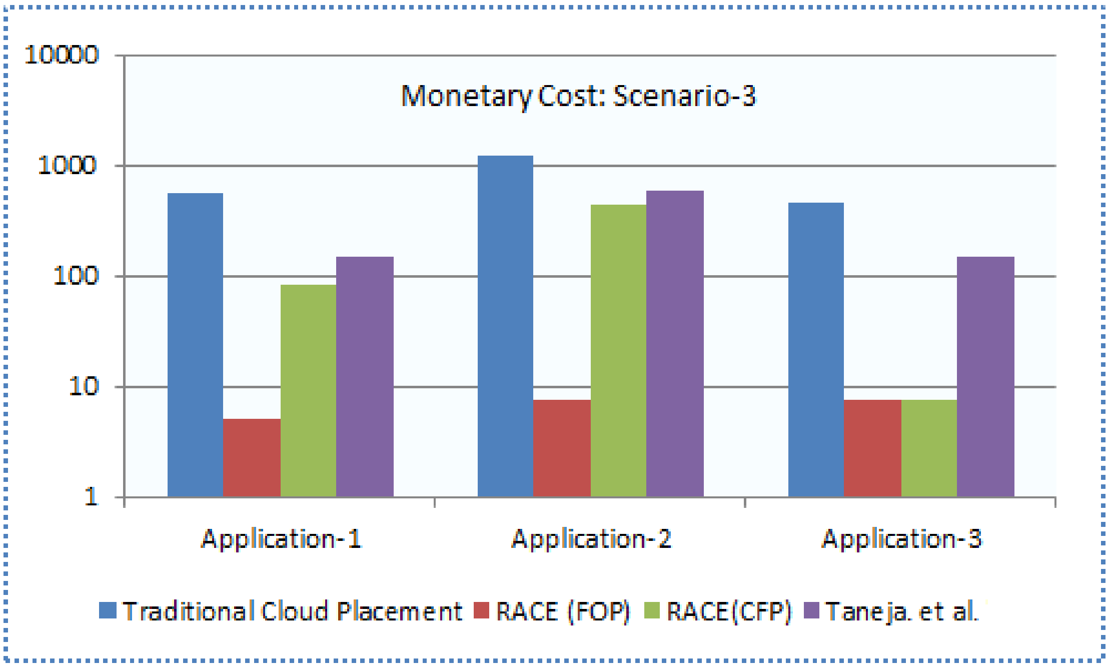

FIGURE 3.15: Monetary Cost of Three Applications Under Network Topology of 10 FDs

For A_2 the same improvement can be seen with *RACE(CFP)* as 34% and 6% less cost as compared to the *COP* and Taneja. et al.[24]. For A_3, the bandwidth-intensive application, the *RACE(CFP)* has almost the same results as *RACE(FOP)* due to increase in the number of fog devices. The *RACE(CFP)* improves the monetary cost by 23% and 73% from the *COP* and Taneja. et al.[24].

3.11 Conclusions

Fog computing has great potential to aid several *IoT* applications. In this research, we introduced a Resource Aware Scheduler to optimize the deployment of application components across a three-tier Fog Cloud infrastructure. The simulation results by using the *iFogSim* simulator show that the proposed algorithm has better results from Taneja. et al. [24] and the traditional Cloud placement for the bandwidth usage, execution time, and the monetary cost performance. Incoming applications and their associated modules characterize the workload. We have put the *RACE* algorithm through its paces using three distinct approaches: placing workload in the Cloud, in the fog layer, and from the fog layer into the Cloud. Whether measured by execution time, bandwidth, or cost, the *RACE(CFP)* performs well across the board. The *RACE(FOP)* performs better with higher numbers of fog devices in some cases. Still, in general, the *RACE(CFP)* has a better execution time, bandwidth consumption, and monetary cost performance.

Chapter 4

Genetic Algorithm based Improved Resource-Aware Cost Efficient Scheduler for Fog Environment

4.1 Introduction

The *IoT* is the networked interconnection of everyday items, including computers, farms, industries, and vehicles [118–120]. The *IoT* has made everyday items "smart" or capable of sensing, processing, and communicating efficiently via a network to carry out essential activities independently of human intervention [121, 122]. With the rapid development of *IoT*, the number of devices connecting to the network is rapidly increasing. According to a report from D. Manyika et al. [123], by 2025, the *IoT* is expected to have a theoretical impact of 11 trillion per year, reflecting 11 percent of the global economy. According to a press release in 2015 [103], 13.4 billion connected devices to the internet already outnumbered humans on the earth. A new study from Juniper Research predicts 38.5 billion connected *IoT* devices by 2020, a 285% increase from 2015 [3], which is a staggering figure. In Cisco's annual internet report, the number of devices and connections connected to the internet is expected to increase from 18.4 billion in 2018 to 30 billion by 2023 [124].

As *IoT* devices grow, a massive amount of data will generate, also known as Big data. Several data centers have been configured as a Cloud to accommodate the enormous amount of Big data generated by *IoT* devices. In the tradition of Cloud Computing, data service subscribers are projected to be able to benefit from efficient and flexible services [4]. There are two main actors in traditional Cloud computing, i.e., Cloud Service Providers *(CSP)* and Cloud users or customers [125]. Users lease the *CSP's* resources on a pay-per-use basis. The *CSP* acquires Cloud resources, such as storage, processing, and so on, and make those resources available to Cloud users [126]. Customer tasks are received by the Cloud task manager in the form of cloudlets and are passed to the task scheduler [21].

These traditional Clouds are expected to provide data service subscribers with flexible and efficient Cloud computing [4]. The transferring of data between the end-user and the Cloud requires high bandwidth for processing, causing delays [106]. Moreover, if some geo-spatially distributed and time-sensitive applications, such as smart healthcare monitoring, virtual reality, smart traffic surveillance, etc. send their requests for processing to the remote Cloud, may impact the quality of service due to the delay caused by the resource bottlenecks and the bandwidth constraints [58, 127, 128]. Many approaches such as Mobile computing, Edge computing, and Fog computing have been proposed to overcome this limitation by bringing computation and storage services closer to the end-user application [32, 42]. Recently, Fog computing has received the most attention among the various proposed approaches.

Fog computing is a new concept introduced by CISCO [6]. It is an extension of Cloud computing which provides the services like storage and processing to the *IoT* users at the edge of the network to reduce network congestion and latency [90]. In contrast to specialized computing facilities such as data centers, Fog computing uses a vast number of locally distributed fog servers, which can be smartphones, intelligent gateways, switches, routers, access points, and cellular base stations with limited aptitudes in terms of storage, and processing power [23, 25, 129]. The Sense Process Actuate Model *(SPAM)* is commonly used in Fog computing to detect and collect data through sensors and then send the collected data to fog devices for processing. Fog nodes can send data from the fog layer to the Cloud, where it is stored and processed for long-term analytic. The data transferred from the fog layer to the Cloud for processing reduces the task's execution time, but it may increase the bandwidth and the monetary cost when using the Cloud resources [21].

Nonetheless, real-time applications need a quicker reaction time than delay-tolerant ones. As a result, these applications must compete for limited resources. One of the biggest struggles of Fog computing is managing the limited resources and time sensitive applications [130]. When jobs are scheduled efficiently, they can produce timely and accurate responses, which are important for any smart application. A patient's medical status, for instance, must be reported quickly in a smart healthcare system if the patient is to have any chance of survival.

Similarly, in an innovative home application, the security surveillance application must report immediately if any suspicious activity happens [128]. Therefore, some efficient job scheduling algorithm is needed to make optimal use of these resource-constrained and heterogeneous fog devices [88]. In this work, we propose a *GA*-based optimized policy for placing application modules in a Cloud fog environment, which minimizes the execution time of applications with optimized usage of bandwidth and the application's monetary cost. The contributions of the paper are summarized as follows:

An optimized scheduling technique is proposed for mapping application modules in a Cloud Fog computing environment. The proposed solution is leveraged by a genetic algorithm to evolve the solution using the applications' execution time. Initially, the proposed approach takes into account application and fog device properties to encode the chromosome-based representation of the problem. Then, genetic algorithm-based mutation and crossover operators are employed to evolve the solutions using multiple variations of the parameters. The evolved solution has been utilized to schedule the application modules, using the *iFogSim* simulator to validate the results. Experimental results are also compared with baseline and state-of-the-art algorithms for three application corpus.

4.2 Architectural Model

Fog computing is a new computing paradigm that enables storage, processing, and communication at the edge of the network. Fog computing can use Cloud data-centers to process and store large-scale applications [6, 131, 132].

Figure 4.1 shows the fog architecture, in collaboration with end-users and Cloud, developing a three-layer hierarchical, bi-directional design [108]. In the Cloud fog architecture, the top layer is made up of the Cloud layer, with fog devices acting as an intermediate layer in the middle. The bottom-most layer or end layer comprises end-users with sensors or actuators.

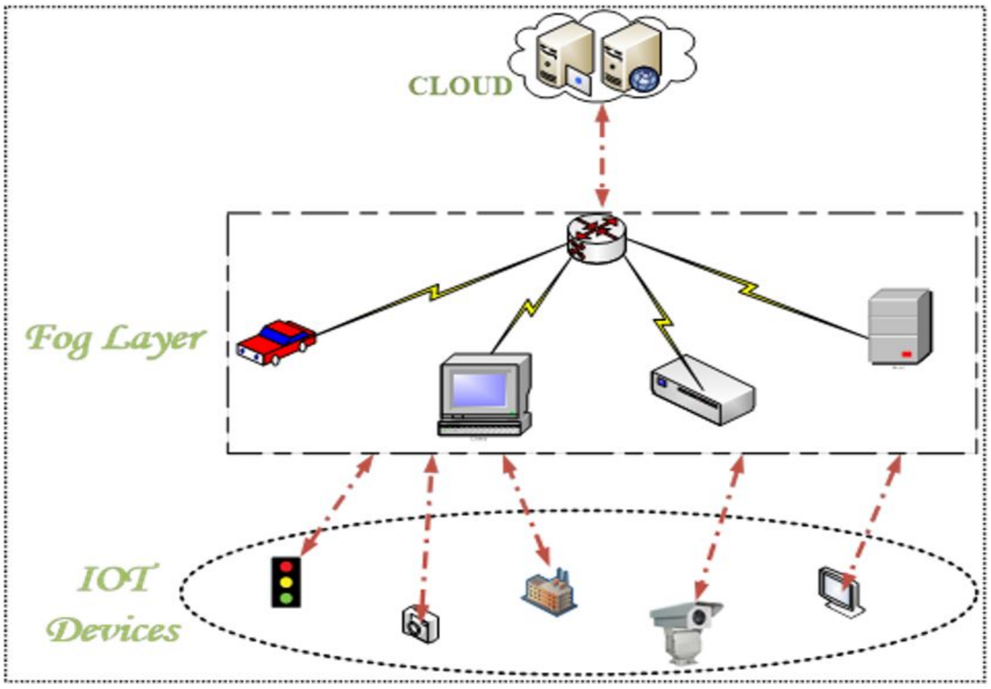

FIGURE 4.1: The Three Layer Architecture of Cloud Fog Computing Environment

4.2.1 End Devices/End Layer/User Layer

The user layer is the closest to the end-user, with many heterogeneous and mobile devices. These devices consist of *IoT* devices like sensors (temperature sensors, heartbeat sensors, GPS sensors, etc.), smart cards, mobile phones, cameras, connected cars, and laptops. Data from these external devices is converted into signals and sent to fog nodes for processing. End devices are frequently battery-powered and relatively restricted in CPU and memory, resulting in limited battery life. End devices often generate data that must be processed and stored elsewhere because the end device's processing or storage capacity is insufficient. In some cases, end devices require data from a different source. As a result, end devices are frequently connected to a network. However, uninterrupted connectivity may not always be possible.

4.2.2 Fog Layer

The computational resources present at the fog layer near the network's edge are referred to as fog nodes. Usually, the fog layer is made up of devices like routers/switches, proxy servers and cellular base stations that are incapable of doing extensive computations or storing large amounts of data. Fog devices may be resource-poor in memory and computation power like access points and routers [110], or they can be resource-rich like cloudlets [23]. Existing network equipment, such as gateways, routers, and switches, can operate as fog nodes if they have adequate free resources.

4.2.3 Cloud Layer

The Cloud layer is the topmost layer of the Cloud fog architecture with multiple high-speed servers having massive storage capacity, which provides various application and storage services, as shown in Figure 4.1. The Cloud layer of the Cloud fog environment resources is used for applications that need a lot of processing power and storage space.

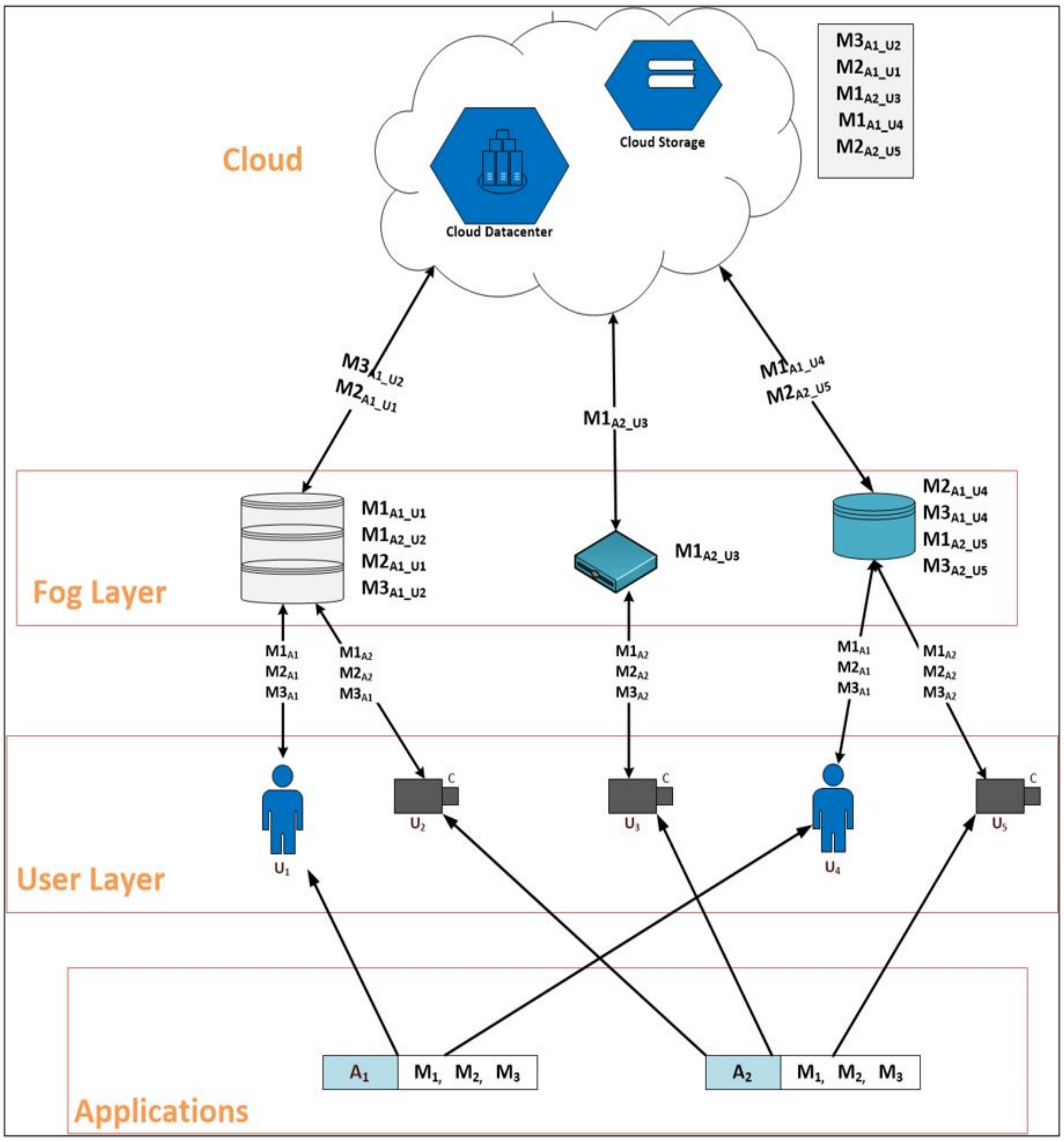

FIGURE 4.2: Example on Scheduling of Application Modules in Cloud Fog Environment

4.3 Scheduling in Fog Computing

Scheduling refers to arranging application modules onto computing resources for better utilization of resources such as CPU, memory, bandwidth, etc. Application modules may be scheduled everywhere, from the fog to the Cloud, owing to the design of the Fog computing environment according to module requirements with the following combinations [85, 86, 133]

Figure 4.2 shows an example of a scheduling problem in the fog environment. We consider five users (U_1, U_2, U_3, U_4, and U_5) at the user layer who want to execute their applications at the fog layer. Each user has two applications, either A_1 or A_2, comprising three modules (M_1, M_2, and M_3). Based on the defined scheduling policies, the scheduler decides the mapping of modules in the Cloud fog environment. As shown in Figure 4.2, modules M_1, M_2 of A_1 for U_1 and M_3 of A_1 and M_1 of A_2 for U_2 are selected for execution at the fog layer, while M_2 of A_1 for U_1 and M_3 of A_1 for U_2 are mapped at the Cloud layer for execution.

