Smart Task Distribution in Combined Fog-Cloud Scenarios

Dissertation
zur
Erlangung des akademischen Grades
Doktor-Ingenieur (Dr.-Ing.)
der Fakultät für Informatik und Elektrotechnik
der Universität Rostock

vorgelegt von
Mohammadreza Pourkiani, geboren am 15.09.1991 in Mashhad, Iran

Rostock, 01.05.2022

https://doi.org/10.18453/rosdok_id00004357

Gutachter:

Prof. Dr. Clemens Cap, Universität Rostock
Prof. Dr. Gero Mühl, Universität Rostock

Mehdi Assefi, PCN Bank Technology Center

Jahr der Einreichung: 2022
Jahr der Verteidigung: 2023

Abstract

In order to collect data, most of the IoT-based applications utilize sensors, which are limited in terms of computational and storage capabilities. Therefore, the collected raw data by the IoT sensors must be transmitted to capable servers for processing, storage, and data mining purposes. Fog and Cloud computing are two leading technologies which can provide computation and storage services for IoT-based applications. Cloud provides powerful servers, which are located far from the users and have high latency, while Fog provides servers with limited computational power in the proximity of the users with low latency. As there are different delay-sensitive and delay-tolerable applications in the world of the IoT, utilization of only Fog or Cloud can not be a perfect approach for all of the scenarios.

Moreover, task distribution between the fog and cloud servers is challenging as in combined fog-cloud scenarios, there are various types of servers, which are heterogeneous in terms of hardware, delay, workload, and computational power. This heterogeneity makes the selection process of the most suitable server at each time slice very difficult. In this thesis, our main goal is to find a solution for resource allocation in combined fog-cloud scenarios with regard to the application requirements. For this purpose, we proposed an intelligent task assignment algorithm (MLTD) that runs in the task distributor unit. This algorithm takes the diversity of servers into account, considers the application requirements, and uses machine learning methods for estimating the response times of the fog and cloud servers, in addition to the size of the data that must be communicated over the Internet. By utilizing this method, after receiving the raw data from the sensors, the task distributor unit selects the most suitable server at that time-slice and assigns the received tasks to that server to be processed.

In order to investigate the performance of MLTD, we used that for distributing the tasks of a delay-tolerable application (that solves mathematical questions) and a delay-sensitive application (that provides online healthcare services and utilizes the wireless body sensor networks for data collection to monitor the health status of people who work in environments with high levels of heat stress, such as the steel and iron industries). For distributing the tasks of the discussed applications between the fog and cloud servers, we utilized the Artificial Neural Networks (as the function approximation method) in the task distributor unit. To train the neural networks, we generated different numbers of tasks and ran them on all of the fog and cloud servers. For the training process, we set the response times of servers as the output, and the parameters of the tasks as the input of the neural networks. In the next step, we set the trained neural networks in the broker and therefore made the broker able to select the fastest server for processing a received task from the IoT sensors. We also added more parameters in the training process of the neural networks in different situations to make our proposed method scalable and usable in different network architectures.

The performance of MLTD has been investigated in different experiments. The achieved results show that this technique can reduce the Internet bandwidth utilization, response time, and resource utilization compared to other proposed methods in the state of the art. The reason is that our proposed technique (unlike the other discussed methods in the state of the art) can predict the response times of available servers, in addition to the future Internet bandwidth utilization at the time of task arrival (before the distribution process). Therefore, MLTD can distribute the tasks with regard to the requirements of applications in terms of response time or bandwidth utilization.

However, we observed that the performance of MLTD entirely depends on the precision of the utilized function approximation methods, which can be affected by using different types of tasks, training methods, and richness of training. Moreover, we also witnessed that our smart task distribution technique performs excellently when the fog and cloud servers provide response times with a difference of more than the error of the utilized function approximation method for predicting the response times.

Abstrakt

Zur Datenerfassung verwenden die meisten IoT-basierten Anwendungen Sensoren, deren Rechen- und Speicherkapazitäten begrenzt sind. Daher müssen die gesammelten Rohdaten von den IoT-Sensoren an fähige Server zur Verarbeitung, Speicherung und zum Data Mining übertragen werden. Fog und Cloud Computing sind zwei führende Technologien, die Berechnungs- und Speicherdienste für IoT-basierte Anwendungen bereitstellen können. Die Cloud bietet leistungsstarke Server, die weit von den Nutzern entfernt sind und eine hohe Latenz aufweisen, während Fog Server begrenzter Rechenleistung in der Nähe der Nutzer mit geringer Latenz bereitstellt. Da es in der Welt des IoT verschiedene verzögerungsempfindliche und verzögerungstolerante Anwendungen gibt, kann die Nutzung von Fog oder Cloud nicht für alle Szenarien ein perfekter Ansatz sein.

Darüber hinaus ist die Aufgabenverteilung zwischen den Fog- und Cloud-Servern eine Herausforderung, da es in kombinierten Fog-Cloud-Szenarien verschiedene Arten von Servern gibt, die in Bezug auf Hardware, Verzögerung, Arbeitslast und Rechenleistung heterogen sind. Diese Heterogenität macht die Auswahl des am besten geeigneten Servers in jeder Zeitscheibe sehr schwierig. Das Hauptziel in dieser Arbeit ist es, eine Lösung für die Ressourcenzuweisung in kombinierten Fog-Cloud-Szenarien unter Berücksichtigung der Anwendungsanforderungen zu finden. Zu diesem Zweck haben wir einen intelligenten Aufgabenzuweisungsalgorithmus (MLTD) vorgeschlagen, der in der Aufgabenverteilungseinheit läuft. Dieser Algorithmus berücksichtigt die Vielfalt der Server, die Anforderungen der Anwendung und nutzt Methoden des maschinellen Lernens, um die Antwortzeiten der Fog- und Cloud-Server sowie die Größe der über das Internet zu übermittelnden Daten zu schätzen. Mit dieser Methode wählt die Aufgabenverteilereinheit nach dem Empfang der Rohdaten von den Sensoren den zu diesem Zeitpunkt am besten geeigneten Server aus und weist die empfangenen Aufgaben diesem Server zur Bearbeitung zu.

Um die Leistung von MLTD zu untersuchen, haben wir es für die Verteilung der Aufgaben einer verzögerungstoleranten Anwendung (die mathematische Fragen löst) und einer verzögerungsempfindlichen Anwendung (die Online-Gesundheitsdienste anbietet und die Wireless-Body-Sensornetzwerke für die Datenerfassung nutzt, um den Gesundheitszustand von Menschen zu überwachen, die in Umgebungen mit hohem Hitzestress arbeiten, z. B. in der Stahl- und Eisenindustrie) eingesetzt. Um die Aufgaben der besprochenen Anwendungen zwischen den Fog und Cloud-Servern zu verteilen, haben wir Artificial Neural Networks (als Methode zur Funktionsannäherung) in der Aufgabenverteilungseinheit eingesetzt. Um die Neural Networks zu trainieren, haben wir eine unterschiedliche Anzahl von Aufgaben generiert und sie auf allen Fog- und Cloud-Servern ausgeführt. Für den Trainingsprozess haben wir die Antwortzeiten der Server als Ausgabe und die

Parameter der Aufgaben als Eingabe der neuronalen Netze festgelegt. Im nächsten Schritt setzten wir die trainierten neuronalen Netze in den Broker ein, so dass dieser in der Lage war, den schnellsten Server für die Verarbeitung einer empfangenen Aufgabe von den IoT-Sensoren auszuwählen. Wir fügten auch weitere Parameter in den Trainingsprozess der neuronalen Netze in verschiedenen Situationen ein, um unsere vorgeschlagene Methode skalierbar und in verschiedenen Netzwerkarchitekturen verwendbar zu machen.

Die Leistung von MLTD wurde in verschiedenen Experimenten untersucht. Die erzielten Ergebnisse zeigen, dass diese Technik die Internet-Bandbreitennutzung, die Antwortzeit und die Ressourcennutzung im Vergleich zu anderen vorgeschlagenen Methoden in früheren Arbeiten reduzieren kann. Der Grund dafür ist, dass die von uns vorgeschlagene Technik (im Gegensatz zu den anderen vorgeschlagenen Methoden) die Antwortzeiten der verfügbaren Server sowie die künftige Auslastung der Internetbandbreite zum Zeitpunkt der Ankunft der Aufgabe (vor dem Verteilungsprozess) vorhersagen kann. Daher kann MLTD die Aufgaben im Hinblick auf die Anforderungen der Anwendungen in Bezug auf die Antwortzeit oder die Bandbreitennutzung verteilen.