Similarly a fog device in the range of U_3 has successfully mapped all application modules (M_1, M_2, and M_3). For U_4 and U_5, modules M_2 of A_1, M_3 of A_1 for U_4 and M_1, M_3 of A_2 for U_5 are mapped at fog layer while M_1 of A_1 for U_4 and M_2 of A_2 for U_5 are placed at Cloud. The abbreviations used in the paper are listed in Table 4.1. The rest of the chapter is organized as follows:
The problem formulation is described in section 4.4. The design and implementation of the problem are covered in section 4.5. Section 4.6 discusses simulation results, and section 4.7 concludes the chapter with future work.

1. The modules are offloaded from end devices to the fog and Cloud layers when all three layers are involved according to the modules' requirements.
2. Only the fog layer is involved by offloading the modules from end devices to the fog nodes.

4.4 Problem Formulation

Fog computing is the concept of utilizing fog node services for end devices that help them to overcome capacity constraints. When the device's generated data exceeds its capacity, it can connect to a fog node, pass data to the fog node, and

TABLE 4.1: GA-IRACE System Preliminary Notations

Notations	**Descriptions**
GA-IRACE	Improved Resource Aware Cost-Efficient Scheduler with Genetic Algorithm
MI	Million of Instruction
MIPS	Million of Instruction per second
M	List of Modules
F	List of Fog Devices
IOT	Internet of Things
CT	Computation Threshold
BT	Bandwidth Threshold
CR	Computation Requirement
BR	Bandwidth Requirement
CP	Computation Power
ST	Start Time
EFT	Estimated Finish Time

offload data to the fog node for processing. Since the network transfers between the fog node and the end devices have minimal latency and the fog node's have the relatively high processing power, it allows the entire process to be quick, enabling real-time interactions.

However, the offloading of tasks from the fog to the Cloud layer and the efficient scheduling at the fog layer becomes critical for latency-sensitive applications like traffic control, augmented reality and gaming. In case when all the application modules are accommodated at the fog layer, it may reduce the monetary cost and bandwidth of the application modules. Still, it may increase the application's execution time due to the limited power of the fog devices not being affordable for the time-sensitive applications. On the other hand, when data is transferred from the fog to the Cloud layer for processing, it may reduce the task's execution time, increasing the bandwidth and the monetary cost for applications [21, 134]. Therefore we are essentially concerned with formulating the goal of scheduling in a cloud fog environment that can reduce the application execution time while optimizing the bandwidth and monetary cost.

The entire set of resources consists of fog devices *(FDs)* and Cloud resources

(CLOUD). The *CPU, RAM, and Bandwidth* of a node in a network are standard metrics used to characterize its resource capability. Modules *(M_i)* of applications may differ in terms of their computational and bandwidth requirements, depending on how they perform computations and use bandwidth [30, 90, 113, 114]. Therefore, if M_i represents the application module with its computation requirement as CR and bandwidth requirement as BR then it can be represented as follows:

Let $M = \{M_1, M_2, M_3 .... M_n\}$ be the set of modules of user input jobs with their computation and bandwidth requirement as follows:

Let $C = \{CR_{M1}, CR_{M2}, CR_{M3} .... CR_{Mn}\}$ be the computation requirement of each module in module set M.

Let $B = \{BR_{M1}, BR_{M2}, BR_{M3} .... BR_{Mn}\}$ be the bandwidth requirement of each module in module set M.

Let $F = \{FD_1, FD_2, FD_3 .... FD_n\}$ be the list of fog devices.

$$CT = median(C) \tag{4.1}$$

$$BT = median(B) \tag{4.2}$$

The computation threshold CT and bandwidth threshold BT are represented in equations 4.1 and 4.2.

$$\begin{matrix} \forall M_i \\ \exists M_i \in M \end{matrix} \left\{ \begin{matrix} CR_{Mi} > CT \\ BR_{Mi} > BT \end{matrix} \right\} \Longrightarrow M_i \rightarrow Cloud \tag{4.3}$$

Alternatively: $M_i \rightarrow M^*$

Because the modules may be computationally or bandwidth-intensive, we begin our module mapping by categorizing them, as shown in equation 4.3. The modules with more extensive computation requirements than the computation threshold are directly sent towards the Cloud for required operation, whereas the remaining modules are stored in the list M^* for optimized scheduling. Equation 4.4 shows the actual start time of the module M_i on the fog device FD_j.

In equation 4.4 $ST_{i,j}$ shows the start time of the module. The modules executed before M_i on the fog device FD_j are denoted by $pred(M_i)$ .The module M_i cannot start its execution until its predecessor completes its execution. If more than one module needs to be executed simultaneously on same device i.e., FD_j we select a task according to the order in which modules are scheduled. The execution time of the module M_i on the scheduled fog device can be calculated as shown in equation 4.5.

$$ST_{i,j} = \begin{cases} 0; \quad pred(M_i) = 0 \\ max \\ k \in pred(M_i) \quad \left\{ET_{j,k}(k \to j)\right\} \\ \qquad if(pred(M_i) \neq 0) \end{cases} \tag{4.4}$$

$$ET_{(i,j)} = \frac{MI(M_i)}{MIPS(FD_j)}, \qquad \forall\, M_i \in M, \quad \forall\, FD_j \in F \tag{4.5}$$

The estimated finish time EFT of a module M_i scheduled on a fog device FD_j can be calculated in equation 4.6.

$$EFT_{i,j} = ST_{i,j} + ET_{i,j} \tag{4.6}$$

Let P_n be the processing node, either fog or the Cloud, where the modules are scheduled for processing. The cost *Cost*(M_i,P_n) is the monetary cost of executing the application modules M_i when scheduled on P_n is computed as shown in the following equation 4.7.

$$Cost(M_i, P_n) = \begin{cases} C_p(M_i, P_n) + C_s(M_i, P_n) + C_{cm}(M_i, P_n) + \\ \sum_{P_n \in N_{cloud}} C_{cm}(M_i, P_n) \qquad if(P_n \in N_{cloud}) \\ \qquad otherwise \\ \sum_{P_n \in N_{fog}} C_{cm}(M_i, P_n) \qquad if(P_n \in N_{fog}) \end{cases} \tag{4.7}$$

The processing cost in equation 4.7 is calculated as:

$$C_p M_i, P_n) = c_1 * T_i; \quad T_i = A(M_i, P_n) \tag{4.8}$$

TABLE 4.2: GA-IRACE Parameter Setting

Parameters	Values
Number of population	100
Population Size	50
Reproduction Operator	Crossover and Mutation
Crossover type	Single point
Mutation rate	1%, 3%, 5%
Stopping criteria	100 iterations

Where $c_1$is the processing cost per time unit of the application module on the node P_n. The communication cost for sending outgoing data from the Cloud node P_n can be calculated using equation 4.9 in which c_2 shows the price per unit of data.

$$C_{cm}(M_i, P_n) = c_2 * \left(\sum_{M_i \in Ex(P_n)}^{M_j \in pred(M_i)} C_i \right) \tag{4.9}$$

The overall cost of an application can be computed as given below in the following equation 4.10:

$$Cost(A) = \sum_{i=1}^{n} C(M_i) \tag{4.10}$$

Similarly, the bandwidth requirement will also be the sum of all the application modules placed on Cloud and fog devices and is computed as shown in the following equation 4.11.

$$BW(A) = \begin{cases} P_k * BR_{M_i}, & if(M_i \rightarrow cloud) \\ BR_{M_i}, & if(M_i \rightarrow fog) \\ \quad \forall (M_i, P_n) \in A \end{cases} \tag{4.11}$$

4.5 Design and Implementation

The Genetic Algorithm *(GA)* was created by Holland in 1975 [135] based on the theory of natural selection, according to which the fittest individuals are chosen to reproduce the next generation. The *GAs* begins with a primary population of chromosomes. A new population is formed in each generation using genetic operators such as mutation, crossover, and selection. The mutation operator searches the entire search space to recover lost genetic information and maintain population diversity. The crossover is the process of selecting two individuals and producing a new child from them.

The crossover operator's main task is to collect and combine both parents' positive characteristics to make more suitable offspring. In evolutionary algorithms, the selection operator is analogous to natural selection, which results in the survival of the fittest individuals. During evolution, the best solutions' genetic information has a higher chance of being passed down to future generations through reproduction.The main goal of the selection operators is to survive the fittest individuals by increasing the number of appropriate solutions in the next generation's population. It increases the chances of choosing the best current-generation solutions as parents and producing offspring. As a result, a new population emerges, establishing the next generation. The process is repeated until all of the specified stopping criteria have been met.

The basic flow of the Genetic Algorithm for our proposed scheduling model is shown in Figure 4.3. The scheduling is based on the computation and bandwidth requirements of application modules. After finding the threshold for application modules w.r.t the compute and bandwidth intensity, the more dense modules are sent directly to the Cloud for processing. The remaining application modules are sent to the *GA* for the optimized placement over the fog layer. In the next step, the parameters are initialed as the number of populations and the chromosomes in each population. We have used populations from 50 to 100 for our experiments, each having 50 chromosomes. The parameter setting is defined in the Table 4.2. For the initial population, we have used random numbers.

The fitness function is applied to the initial population to find the best chromosomes for the next population. After applying the fitness function on the randomly generated initial population, the fitness value is checked in the next phase to select the best chromosomes. We used the 20% ratio to choose the best chromosomes from the initial population to become part of the new population. The remaining chromosomes are sent to the next phase to apply crossover and mutation operator, as shown in Figure 4.3 (selection of chromosomes for the new population). The 20 selected best chromosomes from the initial population, and the remaining 80 chromosomes after applying crossover and mutation create the new population. The process is repeated until found the converged solution.

Algorithm 3 describes the main scheduling algorithm. After finding the threshold at lines 2 and 3, the application modules with more than computation and bandwidth threshold are directly passed to the Cloud for the processing from lines 4 to 7. The remaining modules are then passed to the genetic algorithm to define the optimized scheduling policy lines 9 to 12.

The performance of meta-heuristics is greatly influenced by the initial solutions. i.e., if the *GA* starts with a good initial population, it will almost certainly result in better final solutions. The diversity of an algorithm loses when a rule generates all the initial solutions as the produced solutions may be close to a particular solution space. In literature, most initial populations are randomly generated; therefore, we also decided to start the initial population randomly.

Algorithm 4 describes the main genetic algorithm for our proposed approach. For the initial population, we have the number of application modules from M_1 to M_n that need to be scheduled and the number of fog devices from F_1 to F_n as the resources for the application modules. The random numbers are generated according to the size of the number of application modules. For example, if we have four applications that need to be scheduled as A_1, A_2, A_3, and A_4 with modules as 14, 50, 30, and 20, respectively, the total number of modules that need to be scheduled is 114.

Hence we generate the 114 unique random numbers. In step 2, the initial population is generated at random based on chromosome encoding. These produced random numbers are assigned to each application module as their placement order in the scheduling scheme on the available fog devices, as shown in Figure 4.4.

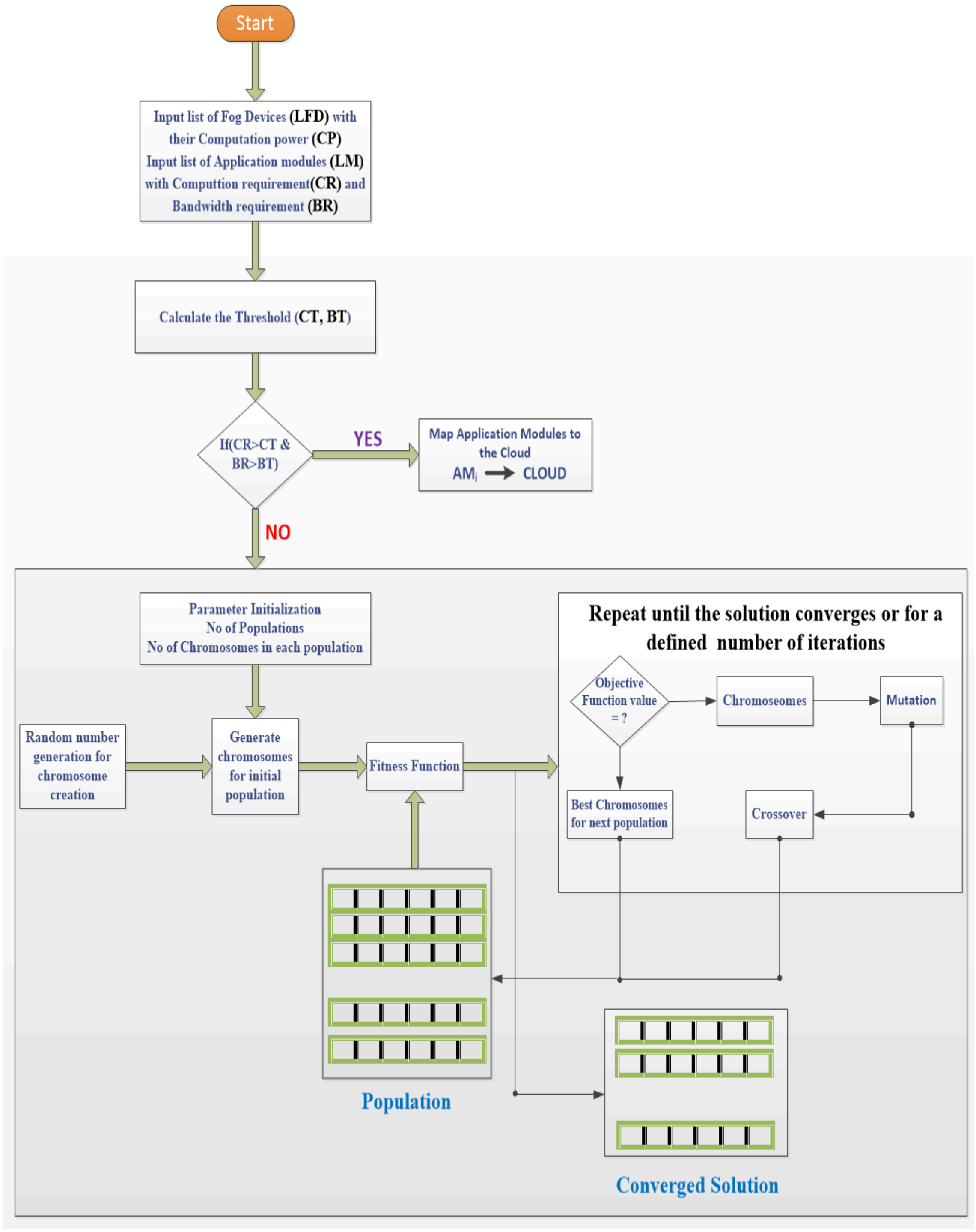
Start
Input list of Fog Devices (LFD) with their Computation power (CP) Input list of Application modules (LM) with Computtion requirement(CR) and Bandwidth requirement (BR)
Calculate the Threshold (CT, BT)
If(CR>CT & BR>BT)
YES
Map Application Modules to the Cloud
AM$_i$ → CLOUD
NO
Parameter Initialization No of Populations No of Chromosomes in each population
Random number generation for chromosome creation
Generate chromosomes for initial population
Fitness Function
Repeat until the solution converges or for a defined number of iterations
Objective Function value = ?
Chromoseomes
Mutation
Best Chromosomes for next population
Crossover
Population
Converged Solution

FIGURE 4.3: Flowchart of GA-IRACE

Step 3 to 5 computes the fitness value for each individual in the initial population. After finding the fitness, steps 6-25 conduct the evolution's iteratively.