Wir haben jedoch festgestellt, dass die Leistung von MLTD vollständig von der Genauigkeit der verwendeten Funktionsapproximationsmethoden abhängt, die durch die Verwendung verschiedener Aufgabentypen, Trainingsmethoden und die Reichhaltigkeit des Trainings beeinflusst werden kann. Darüber hinaus konnten wir feststellen, dass unsere intelligente Aufgabenverteilungstechnik hervorragend funktioniert, wenn die Antwortzeiten von Fog- und Cloud-Servern um mehr als den Fehler der für die Vorhersage der Antwortzeiten verwendeten Funktionsapproximationsmethode abweichen.

Acknowledgment

This work would not have been possible without the support of various people. First of all, I thank my supervisor Prof. Dr. Clemens Cap, for his support, guidance, and patience during my Ph.D. studies. I am also very grateful to Prof. Dr. Gero Mühl for helping me by making comments on my works and for his support for the extension of my scholarship. I am also thankful to Mr. Masoud Abedi, my friend and co-author in more than six papers. In addition, I would like to thank the University of Rostock for funding my Ph.D. research through the Mecklenburg-Vorpommern State Scholarship Program.

And most of all, I thank my parents, family members, and friends for their love and support.

Contents

List of Acronyms

AHP	Analytic Hierarchy Process
AI	Artificial Intelligence
AITDA	Artificial Intelligence Based Task Distribution Algorithm
ANN	Artificial Neural Networks
DC	Data Center
DFDW	Different Fog Servers with Different Workloads
DFSW	Different Fog Servers with Similar Workloads
DSSP	Different Sizes Similar Processing Times
DSP	Different Sizes and Processing Times
F2C	Fog to Cloud Architecture
FAM	Function Approximation Method
FCSTD	Fog-Cloud Smart Task Distribution
GA	Genetic Algorithm
GPRFCA	Gaussian Process Regression for Fog-Cloud Allocation
IBU	Internet Bandwidth Utilization
IoT	Internet of Things
ITS	Intelligent Transportation System
ITU	International Telecommunication Union
LAN	Local Area Network
LM	Levenburg Marquardt
MANET	Mobile Adhoc Network
MLTD	Machine Learning Based Task Distribution
NIST	National Institute of Standards and Technology
OFC	Open Fog Consortium
OF2C	Optimized Fog to Cloud Architecture
QoS	Quality of Service
SFDW	Similar Fog Servers with Different Workloads
SFSW	Similar Fog Servers with Similar Workloads
SSDP	Similar Sizes Different Processing Times
SSP	Similar Sizes and Processing Times
TSWC	Number of Time Slices in which the Workloads Change
VFR	Virtual Fog Resolver
WBAN	Wireless Body Area Network

WBSN	Wireless Body Sensor Network

1 Introduction

1.1 Motivation

IoT (Internet of Things)-based applications utilize different types of sensors and actuators for data collection and making changes in the environments. It must be noticed that the sensors collect raw data, which needs to be processed by capable servers to be available for users as meaningful information [1]. In IoT-based applications, one or several sensors can be used to collect data from an environment. For example, healthcare applications use environmental sensors to collect information about the parameters of an environment, such as temperature, pollution, and humidity. Healthcare applications also use medical sensors, which are used to monitor the patient's vital signs such as heartbeat, respiration rate [2], body temperature, blood pressure, and so on (more information about the different types of sensors is presented in [3]).

The development of medical sensors has led to the emergence of Wireless Body Area Network (WBAN), a key technology in e-health applications. WBANs are composed of implanted or wearable sensors that collect biomedical data continuously [4]. This continuous monitoring provides many advantages, such as quick detection of any abnormal sign in the human body. This is a significant advantage because fatal diseases such as cardiovascular problems are usually diagnosed too late, and this late diagnosis increases the death rate of patients [5]. Fortunately, with the advent of WBANs and other medical applications, early detection of diseases is now possible.

WBAN-based healthcare applications are considered as an important application of IoT. As we discussed earlier and as is mentioned in [6, 7, 8, 9, 10], provisioning the Quality of Service (QoS), especially reducing the response time (even a few milliseconds), and Internet bandwidth utilization (because of its impact on cost and quality of data communication) in these applications is of great importance. Therefore, in this thesis, our motivation is to improve the response time (as a critical QoS parameter) and Internet Bandwidth Utilization (IBU) for WBAN-based applications to improve their performance. The state of the Art shows the importance of task distribution between the Fog and Cloud resources, which has been discussed in [11, 12, 13, 14], but the impact of predicting the processing times of tasks on reducing the response time has not been considered. Therefore we aim to reduce the response time by proposing an Artificial Intelligence (AI) based resource provisioning algorithm, which is able to predict the processing times of tasks and their generated data, and runs in combined fog-cloud scenarios, and distributes the generated tasks of WBANs. It must be mentioned that besides reducing the response time (which is our main goal), our proposed resource provisioning algorithm also aims to reduce the IBU and resource utilization. As another advantage, it can be said that our proposed method is not restricted to healthcare applications and can be used for the task distribution of any delay-sensitive applications (where the AI-based method can predict the processing times of the tasks properly).

1.2 Problem Description

As we mentioned in the previous section, the collected raw data by the IoT devices need to be processed in the first step to become available as meaningful information. The general problem in IoT-based scenarios is that the utilized sensors do not have enough computing and storage capabilities to store and process the collected raw data [15]. Therefore, the raw data must be sent to capable servers to be processed and stored.

As reported in my publication [16]; with regard to the previous works, one of the best computing resources that can be utilized for the processing of collected raw data by the IoT sensors is Cloud [17]. The integration of Cloud computing and IoT technologies provides the feasibility for transmission of the generated raw data by the IoT devices to the cloud resources for further processing and continual storage [18]. However, there are some IoT-based applications that have stringent requirements (such as real-time data communication), which can not tolerate the high response time of Cloud [19, 20]. With regard to [21], one of the most notable challenges of the Cloud is its high latency to the users. As the data providers need to transfer the raw data over the Internet to send it to the cloud, the distance between the users and Cloud increases the communication delay because the data packets must be passed through several routers and firewalls to be received by the cloud servers. This high latency badly affects the performance of delay-sensitive applications, like pervasive smart healthcare [19]. This high latency has been quantitatively discussed in [22], where Cloud was used for processing the healthcare-related data, and provided a response time that was 50% higher than a competitor approach. In addition, authors in [23] compared the response time of cloud with edge computing, where cloud provided a response time that was three times more than the response time of edge servers. It must also be mentioned that utilization of Cloud increases the IBU as the massive amount of generated data by the IoT devices need to be communicated over the Internet [24], which consequently leads to network congestion, increased packet loss rate, jitter, and delay.

In order to deal with the mentioned challenges, Cisco presented the Fog Computing technology in 2012 [20], which extends the cloud services to the edge of the network (closer to the users) [16]. In fog computing, the produced raw data by the IoT sensors are processed by those electronic devices that have computing and storage capabilities (which are located at the edge of the network) [25, 16]. These devices are considered as Fog servers. The extension of cloud computing services to the edge of the network and distributing the computing tasks between the fog servers has several advantages. For example, the latency reduces (up to 33%) [26], as fog servers are located in the vicinity of users, and therefore data does not need to be transferred through the Internet [16]. Using fog servers also reduces the IBU significantly (up to 90%) [24, 27, 28, 29], and improves the security of network as the data packets do not need to be processed by different routers and firewalls before being received by the cloud servers. However, it must be mentioned that utilization of only fog servers in the network is not a suitable approach, as they have considerably less computing power in comparison to cloud servers, and therefore they might provide higher response times in case of higher workloads [28, 16].

As we discussed, both fog and cloud-based approaches have disadvantages, which might make them unsuitable for specific types of applications. Therefore the best approach is efficient utilization of both Fog and Cloud resources and distributing the tasks of IoT devices between the fog and cloud servers. In a combined Fog-Cloud network, the data providers send the generated tasks to a broker, which is responsible for assigning the processing tasks to the fog or Cloud servers. As is presented in figure 1.1, in a combined fog-cloud network scenario, there are heterogeneous fog and cloud servers that might have completely different computational powers, workloads, and latencies. The response time of servers can be affected by all of these factors. For instance, cloud servers have more computational capabilities than fog servers, which means they can provide a faster processing time for processing the compute-intensive tasks (in comparison with fog servers). However, it is worth mentioning that the remote servers in the cloud environments are accessible via the Internet as they are located in specific locations in the data centers that are far from the users; therefore, they have higher delay in comparison to fog servers [16]. Moreover, the workload of fog and cloud servers is another important factor that affects the response time directly. In a fog-cloud network, each of the servers might have different workloads, which causes the response times of the servers to be different. Considering these diverse servers with different workloads, latencies, and computational powers in both cloud and fog environments, when a data provider (for instance, an IoT device) forwards a processing task to the distributor unit (broker) to be transmitted to one of the fog or cloud servers for processing, the broker can not distinguish the fastest server at that time section [16]. As a result, the broker might forward the received task (from the user) to a server that has a higher workload or delay, which causes the IBU and response time to be increased (which is not tolerable by delay-sensitive applications).