We have selected the best 10% of individual for the next population from step 6 to 19. Steps 22 to 25 determine the individuals for crossover, mutation, and chromosome modificationbased on their fitness value.

```
Input: i)List of Modules ($LM$) with their computation requirement ($CR$)
       and bandwidth requirement ($BR$)
ii) List of Fog devices ($LF_d$) with their Processing Power ($MIPS$)
Output: $modulemap[][]$: Mapping of modules $LM$ on $LF_d$
MODULEMAP ($LM$, $LF_d$)
begin:
Find Computation Threshold ($CT$) for $CR$
Find Bandwidth Threshold ($BT$) for $BR$
for $i = 0$ to $LM.size$ do
    if $LM_{i(CPU)} > CT$ and $LM_{i(network)} > BT$ then
        $MAP\ (LM \rightarrow CLOUD\ )$
    end
    else
        $LM_{(R)} \leftarrow LM_i$
    end
end
$modulemap[][] = RACE\text{-}GA\ (LM_{(R)}, LF_d)$
return modulemap
end
```

Algorithm 3: GA-IRACE Scheduler

4.5.1 Chromosome Encoding

The first step in implementing *GA* is to create a chromosome-like scheme for the problem information. In our scheduling problem, we have to reduce the execution time while optimizing the monetary cost and the use of network bandwidth. The entire set of resources consists of fog devices *(FDs)* and Cloud resources *(CLOUD).* The *Bandwidth, RAM,* and *CPU* often describe a network node's resource capacity. Similarly, the A_iM_i application modules may be intensive w.r.t computation and bandwidth, depending on their specific needs [30, 90, 113, 114].

A two-dimensional matrix represents each application module and each fog device. As can be seen in Figure 4.4, modules are allocated to resources in rows, with each column representing a different resource type. The assignments of modules are under the scheme generated through random numbers. One random number generated is associated with one application module showing the application module's

```
Input: i) List of Modules ($LM$) with their computation requirement ($CR$)
       and bandwidth requirement ($BR$)
ii) List of Fog devices ($LF_d$) with their Processing Power (MIPS)
iii) Size of population (P)
iv) Maximum Generation ($Max_g$)
Output: modulemap[][]: Mapping of modules $LM$ on $LF_d$
$RACE\text{-}GA$ ($LM$, $LF_d$, $P$, $Max_g$)
begin:
$POP = InitialPopulation$ ($LM$, $LF_d$, $P$)
for $each\ Chromosomes\ in\ POP$ do
    $Calculate\_fitness$
end
for $generation = 1\ to\ Max_g$ do
    $new\_Elite\_fitness = Calculate\_Fitness\ (POP, P)$
    if ($elite\_fitness()> new\_elite.fitness()$) then
        $Elite\_POP = Elite\_ind$
    end
    else
        $new\_POP = Elite\_ind$
    end
    for $i = 1\ to\ Elite\_POP$ do
        $POP.insert(Elite\_POP)$
    end
    for $j = 1\ to\ new\_POP$ do
        $POP.insert(new\_POP)$
    end
    $Elite\_POP = Top\ 10\%of POP$
    $New\_POP = POP - Elite\_POP$
    $Parents = Selection\ (New\_POP, P)$
    $Children = Crossover\ (Parents, P)$
    $Mutated\_POP = Mutation\ (Children, P)$
    $POP = Mutated\_POP + Elite\_POP$
end
$return\ Elite\_POP$
end
```

Algorithm 4: GA-IRACE

position under the resource. e.g., the application module having random number 1 will be assigned first. Similarly, the module associated with random number 2 will be given to the next resource and in a similar fashion; all the modules are set to the available resources according to the assigned random numbers. One complete assignment of application modules to the fog devices according to the produced random numbers shows the completion of one chromosome as illustrated in Figure 4.4.

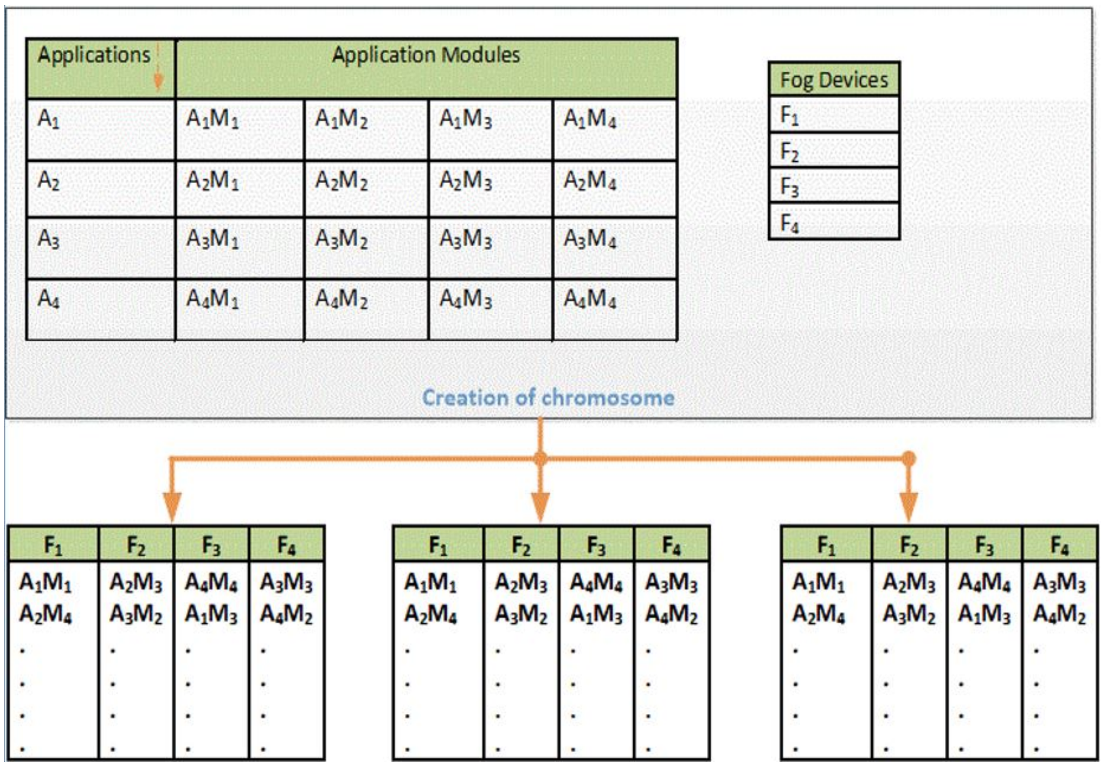

FIGURE 4.4: Chromosome Creation

4.5.2 Parent Selection Strategy

A parent selection is a method of selecting the chromosomes for crossing from among the current population. In our scenario, the fitness value for all the chromosomes in the population is calculated after the initial population is created by randomly assigning resources to application modules, as shown in Figure 4.3. Based on the calculated fitness value, the top 10 chromosomes are selected for the following population, while the remaining chromosomes are forwarded to the next phase as parents for the process of crossover and mutation.

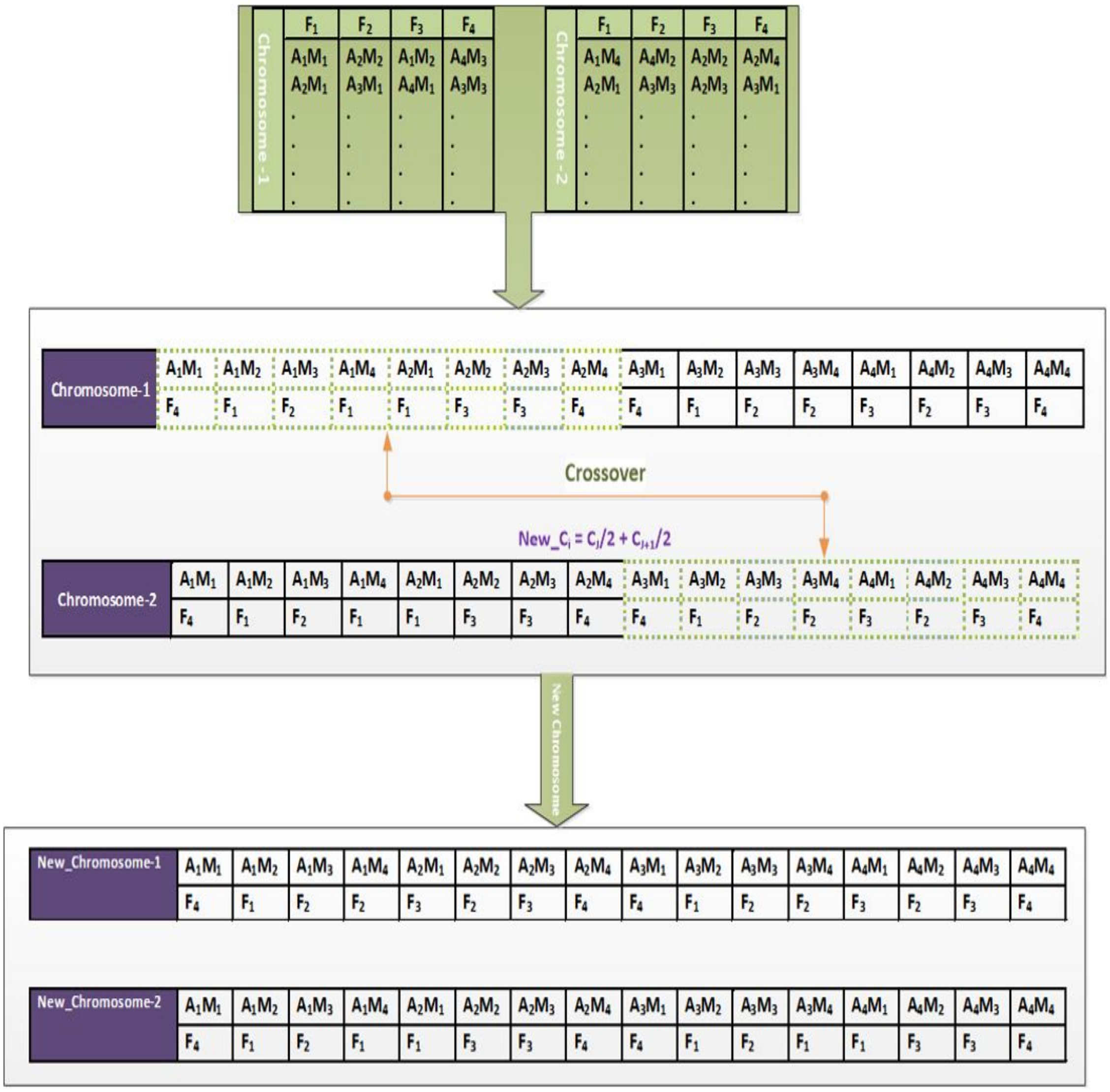
Chromosome-1
F1 F2 F3 F4
A1M1 A2M2 A1M2 A4M3
A2M1 A3M1 A4M1 A3M3
Chromosome-2
F1 F2 F3 F4
A1M4 A4M2 A2M2 A2M4
A2M1 A3M3 A2M3 A3M1
Chromosome-1
A1M1 A1M2 A1M3 A1M4 A2M1 A2M2 A2M3 A2M4 A3M1 A3M2 A3M3 A3M4 A4M1 A4M2 A4M3 A4M4
F4 F1 F2 F1 F1 F3 F3 F4 F4 F1 F2 F2 F3 F2 F3 F4
Crossover
New_Cj = Cj/2 + Cj+1/2
Chromosome-2
A1M1 A1M2 A1M3 A1M4 A2M1 A2M2 A2M3 A2M4 A3M1 A3M2 A3M3 A3M4 A4M1 A4M2 A4M3 A4M4
F4 F1 F2 F1 F1 F3 F3 F4 F4 F1 F2 F2 F3 F2 F3 F4
New Chromosome
New_Chromosome-1
A1M1 A1M2 A1M3 A1M4 A2M1 A2M2 A2M3 A2M4 A3M1 A3M2 A3M3 A3M4 A4M1 A4M2 A4M3 A4M4
F4 F1 F2 F2 F3 F2 F3 F4 F4 F1 F2 F2 F3 F2 F3 F4
New_Chromosome-2
A1M1 A1M2 A1M3 A1M4 A2M1 A2M2 A2M3 A2M4 A3M1 A3M2 A3M3 A3M4 A4M1 A4M2 A4M3 A4M4
F4 F1 F2 F1 F1 F3 F3 F4 F4 F1 F2 F1 F1 F3 F3 F4

FIGURE 4.5: Crossover Operation

4.5.3 Fitness Function

In the evolutionary algorithm, the fitness function determines the quality of the solution during the evolution process. In this study, the fitness of a schedule is related to minimizing the application's execution time, as shown in Equation 4.6. The fitness function is shown in the following equation 4.12 given below.

$$Fitness = \sum_{i=1}^{n} EFT(M_i), \qquad \forall (M_i \in A) \tag{4.12}$$

Where M_i show the application modules to be scheduled on the available fog devices. The EFT in equation 4.12 shows the estimated finish time of the application module and is computed using equation 4.6. The main objective of this optimization is to minimize the fitness value.

4.5.4 Crossover Operator

By swapping the information contained in the existing parents, the crossover operator is aimed at developing more compatible chromosomes, also known as offspring. In our case, we have the chromosome comprising of the application modules and their placement on the available resources. In chromosome arrangements, we have two options for the crossover operation.

1. Replace the fog resources of parent-1 (P_1) with the fog resources of parent-2 (P_2) without changing the sequence of application modules.

2. Arrange the application modules and resources in the chromosome in a 2D array format so that application modules are placed column-wise. Each subsequent row shows the assignment of fog devices for these application modules.