In addition, if the number of available tasks in the broker increases (the tasks that must be sent to the servers for processing), the broker can not distribute the tasks between the servers such that the applications experience the least possible response time or IBU [16, 30]. Table 1.1 summarizes the problems for processing the generated data by the IoT devices in different scenarios.

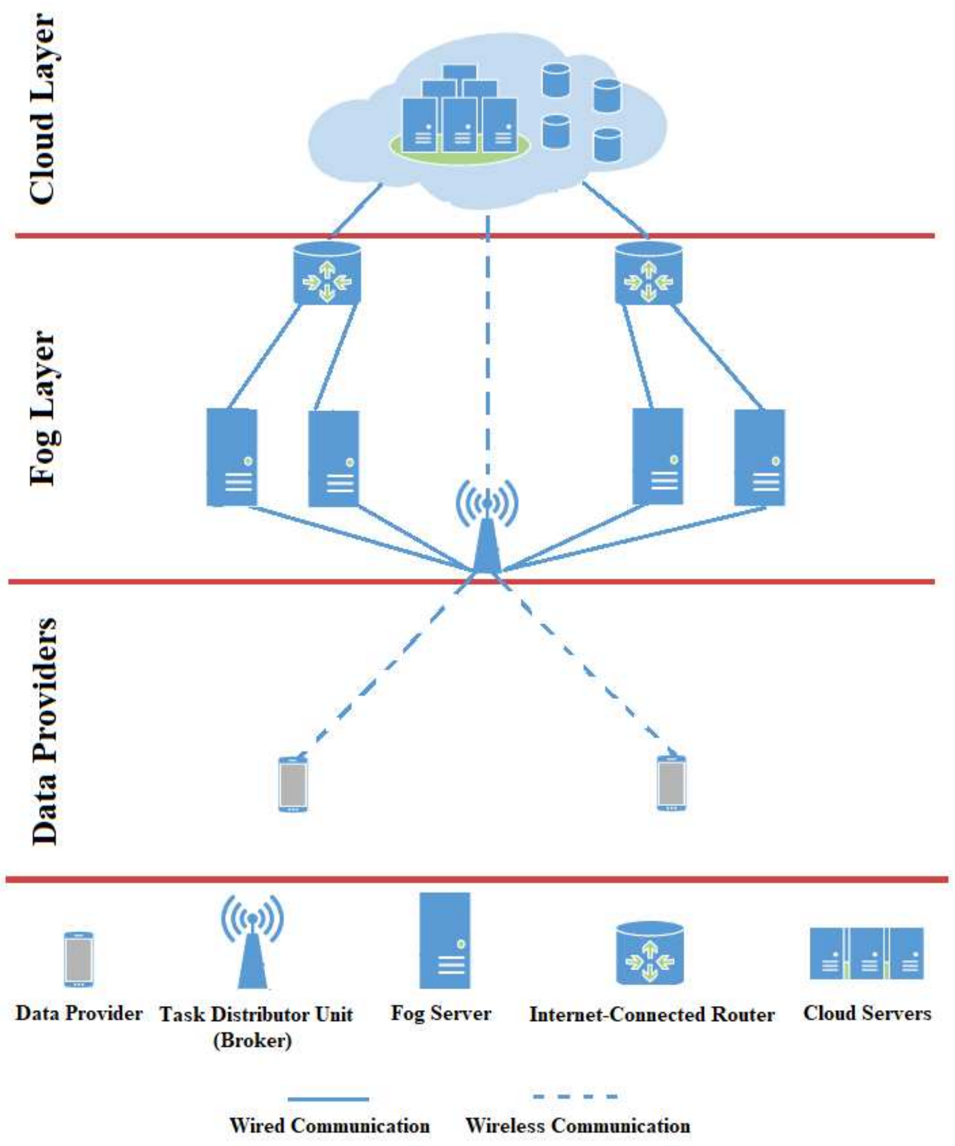


Figure 1.1: Architecture of Combined Fog-Cloud Networks

Table 1.1: The Research Problems

Computing Resource	**Type of Task Assignment to the Servers**	**Problem**
Cloud	One by one, or in batches	Increased Delay, IBU and Security Risks
Fog	One by one, or in batches	Increased Delay in Case of High Workloads of Servers
Combined Fog-Cloud	One by one	Inability of Broker to find the Fastest Server
	in Batches	Inability of Broker to Distribute the Tasks Efficiently

In order to deal with the discussed problems, in this dissertation, we aim to find the answer of the following research question:

- How is it possible to make the broker intelligent, to be able to predict the response times of servers at the time of task arrival, in order to forward the received processing task(s) to the fastest server?
- How to develop the broker to distribute the tasks with regard to the application requirements?
- How to develop the broker to distribute the tasks with regard to the workloads and computing capabilities of servers?
- How does the reduction in response time affect the IBU?