We have selected option 2 for our crossover method in which each row shows one chromosome, as shown in Figure 4.5. With crossover points CP_1 and CP_2, the two chromosomes are designated as parent-1 (P_1) and parent-2 (P_2). The cross points are selected as the central point of the P_1 and P_2. All the genes between CP_1 from P_1 are copied at CP_2 of P_2. Similarly, the genes from CP_2 of P_2 are copied to the CP_1 of P_1. The two new chromosomes, each carrying some genetic information from both parents, are created and are added to the new population.

```
**Input:** $i)\ Two\ parent\ individual's\ C_1,\ C_2$
$ii)\ Fog\ Device\ configuration,\ FDC$
**Output:** $new\_C_1\ ,\ new\_C_2$
$Crossover(C_1; C_2)$
**begin:**
$r_1 = length(C_1)/2$
$r_2 = length(C_2)$
**for** $i = r_1\ to\ r_2$ **do**
    $V_1 = C_1.Module[i]$
    $new\_C_1 \leftarrow V_1$
    $new\_C_1 \leftarrow FDC[V_1]$
    $V_2 = C_2.Module[i]$
    $new\_C_1 \leftarrow FDC[V_2]$
**end**
**for** $j = 1\ to\ r_1$ **do**
    $V_3 = C_2.Module[j]$
    $new\_C_2 \leftarrow V_3$
    $new\_C_2 \leftarrow FDC[V_3]$
    $V_4 \leftarrow C_1.Module[j]$
    $new\_C_2 \leftarrow FDC[V_4]$
**end**
$return\ new\_C_1$
$return\ new\_C_2$
**end**
```

Algorithm 5: Crossover

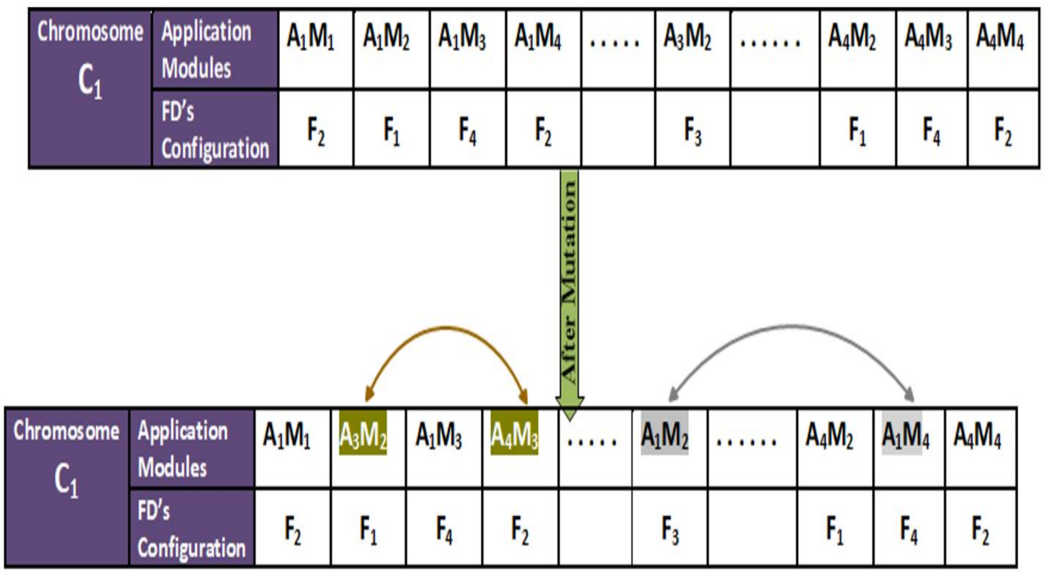

FIGURE 4.6: Mutation Operation

The process is repeated to create new chromosomes of the subsequent population. Algorithm 5 describes the crossover operation for our proposed methodology in which, at step 1, the two chromosomes are passed to the algorithm for crossover operation.

```
Input: i)Chromosome $C_1$
ii)Mutation rate $M_r$
Output: Mutated $C_1$, $M\_C_1$
Mutation $(C_1, M_r)$
begin:
length = $C_1.size()$
for $i$= 1 to length do
    Value = random ( )
    if ($M_r \leq Value$) then
        $M\_C_1 \leftarrow C_1.Module[i]$
        $M\_C_1 \leftarrow FDC[M_i]$
    end
end
return $M\_C_1$
end
```

Algorithm 6: Mutation

We have used the one-point crossover method in which the central point is designated as 'crossover point' at line 2. From lines 4 to 7, the genes to the left of the crossover point of C_1 are copied to the new chromosome new_C_1 along with the resource configuration i.e., the allocated fog devices for processing. Similarly from lines 9 to 13 the genes to the right of the crossover point, as well as the resource configuration, i.e., the assigned fog devices for processing, are copied to the new chromosome new_C_1.

4.5.5 Mutation

As a genetic operator, mutation provides genetic diversity in a population of genetic algorithm chromosomes from generation to generation. We have used the single-point mutation method, one of the common methods for implementing the mutation operator. Algorithm 6 shows the mutation method for our proposed approach in which, at line 4, the random numbers are generated for each gene in the chromosome.

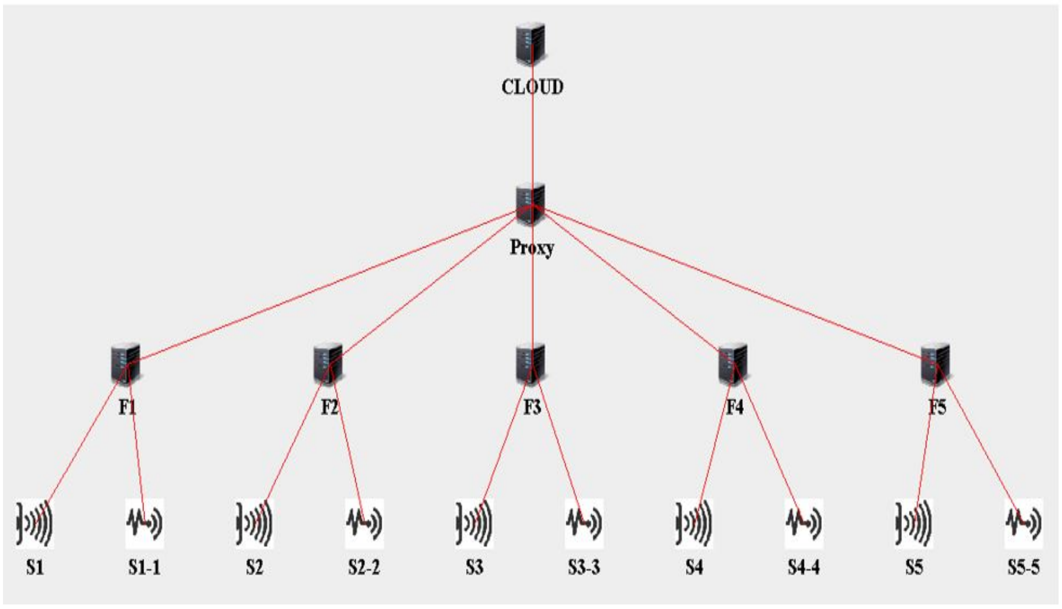

FIGURE 4.7: Example of Network Topology used in the Simulation [90]

In line 5 the generated random number is checked with the defined mutation rate. From line 6-8 the chromosomes are mutated if the generated random number and the mutation rate match with each other. Figure 4.6 also reflects the mutation process in which the random number generated for g_2 in c_1 is selected for swapping with g_4 in c_1. Similarly the following gene selected for swapping is A_1M_2 and A_1M_4. The process is repeated for all genes in each chromosome.

4.6 Simulation and Result Discussion

For experiments, we have used the system with an Intel Core i7-8550U Quad-core processor (clocked at 1.9 GHz) and 8 GB of main memory. The *GA* is implemented in C# to generate the optimized scheduling policy for the application modules. The scheduled policy obtained from *GA* is then passed to the fog system simulated in *iFogSim* [90].

To ensure that our suggested strategy is effective, we have modeled and analyzed the Cloud fog ecosystem using the open-source *iFogSim* framework. Table 4.3 outlines the simulation environment settings for our experiments. The population size used for the genetic algorithm was 100, with chromosome size remaining 50 in each population. Similarly, the crossover rate was set to 5% with the mutation rate of 3% [136], as shown in Table 4.2.

To verify the performance of *GA-IRACE*, we have used three different scenarios with network topologies of 4, 8, and 10 in a Cloud Fog computing environment. Figure 4.7 shows a graphical representation of one of these topologies generated using *iFogSim*. We have used the workload of three different applications A_1, A_2, and A_3, with varying scales as bandwidth-intensive, compute-intensive and mixed (bandwidth and compute-intensive). The number of modules in the application varies from 70 to 100, with *MI's* of the module ranges from 100 to 900*MI* as shown in Table 4.3.

TABLE 4.3: GA-IRACE Simulation Setup

Application Type	Compute Intensive (Application-1)	Compute Intensive (Application-2)	Mixed (Compute + Bandwidth Intensive) (Application-3)
No of Modules	70	80	100
MI in modules	100-900	20-200	40-500
No of Fog Devices	4,8,10		
CPU(MIPS) of Fog Devices	200-3000		
CPU(MIPS) of Cloud	42800		

4.6.1 Discussion of Experimental Results

In the experiments, the proposed scheduling algorithm *GA-IRACE* was compared with the traditional Cloud placement, also called conventional Cloud placement (*CCP*), *random solution*, *RACE* [21] and the other baseline algorithms, i.e., Taneja. et al. [24] regarding bandwidth consumption, execution time, and the Cloud resources cost. The simulation findings Figures 4.8 to 4.16 show that the proposed placement technique has a tremendously positive effect on execution time, bandwidth, and monetary cost of applications in all three network topologies.

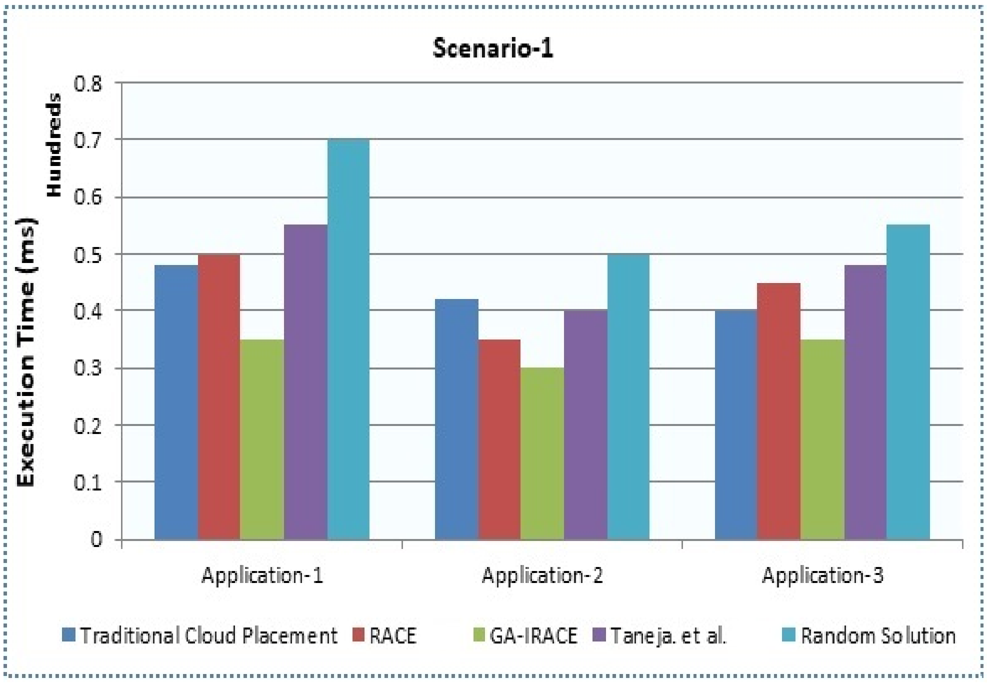

FIGURE 4.8: Execution Time with 4FDs

Figure 4.8 shows that the execution time of A_1 is significantly improved as 13%, 5%, 7%, and 5%, respectively, when using the *GA-IRACE* scheduling policy as compared to the *random solution*, *RACE* [21], Taneja. et al.[24] and *CCP*. It can be seen in Figure 4.8 that increasing the number of applications with modules affects the performance of *RACE* [21] however, the optimized scheduling policy of *GA-IRACE* improves the execution time with the placement of application modules from fog to the Cloud layer. For A_2 the bandwidth-intensive application *GA-IRACE* improves the execution time to 11%, 5%, 3%, and 6%, respectively, compared to the random solution, Taneja. et al.[24], *RACE* [21] and *CCP*.

Similarly, for A_3, the mixed application for computation and bandwidth-intensive, the *GA-IRACE* is faster than the baseline algorithm, the Taneja. et al.[24], the *RACE* [21], *CCP* and the random solution in execution time as 5%, 4%, 3%, and 9% respectively.

Figure 4.9 shows the bandwidth consumption of three applications, i.e., A_1, A_2, and A_3. It can be seen that all three applications have maximum bandwidth consumption when considering only the *CCP* and the random placement. Considering the fog devices with the Cloud resources reduces the bandwidth consumption for three applications by RACE [21] and the Taneja. et al. [24], however, the *GA-IRACE* has the better results from all three baseline algorithms.

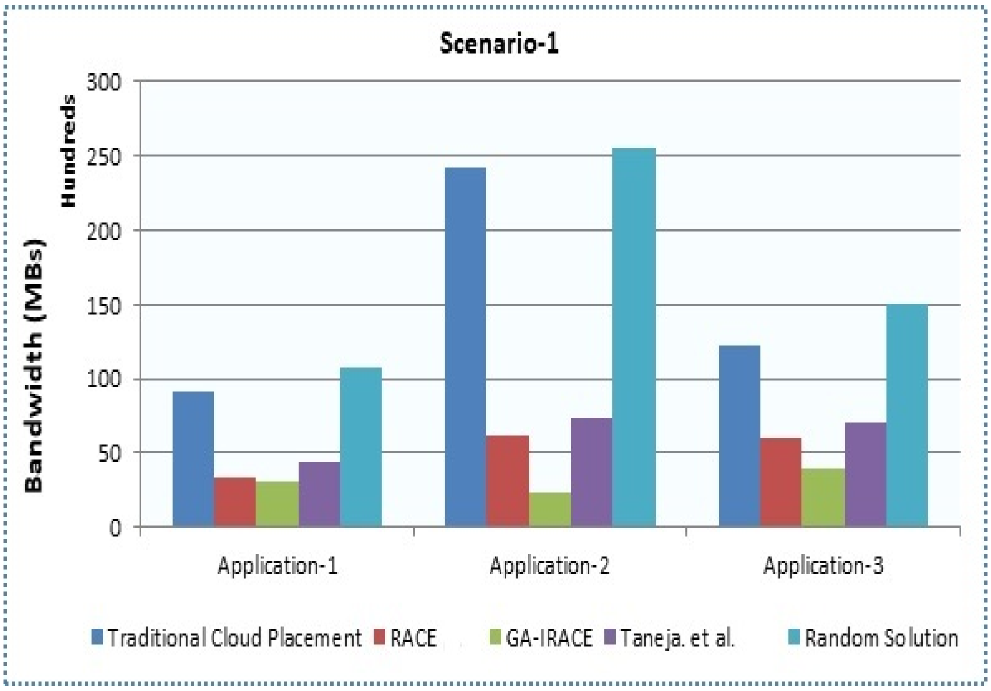

FIGURE 4.9: Bandwidth Consumption with 4FDs

For A_1, the *GA-IRACE* consumes 4%, 2%, 20%, and 25% less bandwidth consumption when compared with the Taneja. et al.[24], *RACE* [21], *random solution*, and *CCP*. Similarly, for A_2, the *random solution* and *CCP* have the maximum bandwidth consumption of 37% and 39%, respectively, due to the nature of the application and the placement strategy in which maximum modules are placed at the Cloud. *GA-IRACE* improves the bandwidth consumption of A_2 to 7%, 8%, 32% and 34% when compared with the *RACE* [21], Taneja. et al.[24], *random solution*, and *CCP*.