1.3 Hypothesis

There are classic [31, 32] and new [33, 11, 34] methods that are proposed for processing the generated tasks of the IoT devices. The shortcoming of classic methods is that they only utilize one of the fog or cloud resources, and as we discussed in the previous section, these approaches fail in different situations. On the other hand, recent methods are utilized for task distribution between the fog and cloud servers by following specific policies. The problem is that these contemporary methods do not consider the variation of different major parameters such as workload of server, processing time, and latency when there are completely diverse servers in a combined fog-cloud network [1]. It must be mentioned that ignoring these parameters leads to task assignment to unsuitable servers, which causes the response time and

IBU to be increased.

Considering the discussed issues, we propose using the Function Approximation Methods (FAM) to create an intelligent broker (which considers the above-mentioned variables and is aware of the most suitable server at each time slice) to solve the mentioned problems.*"Function approximation is a technique for estimating an unknown underlying function using historical or available observations from the domain"* [35]. Therefore, by training the FAM, prediction of the response time and generated traffic can be possible. So, our hypothesis is that:

- If we use FAM in the broker to make it able to predict the response times of servers at the time of task arrival, the delay-sensitive tasks can be assigned to the fastest servers, which reduces the response time.
- If the broker uses FAM for predicting the size of results, it can make a trade-off between the response time and IBU and reduce both of them efficiently.
- If the broker becomes able to consider the different workloads of servers before task distribution, it can find the fastest server even in dynamic network environments, in which the workloads of fog and cloud servers change over time.
- The performance of the intelligent broker entirely depends on the error of FAM for predicting the response times of servers.
- Utilization of intelligent broker is only useful for specific types of applications, in which there is a relation between the input and output.
- The training method of FAM, and the richness of training, play important roles in the precise prediction of response times (which can help us for validation of results).

1.4 Contributions

In this thesis, we propose a solution for improving the QoS for the IoT-based delay-sensitive applications. Our proposed method distributes the tasks of the IoT devices between the fog and cloud servers by considering the user requirements, and reduces the response time and IBU up to 53% and 50% in comparison to the cloud based approach (also up to 3% and 57% in comparison to a competitor method), respectively. It is worth mentioning that the provided comparison is based on only one of the experiments that we performed (more quantitative data are provided in chapter 5). This method uses an AI-based solution to make the broker intelligent to be aware of suitable servers at the time of task arrival. Moreover, we provided a scalable solution, which can be used in any network architecture. To be more precise, with regard to the mentioned problems in previous sections, we provided the following specific contributions:

- **Creation of an Intelligent Broker**

In a combined fog-cloud scenario, when delay-sensitive applications send a task to the broker, the broker cannot find the fastest server at that time slice. The reason is that in such networks, there are many different servers in both fog and cloud layers, which are heterogeneous in terms of computational power, delay, and workload. However, the broker can use Artificial Intelligence and specifically function approximation methods to predict the response times of servers, and select the fastest server at each time slice. Therefore, by considering the previous works, in which artificial intelligence helped for improving the response time in fog-cloud scenarios (see chapter 3), we propose an AI-based solution for making an intelligent broker that can predict the response times of available servers in the network at the time of task arrival.

To this end, we generated a number of tasks and ran them on the fog and cloud servers. Then for each server, we trained a function approximation method, in which the task parameters were the inputs, and the response time of the server was set as the output. Therefore, when a delay-sensitive task arrives, the broker sets the task parameters as the input of the neural networks and then finds the fastest server at the time of task arrival and assigns the task to that server for processing. It is worth mentioning that this intelligent broker can distribute any number of received tasks between the servers.