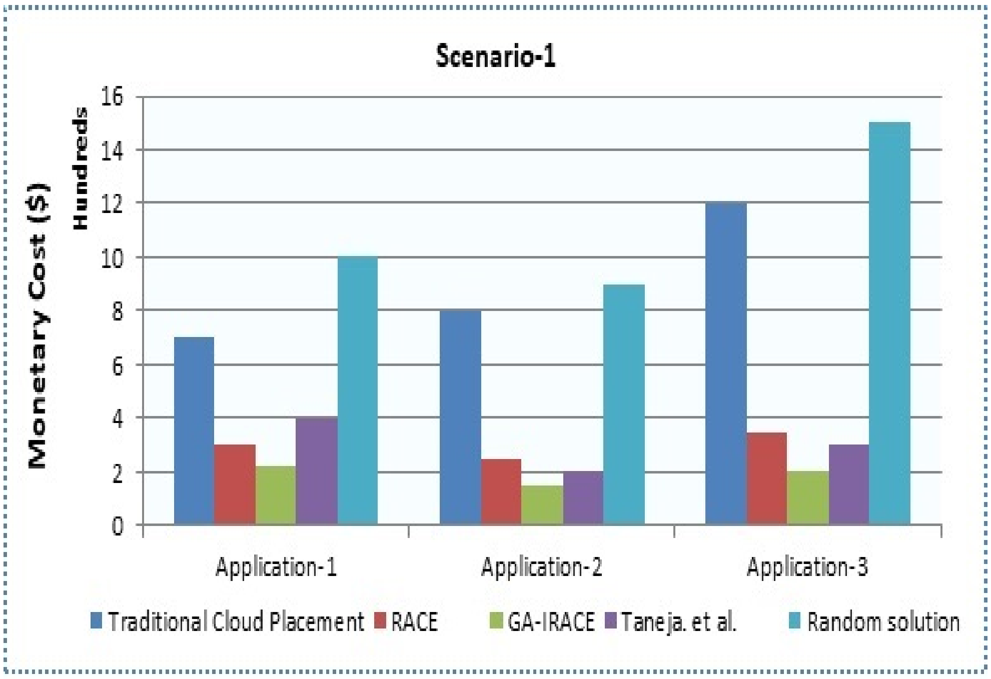

FIGURE 4.10: Monetary Cost with 4FDs

Similar improvements can be seen for A_3 for which using the *GA-IRACE* reduces the bandwidth consumption to 25%, 19%, 7%, and 4% when compared with the *random solution*, *CCP*, Taneja. et al.[24] and *RACE* [21]. The increase in usage of Cloud resources increases the monetary cost of the application, as shown in Figure 4.10 in which the *CCP* and *random solution* strategy has the maximum monetary cost for all three applications in comparison to the Taneja. et al.[24], *RACE* [21] and *GA-IRACE*.

The optimized scheduling of *GA-IRACE* reduces the monetary cost for A_1 to 7% and 4% from the Taneja. et al.[24] and *RACE* [21] scheduling scheme. For A_2 the *GA-IRACE* reduces the monetary cost to 3%, 5%, 33%, and 29% respectively, compared to Taneja. et al.[24], *RACE* [21], *random solution* and *CCP*. For A_3 the *GA-IRACE* also has better results for monetary cost with an improvement of 2% and 4% compared with the Taneja. et al.[24], *RACE* [21] placement strategy.

Increasing the number of fog devices improves the execution time of the applications along with changes in bandwidth consumption and the monetary cost, as can be seen from Figures 4.11 to 4.13.

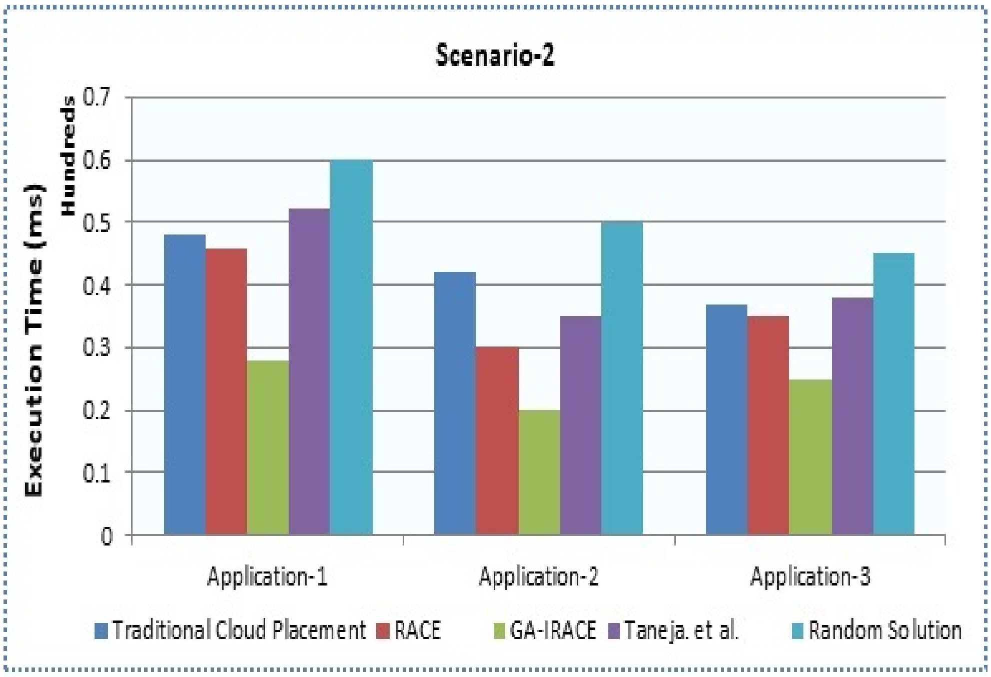

FIGURE 4.11: Execution Time with 4FDs

Figure 4.11 shows the efficient scheduling of our proposed algorithm for the three applications, i.e., A_1, A_2, and A_3, respectively.

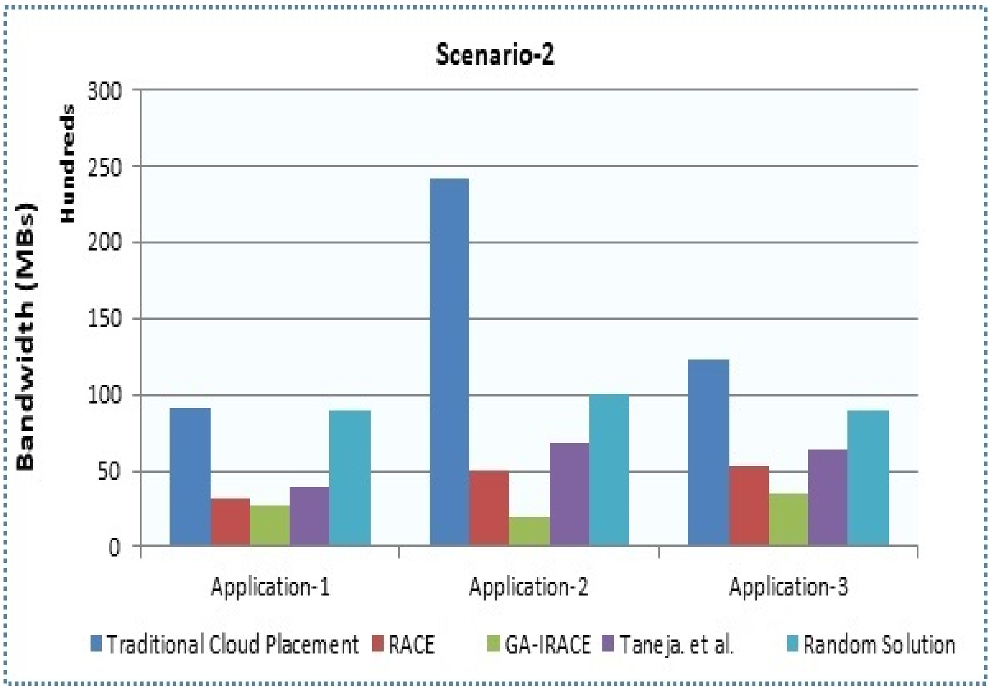

FIGURE 4.12: Bandwidth Consumption with 8FDs

It can be seen in Figure 4.11 that the *GA-IRACE* improves the execution time for A_1 from the *RACE* algorithm [21], Taneja. et al. [24], *random solution*, and *CCP* as 5%, 7%, 13% and 5% respectively. Similarly, for A_2, the *GA-IRACE* has a better performance with less execution time of 9%, 6%, 17%, and 13% compared to the Taneja. et al. [24], *RACE* algorithm [21], *random solution* and *CCP*.

For A_3, the mixed application for computing and bandwidth-intensive the Taneja. et al.[24], and *RACE* algorithm [21] has minor improvement in the execution

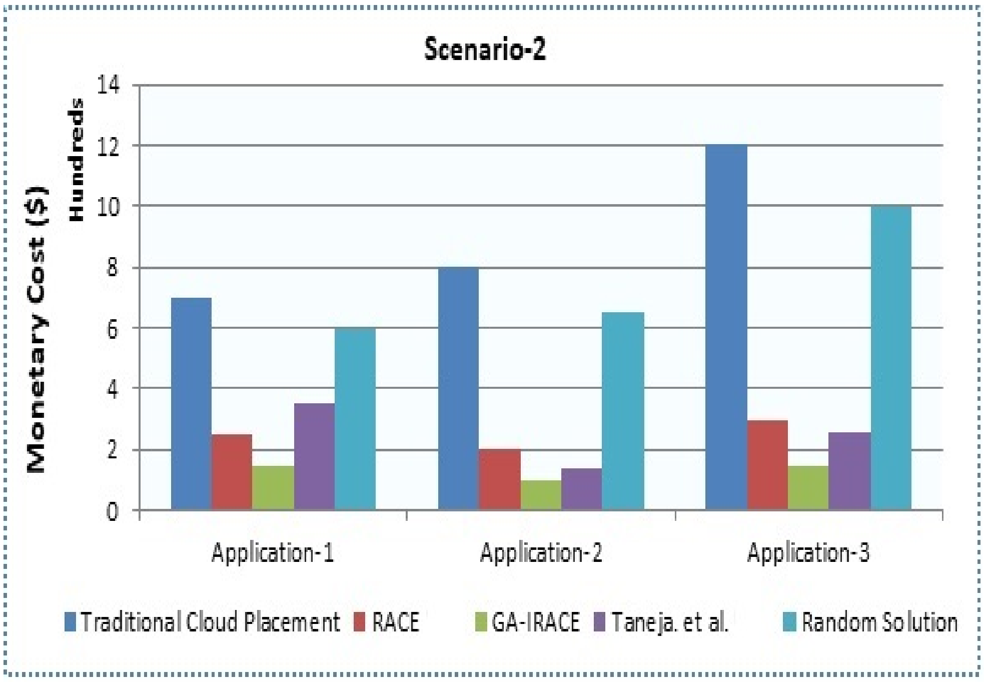

FIGURE 4.13: Monetary Cost with 8FDs

time of 2% and 3% when compared with the *CCP*. However, both improves the execution time to 4% and 6% compared to the *random solution*. The *GA-IRACE* has less execution time of 7% and 5% compared with the Taneja. et al. [24] and *RACE* algorithm [21]. Figure 4.12 illustrates the positive impact of *GA-IRACE* on the bandwidth consumption when we have eight fog devices.

By adding more fog nodes, bandwidth consumption is improved compared to *CCP* that uses the Cloud as a platform for modules, requiring more bandwidth from the end user to the Cloud.

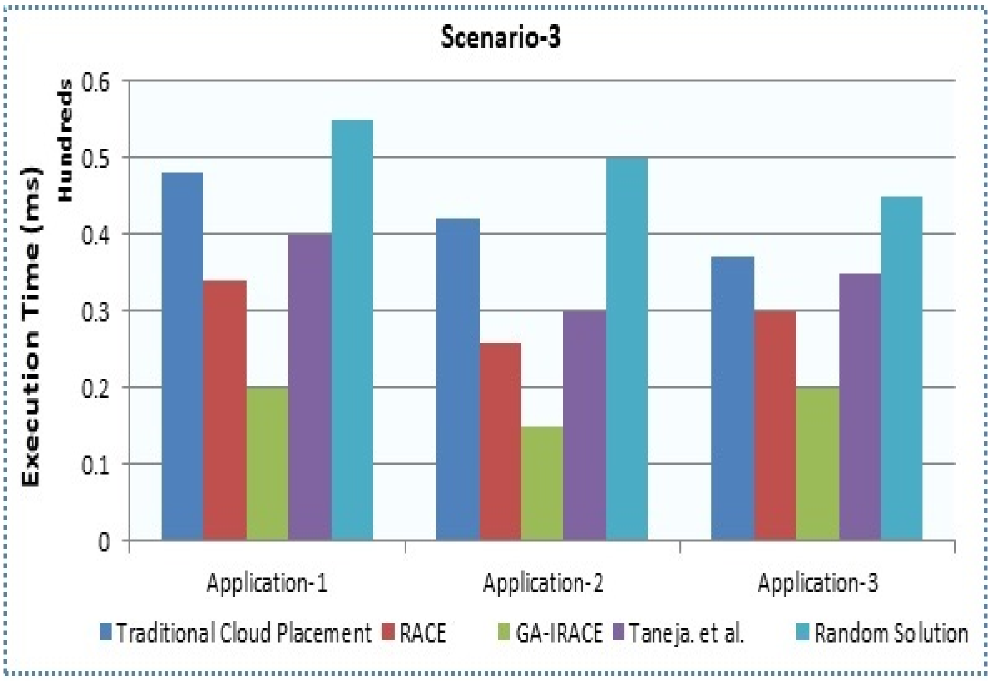

FIGURE 4.14: Execution Time with 10FDs

Although, the performance of *RACE* [21] is 4% better than Taneja. et al.[24] for A_1, 3% for A_2 and 3% for A_3 however, *GA-IRACE* outperforms for all three applications with lower bandwidth consumption. Figure 4.12 shows that *GA-IRACE* has 2%, 7% and 4% less bandwidth consumption than the *RACE* [21].

Figure 4.13 shows the monetary cost for the three applications with eight fog devices. Although Taneja. et al.[24] improves the monetary cost for A_1 when compared with the random solution and the *CCP*; however, the *RACE* [21] has 5% less monetary cost with better utilization of resources at the fog layer. Following this, the *GA-IRACE* has a lower monetary cost of 4% than the *RACE* [21]. For A_2, the *GA-IRACE* has the less monetary cost of 3% and 6% than the Taneja. et al.[24] and *RACE* [21]. Similarly, for A_3, the *GA-IRACE* also improves the monetary cost with 4% and 5% less compared to the Taneja. et al.[24] and *RACE* [21]. Figures 4.14 to 4.16 illustrates the execution time of three applications, bandwidth, and the monetary cost when using the network topology of ten fog devices. Figure 4.14 depicts that in the case of A_1, *GA-IRACE* improves the execution time compared to the *CCP*, *random solution*, *RACE* [21], and the baseline algorithm, i.e., Taneja. et al.[24] with 38%, 45%, 7% and 10% respectively.

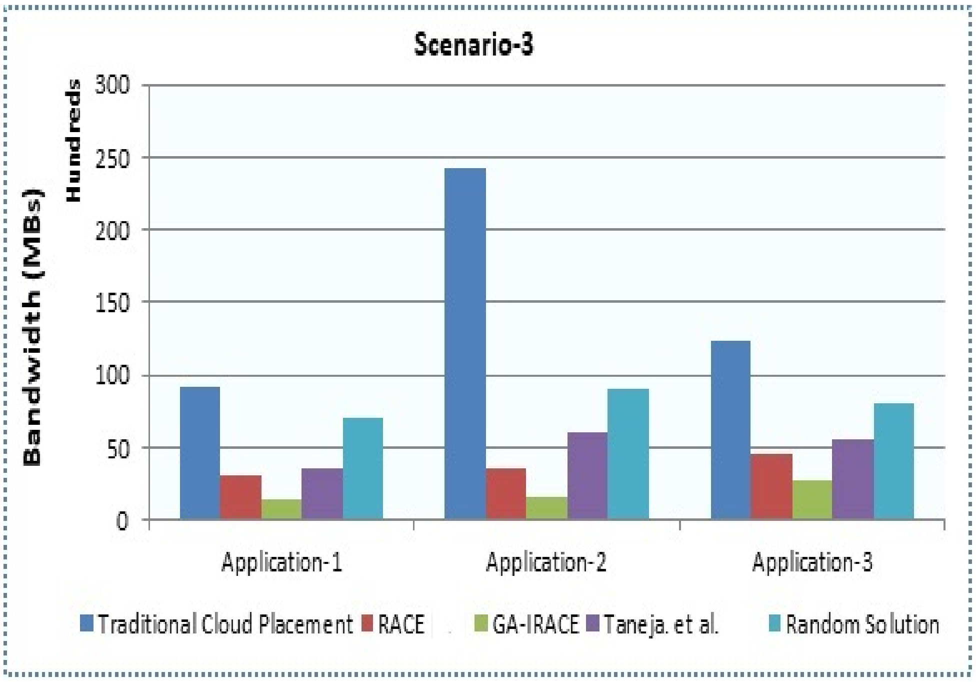

Figure 4.15: Bandwidth Consumption with 10FDs

For A_2, the bandwidth-intensive application, the execution time improves to 3% and 2% respectively by *GA-IRACE* scheduling policy compared with the *RACE* [21], and Taneja. et al.[24]. Similarly, for A_3, the *GA-IRACE* also has better results with an improvement of 6% and 9% respectively, in execution time as compared to *RACE* [21], and Taneja. et al.[24]. From all the three approaches, our proposed approach, i.e., *GA-IRACE*, better absorbs the increasing number of fog devices and has a lower monetary cost for all three applications.

A graph of the bandwidth consumption is presented in Figure 4.15 for the three applications with 10 fog devices. For the three applications A_1, A_2, and A_3, the *CCP* consumes the most bandwidth, as the modules are located within the Cloud. For *RACE* [21] and Taneja. et al.[24], shifting modules from the fog to the Cloud layer under the defined scheduling policy reduces the bandwidth consumption for A_1 to 25% and 20%, respectively, compared to the *CCP*. Similar improvement can be seen in comparison to the *random solution* for which the *RACE* [21] and

Taneja. et al.[24] has 16% and 14% less bandwidth consumption for A_1, 12% and 7% for A_2, and 10% and 6% for A_3.

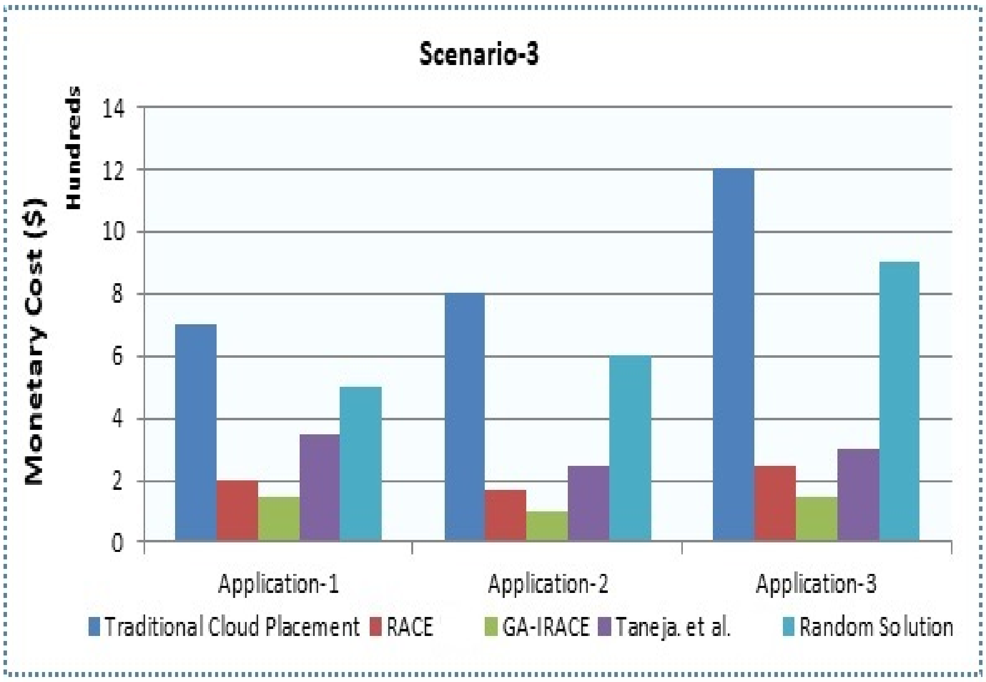

FIGURE 4.16: Monetary Cost with 10FDs

In comparison to *CCP*, *random solution* and other baseline algorithms (*RACE* [21], Taneja. et al.[24]), *GA-IRACE* has better results for A_1 with an improvement of 32%, 23%, 7%, and 8% respectively. For A_2, the *GA-IRACE* also improves the bandwidth consumption to 6% and 11% in comparison to the *RACE* [21] and Taneja. et al. [24] baseline algorithms. In Figure 4.16, the cost per application for using Cloud resources in a network topology of 10 fog devices is shown. As shown in Figure 4.16, by adding more resources to the fog layer, Cloud resources are used with less monetary cost, as there are adequate resources at the fog layer.

Therefore, the *RACE*[21] scheduling policy has less cost for all three applications when compared with the *CCP*, *random solution*, *GA-IRACE*, and the Taneja. et al.[24]. However, the efficient scheduling with maximum resource utilization also affects the performance as it can be seen that the optimized scheduling with *GA-IRACE* has better results of 43%, 38%, and 29% respectively from *CCP*, *random solution* and Taneja. et al.[24]. Similarly, the *GA-IRACE* has a lower monetary cost of 6%, 7%, and 15% for A_2, 7%, 40%, and 53% for A3 when compared with the Taneja. et al.[24], *random solution* and *CCP*.

4.7 Conclusions

Advances in communication technologies have anticipated the ubiquitous use of Internet of Things *(IoT)* devices in the future. Consequently, proper utilization of Cloud computing for real-time applications regarding latency, cost, and bandwidth consumption remains inevitable. The Fog computing paradigm mitigates this limitation because it provides a convenient and efficient way to provide a processing facility close to the edge of the user network. However, scheduling jobs on Fog computing becomes a challenging task due to the resource constraint of edge devices. This research presents a genetic algorithm-based strategy for optimizing module scheduling across the fog and Cloud layers of the three-tier Cloud Fog computing architecture.

According to the results, implementing the *GA*-based recommendation improves execution time, cost, and bandwidth utilization by 15-40%. Furthermore, compared to fundamental algorithms, the enhanced scheduling makes less use of network resources, faster and has less financial costs for using the Cloud resources. This work can be extended in several directions in the future. We utilize only the execution time metric as a fitness function to evolve the scheduling of the application modules. However, the monetary cost, bandwidth requirements, and/or latency can be studied as a fitness function. Another direction for the future is developing a multi-objective fitness function to find the trade-offs between these factors in the context of Cloud Fog computing scheduling. Another exciting aspect of the study would be to evaluate the proposed approach for large corpus applications.

Chapter 5

Conclusion and Future Work

5.1 Conclusions

Increases in the capacity of networks indicate that IoT devices will soon be commonplace, which means optimizing Cloud services for real-time applications concerning latency, cost, and bandwidth usage will continue to be essential. Fog computing can overcome this barrier because it places a processing facility at the user network's edge, which is more accessible and efficient. However, edge devices' limited resources make scheduling work on Fog computing arduous. This study introduces a heuristic based and a genetic algorithm-based approach to improving module scheduling in the three-tier Cloud Fog computing architecture.

Data produced daily by the IoT has skyrocketed as the number of connected devices has grown. Using Cloud services is essential for storing, processing, and analyzing this massive volume of data. Thanks to cloud computing, we now have a simple mechanism for archiving, processing, and analyzing this deluge of information. Real-time applications like smart city management, healthcare, online gaming, and autonomous cars are hampered by the high latency and bandwidth requirements. By using Cloud services, these applications slow down execution, increase the price and bandwidth costs for end users. The Fog computing architecture is one of the emerging technologies designed to provide quality service to applications by bringing Cloud services closer to the edge of the network. As technologies like the *IoT*, Cloud computing, and mobile internet advance quickly, the Fog computing paradigm is becoming more critical. Fog computing's main objective is to address the issues that have surfaced with Cloud computing. The system's resources are dynamic and spread out around the globe, making management difficult. A well-oiled system for scheduling and allocating resources is essential to make the most of these assets and guarantee customer satisfaction.

Fog computing, which moves Cloud resources to the network's edge, might be used to provide quality of service*(QoS)* for such applications. Since the execution time depends on the kind of application and the level of computing required, we cannot predict how long it will take if all of the modules are placed on fog. Putting everything on the Cloud ensures quicker processing times but at the cost of increased bandwidth and Cloud resources. Since there is a delay in moving applications from fog to the Cloud, execution time might sometimes increase. Thus, applications need to be scheduled from the fog layer to the cloud layer. Scheduling is normally NP-hard, but Fog computing makes scheduling more challenging due to the diversity and scattered nature of fog layer resources, user mobility, and varying application requirements.

Taking into mind the computing capabilities of fog devices and the computational needs of the applications, we provide a novel scheduling approach termed *RACE* in the first part of this thesis. The Resource Aware Cost-Efficient scheduler *(RACE)* aims to reduce the financial cost of using Cloud resources while maximizing resource utilization at the fog layer. The heuristics we implemented in *RACE* scheduling considered computation and bandwidth usage, stacking applications so that applications with fewer resources were at the bottom and applications with more resources were grouped and placed at the top.

There are two aspects to the proposed strategy: One section sorts incoming application modules by computing, and bandwidth needs to generate a list of modules in order of priority for placement in the fog layer. Based on the prioritized list, the application modules are placed from the fog to the Cloud layer, closely monitoring resource capacity and module requirements Chapter 3 covers the *RACE*. Since our solution involves moving application components from the fog to the Cloud, taking the benefit of the scalability and speed of the Cloud at a reduced cost, a weighted ranking mechanism for the placement of application modules has been created to lessen the impact on latency, processing time, and the price of using Cloud resources. Both whole applications and their portions are considered to be aspects of the workload.

The experiments were carried out on the three application datasets, varying the network topology of fog devices at the fog layer from 4 to 10. The simulations are carried out with the placement of application modules by using only the Fog layer, only using the Cloud layer, and a hybrid Cloud-Fog configuration. The real-world testing of the suggested approach would be challenging due to its limited applicability. Also, with the novelty and rapid evolution of this sector, it is now preferred to conduct research and development in a virtual environment; therefore, we have used an open-source program for simulating and analyzing the operation of Cloud fog infrastructure and services. The *iFogSim* architecture is widely used to model various computer system designs. Three variations of this situation, each with a unique network topology and workload, have been developed.

In most cases, extensive modeling with the *iFogSim* simulator confirms that our approach outperforms the conventional Cloud placement*CCP* and baseline methods. Regarding time efficiency, cost efficiency, and reducing network traffic, *RACE(CFP)* performs well across the board. The *RACE(FOP)* has better results for the bandwidth and Cloud resources cost but for compute-intensive applications; it reduces the execution time to almost 10%. Compared to the *CCP*, the *RACE(FOP)* improves the execution time from 20% to 30% with an increase in the number of fog devices. For bandwidth and the Cloud resources cost, the *RACE(FOP)* has almost 30% to 60% less bandwidth and Cloud resources cost when compared with the *CCP*. The *RACE(CFP)* strategy has 20% to 40% better results in execution time, bandwidth consumption, and the Cloud resource cost with an increase in the number of fog devices at the Fog layer for *RACE(FOP)*, the *CCP*, and the Taneja. et al. [24]

The simulation results of *RACE* show that the proposed scheduling method, when applied using the combination of fog and the Cloud layer, is more effective than the methods of Taneja. et al. [24], the more common Cloud methods, and the stand-alone fog placement approaches. However, as the number of applications expanded, they became resource-intensive, needing more time to execute and more bandwidth. Also, sending modules to the Cloud raises cost and bandwidth, and sending them to the network's edge causes delays for applications. Therefore optimized scheduling of applications modules from fog to the Cloud is required. The second section of the thesis suggests an enhanced version of a Resource-Aware, Cost-Effective Scheduler *(GA-IRACE)* for the Cloud fog setting. The findings show that the proposed enhanced *GA*-based strategy reduces execution time, costs, and bandwidth utilization by 15-40%.

Since we need to optimize the execution time and the bandwidth, we turned to *GA* for help. *GA-IRACE*'s module placement optimization saves time and money by strategically placing applications inside a Cloud's infrastructure. *GA-IRACE*'s execution time was also lower than that of the *RACE* algorithm, particularly as the number of applications increased. Therefore, *GA-IRACE* was used to carry out optimized placement, producing better results than any other approach.

A genetic algorithm repeatedly refines the proposed answer using data on how long it takes different programs to do their work. At the outset, the suggested technique encodes the chromosome-based problem representation by considering the features of the application and the fog devices. Using re-population operators, the solutions emerged by modifying the GA's many parameters. The program uses *GA* to schedule its components, and the results have been verified using the *iFogSim* simulator.

The experiments are carried out on the three application datasets with modules varying from 70 to 100 and the MI in modules ranging from 100 to 500. The experiments are varied under three different network topologies, changing the number of fog devices from 4 to 10 at the fog layer. The *iFogSim* simulator was used to compare the *GA-IRACE* with the *RACE* and other baseline algorithms.

Simulation results show an improvement of 15% to 20% in execution time, bandwidth, and the Cloud resources cost. For all three applications, the *GA-IRACE* has less execution time, with low bandwidth consumption and Cloud resources cost of almost 10% to 15% when compared with the baseline algorithms. As a bonus, the improved scheduling uses fewer Cloud resources, is quicker, and needs less bandwidth than the standard approach.

The optimized placement of application modules using *GA-IRACE* reduces the execution time and has less bandwidth and overall costs when using Cloud resources. Compared to the results of the *RACE* algorithm, the execution time with *GA-IRACE* was also lower, especially in the case of an increasing number of applications. As a result, *GA-IRACE* was used to perform optimized placement, yielding superior results compared to alternative methods.

5.2 Limitations

Following are some of the limitations of the work proposed in this thesis.

1. A limitation of the scheduling approach proposed in this thesis is that the computation and the bandwidth requirement of all application modules should be known before mapping in Cloud fog computing environment.

2. In this research, it is assumed that the execution starts after the successful mapping of all incoming application modules arrived at the same time.

3. This research do not consider the deadline of application modules.

5.3 Future Research Directions

A list of possible research directions is presented in this section for further development of the work in this thesis.

1. Future work of this research has numerous factors, such as time constraints associated with latency-sensitive applications. since there are different time-constraints depending on the application. As a glimpse into the future, it has to be considered how Fog computing may support these sorts of applications.

2. This work can be extended in several directions in the future. We utilize only the execution time metric as a fitness function to evolve the scheduling of the application modules. However, the monetary cost, bandwidth requirements, and/or latency can be studied as a fitness function. Another direction for the future is developing a multi-objective fitness function to find the trade-offs between these factors in the context of Cloud Fog computing scheduling. Another exciting aspect of the study would be to evaluate the proposed approach for large corpus applications.

3. Fog computing is a new paradigm facing many challenges, from scheduling application modules to placing fog servers with efficient resource utilization and security issues. The efficient utilization of the resources at the fog layer with satisfying the deadline constraints is a big challenge. The heterogeneous and limited computation power of fog devices at the fog layer is a significant obstacle to satisfying the quality constraint.

4. The offloading of tasks from the fog layer to the Cloud for the delay sensitive applications to be placed at the fog layer when all devices are full may affect the performance due to data transfer cost as faced in Cloud. Moreover, novel techniques should be investigated for sharing of resources at the fog layer.

5. There is a need to define energy-efficient algorithms to save the devices' energy at the fog layer since they are not so energy-efficient. Among the strategies used by the researcher is to sleep idle devices or make those devices inactive by transferring the load to other devices. Due to the interdependence between application modules, the *FSP* must accurately estimate the dynamic resource requirements. Resource demand prediction techniques based on regression and machine learning are needed in Fog computing.