- **Developing the Intelligent Broker to Distribute the Tasks with Regard to the Application Requirements**

As discussed earlier, in terms of delay sensitivity, IoT-based applications can be delay-sensitive or delay-tolerable. The delay-sensitive applications require quick response time as latency can badly impact their performance. On the other hand, there are delay-tolerable applications that aim to communicate the least possible amount of data over the Internet to reduce the IBU and costs. We developed the broker and modified our proposed task distribution algorithm to check the delay sensitivity of the tasks. Besides response time, we also added the "size of result" as the output of the neural networks. In the next step, with regard to the application requirement, our proposed method distributes the tasks among the fog and cloud servers by considering the predicted response times and sizes, aiming to reduce the response time for delay-sensitive, and IBU for delay-tolerable applications [16].

- **Developing the Intelligent Broker to Distribute the Tasks by Considering Workloads and Computing Capabilities of the Servers**

As discussed earlier, in a combined fog-cloud network, there are different servers in both fog and cloud layers, which are heterogeneous in terms of hardware specification, workload, and latency to the user, and all of these factors impact the response time directly [16]. Therefore, when a task arrives from a data provider, as the task distributor unit is not aware of this heterogeneity, it might forward a task to a server with a high workload (or to a weak server in terms of computation power), which provides a high response time (that is not tolerable by delay-sensitive applications).

In order to deal with this problem, we did the training process of the neural networks by considering the different workloads that servers might experience. Therefore we develop the broker to predict the response times by considering the different workloads and computational power of servers.

- **Investigating the effective Parameters on the Performance of our Proposed Approach**

To validate our proposed task distribution approach, we investigated its performance under different conditions to observe where it works better than the other proposed methods in the literature and where it fails. For this purpose, we used different training algorithms and different numbers of tasks for the training process of the neural networks. We also considered an ideal situation where our proposed method was able to predict the response time without any error and compared the achieved results with each other.

1.5 Evaluation of Achievements

In this thesis we propose an intelligent method that utilizes Artificial Neural Networks (ANNs) for task distribution in combined fog-cloud scenarios, and compare the performance of our proposed method with five other classic and new methods in different experimental conditions. The achieved results show that our intelligent method performs better than the other techniques and can reduce the response time, IBU, and resource utilization [16]. Just as an example, in a network architecture, where there are different fog and cloud servers with different workloads and latencies, our proposed method reduced the response time and IBU up to 3% and 57% in comparison to a competitor method, respectively. The reason is that our method considers different variables, such as the computing capabilities of both fog and cloud servers, in addition to their workloads and latencies (more comparisons, results, figures and details can be found in chapter 5).

It must be noticed that the performance of our proposed method entirely depends on the error of predictions that can be improved or deteriorated by utilization of different training methods or altering the richness of the training process of the ANNs [1]. It is also worth mentioning that our proposed method efficiently predicts the response times of those tasks that variation of their parameters affects the response time. The reason is that in these situations, the ANNs can find a relation between the input (task parameters) and output (response time or data size), which consequently leads to better predictions.

1.6 Thesis Structure

In the following, we present an overview of the contents of this dissertation. Each paragraph provides a summary for each chapter. The main contributions of this thesis are presented in chapters 4 and 5.

- **Chapter 2 - Basic Concepts**

Chapter 2 provides information that needs to be discussed for a better understanding of our proposed intelligent task distribution method. This chapter defines basic concepts such as IoT, Cloud and Fog computing, types of communications in IoT-based scenarios, IoT architecture, applications, and their requirements and challenges. We also compare fog and cloud computing technologies from different aspects and discuss the challenges in this field.

- **Chapter 3 - Previous Works**

The previous methods that have been proposed for task distribution in fog-cloud scenarios are discussed in chapter 3. In this chapter, we categorize the previous researches and provide comparisons between the proposed methods. Also, we discuss about the advantages and disadvantages of different task distribution techniques.

- **Chapter 4 - Case Study and Primary Experiments**

In chapter 4, we describe the case study that we have used in our experiments. We also present the primary experiments that we performed to compare the performance of Fog and Cloud computing in terms of response time and IBU.

- **Chapter 5 - Proposed Methods and Experimental Results**

In this chapter, we propose solutions for the research problems that we discussed earlier in section 1.2. In addition, we discuss about the experiments that have been performed for evaluating the performance of our proposed intelligent approach, and finally, we provide a brief discussion about the advantages and disadvantages of our method.

- **Chapter 6 - Conclusion and Future Works**

Chapter 6, concludes the thesis and provides some research questions which must be considered for future works.