6. In the fog, devices may send data or computationally intensive applications to a nearby fog node rather than to the Cloud. While offloading might decrease device energy usage, it may increase execution time due to additional waiting and processing on fog servers and during device transmission. Finding the appropriate offloading probability and transmit power for each device requires a multi-objective optimization problem with the shared goal of reducing execution time, cost, and energy consumption.

7. The number of devices connected to fog nodes and at different gateways raises security concerns in Fog computing. In fog nodes, each device has a unique IP address, and hackers can use this information to access the information. More research is required to secure the data at the fog layer.

Bibliography

[1] J. Manyika, M. Chui, P. Bisson, J. Woetzel, R. Dobbs, J. Bughin, and D. Aharon, “Unlocking the potential of the internet of things,” *McKinsey Global Institute*, vol. 1, 2015.

[2] F. S. Collins, E. D. Green, A. E. Guttmacher, and M. S. Guyer, “A vision for the future of genomics research,” *nature*, vol. 422, no. 6934, pp. 835–847, 2003.

[3] S. Smith, “““internet of things’ connected devices to almost triple to over 38 billion units by 2020 - juniper,” 2016.

[4] B. P. Rimal, E. Choi, and I. Lumb, “A taxonomy and survey of cloud computing systems,” in *2009 Fifth International Joint Conference on INC, IMS and IDC*, pp. 44–51, 2009.

[5] S. Misra and S. Sarkar, “Theoretical modelling of fog computing: A green computing paradigm to support iot applications,” *IET Networks*, vol. 5, 02 2016.

[6] F. Bonomi, R. A. Milito, J. Zhu, and S. Addepalli, “Fog computing and its role in the internet of things,” in *MCC ’12*, 2012.

[7] A. V. Dastjerdi, H. Gupta, R. N. Calheiros, S. K. Ghosh, and R. Buyya, “Fog computing: Principles, architectures, and applications,” in *Internet of things*, pp. 61–75, Elsevier, 2016.

[8] N. V. and R. Lavanya, *Fog Computing and Its Role in the Internet of Things*, pp. 63–71. 01 2019.

[9] B. Varghese, N. Wang, S. Barbhuiya, P. Kilpatrick, and D. S. Nikolopoulos, "Challenges and opportunities in edge computing," in *2016 IEEE International Conference on Smart Cloud (SmartCloud)*, pp. 20–26, IEEE, 2016.

[10] S. Sarkar, S. Chatterjee, and S. Misra, "Assessment of the suitability of fog computing in the context of internet of things," *IEEE Transactions on Cloud Computing*, vol. 6, no. 1, pp. 46–59, 2018.

[11] M. Amani, A. Ghorbanian, S. A. Ahmadi, M. Kakooei, A. Moghimi, S. M. Mirmazloumi, S. H. A. Moghaddam, S. Mahdavi, M. Ghahremanloo, S. Parsian, *et al.*, "Google earth engine cloud computing platform for remote sensing big data applications: A comprehensive review," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 13, pp. 5326–5350, 2020.

[12] A. P. Plageras, K. E. Psannis, C. Stergiou, H. Wang, and B. B. Gupta, "Efficient iot-based sensor big data collection–processing and analysis in smart buildings," *Future Generation Computer Systems*, vol. 82, pp. 349–357, 2018.

[13] S. Lu, J. Wu, Y. Duan, N. Wang, and J. Fang, "Towards cost-efficient resource provisioning with multiple mobile users in fog computing," *Journal of Parallel and Distributed Computing*, vol. 146, pp. 96–106, 2020.

[14] R. Mahmud, K. Ramamohanarao, and R. Buyya, "Latency-aware application module management for fog computing environments," *ACM Transactions on Internet Technology (TOIT)*, vol. 19, no. 1, pp. 1–21, 2018.

[15] H. Tabrizchi and M. Kuchaki Rafsanjani, "A survey on security challenges in cloud computing: issues, threats, and solutions," *The journal of supercomputing*, vol. 76, no. 12, pp. 9493–9532, 2020.

[16] M. Al Yami and D. Schaefer, "Fog computing as a complementary approach to cloud computing," in *2019 International Conference on Computer and Information Sciences (ICCIS)*, pp. 1–5, IEEE, 2019.

[17] S. Kunal, A. Saha, and R. Amin, “An overview of cloud-fog computing: Architectures, applications with security challenges,” *Security and Privacy*, vol. 2, no. 4, p. e72, 2019.

[18] A. M. Alqahtani, S. H. Mohamed, T. E. El-Gorashi, and J. M. Elmirghani, “Pon-based connectivity for fog computing,” in *2020 22nd International Conference on Transparent Optical Networks (ICTON)*, pp. 1–6, IEEE, 2020.

[19] F. Bonomi, R. A. Milito, J. Zhu, and S. Addepalli, “Fog computing and its role in the internet of things,” in *MCC ’12*, 2012.

[20] V. K. Vaishnavi and W. Kuechler, *Design science research methods and patterns: innovating information and communication technology.* Crc Press, 2015.

[21] J. U. Arshed and M. Ahmed, “Race: Resource aware cost-efficient scheduler for cloud fog environment,” *IEEE Access*, vol. 9, pp. 65688–65701, 2021.

[22] J. U. Arshed, M. Ahmed, T. Muhammad, M. Afzal, M. Arif, and B. Bazezew, “Ga-irace: Genetic algorithm-based improved resource aware cost-efficient scheduler for cloud fog computing environment,” *Wireless Communications and Mobile Computing*, vol. 2022, 2022.

[23] S. Yi, Z. Hao, Z. Qin, and Q. Li, “Fog computing: Platform and applications,” in *2015 Third IEEE workshop on hot topics in web systems and technologies (HotWeb)*, pp. 73–78, IEEE, 2015.

[24] M. Taneja and A. Davy, “Resource aware placement of iot application modules in fog-cloud computing paradigm,” in *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pp. 1222–1228, IEEE, 2017.

[25] X.-Q. Pham and E.-N. Huh, “Towards task scheduling in a cloud-fog computing system,” in *2016 18th Asia-Pacific network operations and management symposium (APNOMS)*, pp. 1–4, IEEE, 2016.

[26] V. Cardellini, V. Grassi, F. Lo Presti, and M. Nardelli, “On qos-aware scheduling of data stream applications over fog computing infrastructures,” 07 2015.

[27] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, “Edge computing: Vision and challenges,” *IEEE internet of things journal*, vol. 3, no. 5, pp. 637–646, 2016.

[28] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, “Mobile edge computing—a key technology towards 5g,” *ETSI white paper*, vol. 11, no. 11, pp. 1–16, 2015.

[29] E. Cau, M. Corici, P. Bellavista, L. Foschini, G. Carella, A. Edmonds, and T. M. Bohnert, “Efficient exploitation of mobile edge computing for virtualized 5g in epc architectures,” in *2016 4th IEEE international conference on mobile cloud computing, services, and engineering (MobileCloud)*, pp. 100–109, IEEE, 2016.

[30] Z. Sanaei, S. Abolfazli, A. Gani, and R. Buyya, “Heterogeneity in mobile cloud computing: Taxonomy and open challenges,” *IEEE Communications Surveys Tutorials*, vol. 16, no. 1, pp. 369–392, 2014.

[31] M. Satyanarayanan, G. Lewis, E. Morris, S. Simanta, J. Boleng, and K. Ha, “The role of cloudlets in hostile environments,” *IEEE Pervasive Computing*, vol. 12, no. 4, pp. 40–49, 2013.

[32] M. Aazam, S. Zeadally, and K. A. Harras, “Fog computing architecture, evaluation, and future research directions,” *IEEE Communications Magazine*, vol. 56, no. 5, pp. 46–52, 2018.

[33] M. Aazam and E.-N. Huh, “Fog computing micro datacenter based dynamic resource estimation and pricing model for iot,” in *2015 IEEE 29th International Conference on Advanced Information Networking and Applications*, pp. 687–694, IEEE, 2015.

[34] H. R. Arkian, A. Diyanat, and A. Pourkhalili, "Mist: Fog-based data analytics scheme with cost-efficient resource provisioning for iot crowdsensing applications," *Journal of Network and Computer Applications*, vol. 82, pp. 152–165, 2017.

[35] T. H. Luan, L. Gao, Z. Li, Y. Xiang, G. Wei, and L. Sun, "Fog computing: Focusing on mobile users at the edge," *arXiv preprint arXiv:1502.01815*, 2015.

[36] N. K. Giang, M. Blackstock, R. Lea, and V. C. Leung, "Developing iot applications in the fog: A distributed dataflow approach," in *2015 5th International Conference on the Internet of Things (IOT)*, pp. 155–162, IEEE, 2015.

[37] O. C. A. W. Group *et al.*, "Openfog architecture overview," *White Paper OPFWP001*, vol. 216, p. 35, 2016.

[38] K. Intharawijitr, K. Iida, and H. Koga, "Analysis of fog model considering computing and communication latency in 5g cellular networks," in *2016 IEEE international conference on pervasive computing and communication workshops (PerCom workshops)*, pp. 1–4, IEEE, 2016.

[39] M. A. Nadeem and M. A. Saeed, "Fog computing: An emerging paradigm," in *2016 Sixth International Conference on Innovative Computing Technology (INTECH)*, pp. 83–86, IEEE, 2016.

[40] M. Taneja and A. Davy, "Resource aware placement of data analytics platform in fog computing," *Procedia Computer Science*, vol. 97, pp. 153–156, 2016.

[41] W. Zhang, Z. Zhang, and H.-C. Chao, "Cooperative fog computing for dealing with big data in the internet of vehicles: Architecture and hierarchical resource management," *IEEE Communications Magazine*, vol. 55, no. 12, pp. 60–67, 2017.

[42] H. Sabireen and V. Neelanarayanan, "A review on fog computing: Architecture, fog with iot, algorithms and research challenges," *Ict Express*, vol. 7, no. 2, pp. 162–176, 2021.

[43] H. F. Atlam, R. J. Walters, and G. B. Wills, "Fog computing and the internet of things: A review," *big data and cognitive computing*, vol. 2, no. 2, p. 10, 2018.

[44] P. Habibi, M. Farhoudi, S. Kazemian, S. Khorsandi, and A. Leon-Garcia, "Fog computing: a comprehensive architectural survey," *IEEE Access*, vol. 8, pp. 69105–69133, 2020.

[45] A. Paul, H. Pinjari, W.-H. Hong, H. C. Seo, and S. Rho, "Fog computing-based iot for health monitoring system," *Journal of Sensors*, vol. 2018, 2018.

[46] H. Sabireen and V. Neelanarayanan, "A review on fog computing: Architecture, fog with iot, algorithms and research challenges," *Ict Express*, vol. 7, no. 2, pp. 162–176, 2021.

[47] K. Tange, M. De Donno, X. Fafoutis, and N. Dragoni, "A systematic survey of industrial internet of things security: Requirements and fog computing opportunities," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 4, pp. 2489–2520, 2020.

[48] R. K. Naha, S. Garg, and A. Chan, "Fog computing architecture: Survey and challenges," *arXiv preprint arXiv:1811.09047*, 2018.

[49] M. M. Hussain and M. S. Beg, "Fog computing for internet of things (iot)-aided smart grid architectures," *Big Data and cognitive computing*, vol. 3, no. 1, p. 8, 2019.

[50] M. A. Nadeem and M. A. Saeed, "Fog computing: An emerging paradigm," in *2016 Sixth International Conference on Innovative Computing Technology (INTECH)*, pp. 83–86, IEEE, 2016.

[51] S. Thiruchadai Pandeeswari and S. Padmavathi, “Fog based architectures for iot: survey on motivations, challenges and solution perspectives,” in *International Conference on Remote Engineering and Virtual Instrumentation*, pp. 298–305, Springer, 2019.

[52] S. Yi, C. Li, and Q. Li, “A survey of fog computing: concepts, applications and issues,” in *Proceedings of the 2015 workshop on mobile big data*, pp. 37–42, 2015.

[53] B. Jamil, M. Shojafar, I. Ahmed, A. Ullah, K. Munir, and H. Ijaz, “A job scheduling algorithm for delay and performance optimization in fog computing,” *Concurrency and Computation: Practice and Experience*, vol. 32, no. 7, p. e5581, 2020.

[54] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, “Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading,” in *2012 Proceedings IEEE Infocom*, pp. 945–953, IEEE, 2012.

[55] K. Kai, W. Cong, and L. Tao, “Fog computing for vehicular ad-hoc networks: paradigms, scenarios, and issues,” *the journal of China Universities of Posts and Telecommunications*, vol. 23, no. 2, pp. 56–96, 2016.

[56] C. Chen, T. Qiu, J. Hu, Z. Ren, Y. Zhou, and A. K. Sangaiah, “A congestion avoidance game for information exchange on intersections in heterogeneous vehicular networks,” *Journal of Network and Computer Applications*, vol. 85, pp. 116–126, 2017.

[57] X. Hou, Y. Li, M. Chen, D. Wu, D. Jin, and S. Chen, “Vehicular fog computing: A viewpoint of vehicles as the infrastructures,” *IEEE Transactions on Vehicular Technology*, vol. 65, no. 6, pp. 3860–3873, 2016.

[58] F. Bonomi, “Connected vehicles, the internet of things, and fog computing,” in *The eighth ACM international workshop on vehicular inter-networking (VANET), Las Vegas, USA*, pp. 13–15, sn, 2011.

[59] R. Karasik, O. Simeone, and S. S. Shitz, "How much can d2d communication reduce content delivery latency in fog networks with edge caching?," *IEEE Transactions on Communications*, vol. 68, no. 4, pp. 2308–2323, 2019.

[60] S. Singh and A. Yassine, "Iot big data analytics with fog computing for household energy management in smart grids," in *International Conference on Smart Grid and Internet of Things*, pp. 13–22, Springer, 2018.

[61] T. M. Fernández-Caramés and P. Fraga-Lamas, "Towards next generation teaching, learning, and context-aware applications for higher education: A review on blockchain, iot, fog and edge computing enabled smart campuses and universities," *Applied Sciences*, vol. 9, no. 21, p. 4479, 2019.

[62] S. Rani, A. Kataria, and M. Chauhan, "Fog computing in industry 4.0: Applications and challenges—a research roadmap," *Energy Conservation Solutions for Fog-Edge Computing Paradigms*, pp. 173–190, 2022.

[63] P. Hu, H. Ning, T. Qiu, Y. Zhang, and X. Luo, "Fog computing based face identification and resolution scheme in internet of things," *IEEE transactions on industrial informatics*, vol. 13, no. 4, pp. 1910–1920, 2016.

[64] P. Varshney and Y. Simmhan, "Demystifying fog computing: Characterizing architectures, applications and abstractions," in *2017 IEEE 1st international conference on fog and edge computing (ICFEC)*, pp. 115–124, IEEE, 2017.

[65] M. S. Hossain and M. Atiquzzaman, "Cost analysis of mobility protocols," *Telecommunication Systems*, vol. 52, no. 4, pp. 2271–2285, 2013.

[66] A. Rodríguez Natal, L. Jakab, M. Portolés, V. Ermagan, P. Natarajan, F. Maino, D. Meyer, and A. Cabellos Aparicio, "Lisp-mn: mobile networking through lisp," *Wireless personal communications*, vol. 70, no. 1, pp. 253–266, 2013.

[67] A. Natraj, "Fog computing focusing on users at the edge of internet of things," *International Journal of Engineering Research*, vol. 5, no. 5, pp. 1004–1008, 2016.

[68] K. Lee, D. Kim, D. Ha, U. Rajput, and H. Oh, "On security and privacy issues of fog computing supported internet of things environment," in *2015 6th International Conference on the Network of the Future (NOF)*, pp. 1–3, IEEE, 2015.

[69] Y. Zhang, D. Niyato, P. Wang, and D. I. Kim, "Optimal energy management policy of mobile energy gateway," *IEEE Transactions on Vehicular Technology*, vol. 65, no. 5, pp. 3685–3699, 2015.

[70] L. Bittencourt, J. Diaz-Montes, R. Buyya, O. Rana, and M. Parashar, "Mobility-aware application scheduling in fog computing," 2017. Publisher Copyright: © 2014 IEEE.

[71] L. Ni, J. Zhang, C. Jiang, C. Yan, and K. Yu, "Resource allocation strategy in fog computing based on priced timed petri nets," *IEEE Internet of Things Journal*, vol. 4, no. 5, pp. 1216–1228, 2017.

[72] D. Rahbari and M. Nickray, "Scheduling of fog networks with optimized knapsack by symbiotic organisms search," in *2017 21st Conference of Open Innovations Association (FRUCT)*, pp. 278–283, 2017.

[73] X.-Q. Pham, N. D. Man, N. D. T. Tri, N. Q. Thai, and E.-N. Huh, "A cost- and performance-effective approach for task scheduling based on collaboration between cloud and fog computing," *International Journal of Distributed Sensor Networks*, vol. 13, no. 11, p. 1550147717742073, 2017.

[74] S. Agarwal, S. Yadav, and A. Yadav, "An efficient architecture and algorithm for resource provisioning in fog computing," *International Journal of Information Engineering and Electronic Business*, vol. 8, pp. 48–61, 01 2016.

[75] T. Choudhari, M. Moh, and T.-S. Moh, "Prioritized task scheduling in fog computing," in *Proceedings of the ACMSE 2018 Conference*, ACMSE '18, (New York, NY, USA), Association for Computing Machinery, 2018.

[76] S. Bitam, S. Zeadally, and A. Mellouk, “Fog computing job scheduling optimization based on bees swarm,” *Enterprise Information Systems*, vol. 12, pp. 1–25, 04 2017.

[77] Y. Sun, F. Lin, and H. Xu, “Multi-objective optimization of resource scheduling in fog computing using an improved nsga-ii,” *Wireless Personal Communications*, vol. 102, no. 2, pp. 1369–1385, 2018.

[78] D. Zeng, L. Gu, S. Guo, Z. Cheng, and S. Yu, “Joint optimization of task scheduling and image placement in fog computing supported software-defined embedded system,” *IEEE Transactions on Computers*, vol. 65, pp. 1–1, 12 2016.

[79] R. K. Naha, S. Garg, A. Chan, and S. K. Battula, “Deadline-based dynamic resource allocation and provisioning algorithms in fog-cloud environment,” *Future Generation Computer Systems*, vol. 104, pp. 131–141, 2020.

[80] R. M. Abdelmoneem, A. Benslimane, and E. Shaaban, “Mobility-aware task scheduling in cloud-fog iot-based healthcare architectures,” *Computer Networks*, vol. 179, p. 107348, 2020.

[81] P. Hosseinioun, M. Kheirabadi, S. R. K. Tabbakh, and R. Ghaemi, “A new energy-aware tasks scheduling approach in fog computing using hybrid meta-heuristic algorithm,” *Journal of Parallel and Distributed Computing*, vol. 143, pp. 88–96, 2020.

[82] F. Hoseiny, S. Azizi, M. Shojafar, F. Ahmadiazar, and R. Tafazolli, “Pga: a priority-aware genetic algorithm for task scheduling in heterogeneous fog-cloud computing,” in *IEEE INFOCOM 2021-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 1–6, IEEE, 2021.

[83] Z. Pooranian, M. Shojafar, P. G. V. Naranjo, L. Chiaraviglio, and M. Conti, “A novel distributed fog-based networked architecture to preserve energy in fog data centers,” in *2017 IEEE 14th International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*, pp. 604–609, IEEE, 2017.

[84] D. Hoang and T. D. Dang, "Fbrc: Optimization of task scheduling in fog-based region and cloud," in *2017 IEEE Trustcom/BigDataSE/ICESS*, pp. 1109–1114, IEEE, 2017.

[85] X. Sun and N. Ansari, "Primal: Profit maximization avatar placement for mobile edge computing," in *2016 IEEE International Conference on Communications (ICC)*, pp. 1–6, 2016.

[86] J. Fan, X. Wei, T. Wang, T. Lan, and S. Subramaniam, "Deadline-aware task scheduling in a tiered iot infrastructure," pp. 1–7, 12 2017.

[87] J. Xu, B. Palanisamy, H. Ludwig, and Q. Wang, "Zenith: Utility-aware resource allocation for edge computing," in *2017 IEEE international conference on edge computing (EDGE)*, pp. 47–54, IEEE, 2017.

[88] L. Liu, D. Qi, N. Zhou, and Y. Wu, "A task scheduling algorithm based on classification mining in fog computing environment," *Wireless Communications and Mobile Computing*, vol. 2018, pp. 1–11, 08 2018.

[89] X.-Q. Pham, N. D. Man, N. D. T. Tri, N. Q. Thai, and E.-N. Huh, "A cost- and performance-effective approach for task scheduling based on collaboration between cloud and fog computing," *International Journal of Distributed Sensor Networks*, vol. 13, no. 11, p. 1550147717742073, 2017.

[90] H. Gupta, A. Vahid Dastjerdi, S. K. Ghosh, and R. Buyya, "ifogsim: A toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments," *Software: Practice and Experience*, vol. 47, no. 9, pp. 1275–1296, 2017.

[91] S. Bitam, S. Zeadally, and A. Mellouk, "Fog computing job scheduling optimization based on bees swarm," *Enterprise Information Systems*, vol. 12, pp. 1–25, 04 2017.

[92] K. Gai and M. Qiu, "Optimal resource allocation using reinforcement learning for iot content-centric services," *Appl. Soft Comput.*, vol. 70, pp. 12–21, 2018.

[93] Q. Zhang, M. Lin, L. T. Yang, Z. Chen, S. U. Khan, and P. Li, "A double deep q-learning model for energy-efficient edge scheduling," *IEEE Transactions on Services Computing*, vol. 12, pp. 739–749, sep 2019.

[94] D. Rahbari and M. Nickray, "Low-latency and energy-efficient scheduling in fog-based iot applications," *Turkish Journal of Electrical Engineering and Computer Sciences*, vol. 27, no. 2, pp. 1406–1427, 2019.

[95] L. Mai, N.-N. Dao, and M. Park, "Real-time task assignment approach leveraging reinforcement learning with evolution strategies for long-term latency minimization in fog computing," *Sensors*, vol. 18, no. 9, 2018.

[96] A. Yassine, S. Singh, M. S. Hossain, and G. Muhammad, "Iot big data analytics for smart homes with fog and cloud computing," *Future Generation Computer Systems*, vol. 91, pp. 563–573, 2019.

[97] S. Kabirzadeh, D. Rahbari, and M. Nickray, "A hyper heuristic algorithm for scheduling of fog networks," in *2017 21st Conference of Open Innovations Association (FRUCT)*, pp. 148–155, IEEE, 2017.

[98] L. Yin, J. Luo, and H. Luo, "Tasks scheduling and resource allocation in fog computing based on containers for smart manufacturing," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 10, pp. 4712–4721, 2018.

[99] M. A. Rodriguez and R. Buyya, "A responsive knapsack-based algorithm for resource provisioning and scheduling of scientific workflows in clouds," in *2015 44th International Conference on Parallel Processing*, pp. 839–848, IEEE, 2015.

[100] H. S. Ali, R. R. Rout, P. Parimi, and S. K. Das, "Real-time task scheduling in fog-cloud computing framework for iot applications: a fuzzy logic based approach," in *2021 International Conference on COMmunication Systems & NETworkS (COMSNETS)*, pp. 556–564, IEEE, 2021.

[101] J. Xu, Z. Hao, R. Zhang, and X. Sun, “A method based on the combination of laxity and ant colony system for cloud-fog task scheduling,” *IEEE Access*, vol. 7, pp. 116218–116226, 2019.

[102] A. Yousefpour, A. Patil, G. Ishigaki, I. Kim, X. Wang, H. C. Cankaya, Q. Zhang, W. Xie, and J. P. Jue, “Qos-aware dynamic fog service provisioning,” *arXiv preprint arXiv:1802.00800*, 2018.

[103] S. Smith, “‘internet of things’ connected devices to almost triple to over 38 billion units by 2020 - juniper,” 2016.

[104] M. Ibrahim, S. Nabi, A. Baz, H. Alhakami, M. Raza, A. Hussain, K. Salah, and K. Djemame, “An in-depth empirical investigation of state-of-the-art scheduling approaches for cloud computing,” *IEEE Access*, vol. PP, pp. 1–1, 07 2020.

[105] A. Hussain, M. Aleem, A. Khan, M. Iqbal, and A. Islam, “Ralba: a computation-aware load balancing scheduler for cloud computing,” *Cluster Computing*, vol. 21, 09 2018.

[106] Sandvine, “Sandvine global internet phenomena report 2h-2015,sandvine corp., p. 17, 2015,doi:10.1007/s13398-014-0173-7.2,” 2015.

[107] J. Cao, Q. Zhang, and W. Shi, “Challenges and opportunities in edge computing,” *Edge Computing: A Primer*, pp. 59–70, 2018.

[108] A. T. Thien and R. Colomo-Palacios, “A systematic literature review of fog computing,” *Nokobit*, vol. 24, no. 1, pp. 28–30, 2016.

[109] P. Hu, S. Dhelim, H. Ning, and T. Qiu, “Survey on fog computing: architecture, key technologies, applications and open issues,” *Journal of Network and Computer Applications*, vol. 98, pp. 27–42, 2017.

[110] M. Hajibaba and S. Gorgin, “A review on modern distributed computing paradigms: Cloud computing, jungle computing and fog computing,” *Journal of Computing and Information Technology*, vol. 22, p. 69, 01 2014.

[111] I. Stojmenovic, S. Wen, X. Huang, and H. Luan, “An overview of fog computing and its security issues,” *Concurrency and Computation: Practice and Experience*, vol. 28, no. 10, pp. 2991–3005, 2016.

[112] E. Kavvadia, S. Sagiadinos, K. Oikonomou, G. Tsioutsiouliklis, and S. Aïssa, “Elastic virtual machine placement in cloud computing network environments,” *Comput. Netw.*, vol. 93, p. 435–447, dec 2015.

[113] R. Mahmud, R. Kotagiri, and R. Buyya, *Fog Computing: A Taxonomy, Survey and Future Directions*, pp. 103–130. Singapore: Springer Singapore, 2018.

[114] “Advancing the state of mobile cloud computing,” in *MCS’12 - Proceedings of the 3rd ACM Workshop on Mobile Cloud Computing and Services*, MCS’12 - Proceedings of the 3rd ACM Workshop on Mobile Cloud Computing and Services, pp. 21–27, Association for Computing Machinery, Inc, 2012. 3rd ACM Workshop on Mobile Cloud Computing and Services, MCS’12 ; Conference date: 25-06-2012 Through 25-06-2012.

[115] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. D. Rose, and R. Buyya, “Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms,” *Software: Practice and Experience*, vol. 41, 2011.

[116] W. Lin, C. Liang, J. Z. Wang, and R. Buyya, “Bandwidth-aware divisible task scheduling for cloud computing,” *Software: Practice and Experience*, vol. 44, no. 2, pp. 163–174, 2014.

[117] M. Mahmud, M. Afrin, M. A. Razzaque, M. Hassan, A. Alelaiwi, and M. Alrubaian, “Maximizing quality of experience through context-aware mobile application scheduling in cloudlet infrastructure,” *Software: Practice and Experience*, vol. 46, pp. n/a–n/a, 11 2016.

[118] M. Arif, G. Wang, M. Z. A. Bhuiyan, T. Wang, and J. Chen, “A survey on security attacks in vanets: Communication, applications and challenges,” *Vehicular Communications*, vol. 19, p. 100179, 2019.

[119] M. Arif, G. Wang, V. E. Balas, O. Geman, A. Castiglione, and J. Chen, "Sdn based communications privacy-preserving architecture for vanets using fog computing," *Vehicular Communications*, vol. 26, p. 100265, 2020.

[120] T. Wang, Y. Liang, Y. Zhang, X. Zheng, M. Arif, J. Wang, and Q. Jin, "An intelligent dynamic offloading from cloud to edge for smart iot systems with big data," *IEEE Transactions on Network Science and Engineering*, vol. 7, no. 4, pp. 2598–2607, 2020.

[121] E. Baccarelli, P. G. V. Naranjo, M. Scarpiniti, M. Shojafar, and J. H. Abawajy, "Fog of everything: Energy-efficient networked computing architectures, research challenges, and a case study," *IEEE access*, vol. 5, pp. 9882–9910, 2017.

[122] M. Arif, G. Wang, T. Peng, V. E. Balas, O. Geman, and J. Chen, "Optimization of communication in vanets using fuzzy logic and artificial bee colony," *Journal of Intelligent & Fuzzy Systems*, vol. 38, no. 5, pp. 6145–6157, 2020.

[123] J. Manyika, M. Chui, P. Bisson, J. Woetzel, R. Dobbs, J. Bughin, and D. Aharon, "Unlocking the potential of the internet of things," Feb 2020.

[124] G. Patrick, "Iot and the network: What is the future?." `https://blogs.cisco.com/networking/iot-and-the-network-what-is-the-future`, May 2013.

[125] S. Nabi and M. Ahmed, "Og-radl: overall performance-based resource-aware dynamic load-balancer for deadline constrained cloud tasks," *The Journal of Supercomputing*, vol. 77, 07 2021.

[126] S. Nabi, M. Ahmad, M. Ibrahim, and H. Hamam, "Adpso: adaptive pso-based task scheduling approach for cloud computing," *Sensors*, vol. 22, no. 3, p. 920, 2022.

[127] J. He, J. Wei, K. Chen, Z. Tang, Y. Zhou, and Y. Zhang, "Multitier fog computing with large-scale iot data analytics for smart cities," *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 677–686, 2018.

[128] L. M. Vaquero and L. Rodero-Merino, "Finding your way in the fog: Towards a comprehensive definition of fog computing," *SIGCOMM Comput. Commun. Rev.*, vol. 44, p. 27–32, oct 2014.

[129] X. Lyu, C. Ren, W. Ni, H. Tian, and R. Liu, "Distributed optimization of collaborative regions in large-scale inhomogeneous fog computing," *IEEE Journal on Selected Areas in Communications*, vol. PP, 02 2018.

[130] M. Shojafar, N. Cordeschi, and E. Baccarelli, "Energy-efficient adaptive resource management for real-time vehicular cloud services," *IEEE Transactions on Cloud Computing*, vol. 7, pp. 196–209, 03 2019.

[131] M. Arif, J. Chen, G. Wang, O. Geman, and V. E. Balas, "Privacy preserving and data publication for vehicular trajectories with differential privacy," *Measurement*, vol. 173, p. 108675, 2021.

[132] M. Arif, W. Balzano, A. Fontanella, S. Stranieri, G. Wang, and X. Xing, "Integration of 5g, vanets and blockchain technology," in *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pp. 2007–2013, IEEE, 2020.

[133] W. Tärneberg, A. Mehta, E. Wadbro, J. Tordsson, J. Eker, M. Kihl, and E. Elmroth, "Dynamic application placement in the mobile cloud network," *Future Generation Computer Systems*, vol. 70, 07 2016.

[134] N. Khan, I. A. Khan, J. U. Arshed, M. Afzal, M. M. Ahmed, and M. Arif, "5g-eecc: Energy-efficient collaboration-based content sharing strategy in device-to-device communication," *Security and Communication Networks*, vol. 2022, 2022.

[135] J. H. Holland, *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence.* MIT press, 1992.

[136] S. G. Ahmad, E. U. Munir, and W. Nisar, "Pega: A performance effective genetic algorithm for task scheduling in heterogeneous systems," in *2012*

IEEE 14th International Conference on High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems, pp. 1082–1087, IEEE, 2012.