2 Basic Concepts

2.1 Internet of Things

Today, the Internet is almost accessible everywhere, and data communication over the Internet has significantly changed the quality of life. However, the advancement of technology is not over yet. We will soon experience living in the IoT era, where a wide variety of things will be connected to the Internet, generating and communicating data with each other. There is no universally accepted definition for the IoT, as it is based on the integration of various types of technologies [36]. Therefore, each enterprise or community has defined the IoT based on their own visions and needs. In the following, we review some of the most outstanding definitions.

Kranenburg [37] defines the IoT as *"a dynamic global network infrastructure with self-configuring capabilities based on standard and interoperable communication protocols where physical and virtual' Things' have identities, physical attributes, and virtual personalities and use intelligent interfaces, and are seamlessly integrated into the information network"*.

ITU [38] describes IoT as *"A global infrastructure for the information society enabling advanced services by interconnecting (physical and virtual) things based on, existing and evolving, interoperable information and communication technologies"*.

Vermesan et al. [39] also defined the IoT as a paradigm in which *"physical entities have digital counterparts and virtual representation; things become context-aware, and they can sense, communicate, interact, exchange data, information, and knowledge"*.

Regarding the discussed definitions, we consider the IoT as a computing paradigm in which objects with computing and connectivity capabilities collaborate to solve complex tasks to improve the living standards of people.

2.2 IoT Architecture

Different architectures are proposed in the literature for IoT-based systems. As is shown in figure 2.1, the most common architecture discussed by [40, 41, 42] consists of three layers, namely perception, network, and application.

The physical layer, or the so-called perception layer, is responsible for data collection from an environment by using sensors or other smart devices (which will be called data providers in this thesis). *"Sensor is a device, formed by sensitive cells, that transforms physical or chemical magnitudes in useful signals to measure or control systems"* [43]. The utilization of sensors in the perception layer causes the IoT to be known as a context-aware technology [2]. In order to collect data from an environment, IoT-based applications need one or several sensors, which are mostly small in size, low cost, and limited in terms of battery capacity and ease of deployment. IoT-based applications utilize different types of sensors with regard to their requirements. For example, healthcare applications use medical sensors to monitor different parameters such as respiration rate, heart rate, body temperature, etc. As another example, smart city based applications utilize environmental and chemical sensors (which are used to sense different parameters such as humidity, air pollution, and temperature) for monitoring purposes.

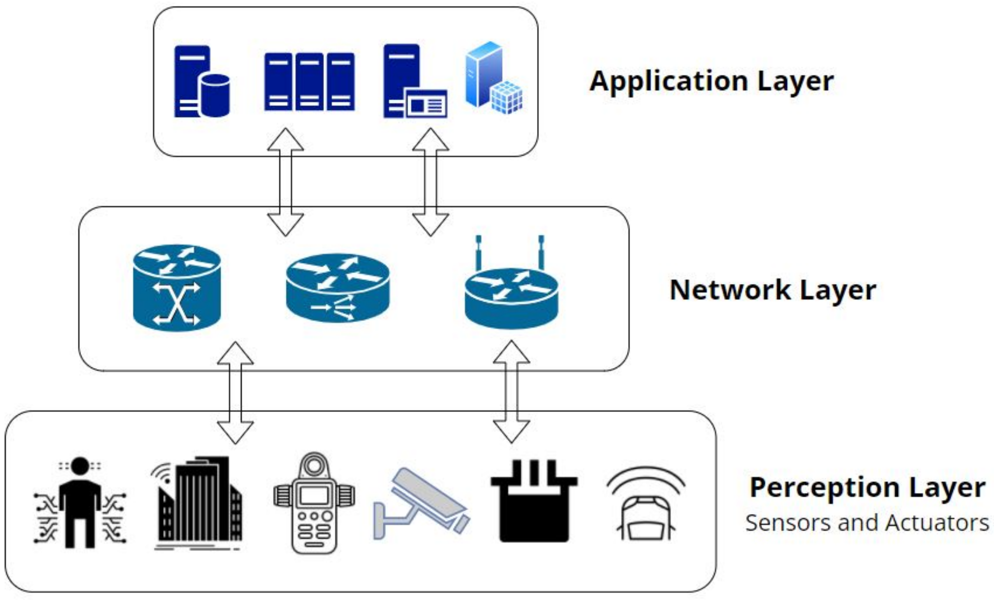


Figure 2.1: The IoT Architecture

The sensors perform the data acquisition process and provide the raw data, which must be processed to be available as meaningful information [1]. In this thesis, the provided raw data by the sensors at each time slice, which must be sent to a server for processing or storage, is considered as a task.

Moreover, the network (transport) layer is responsible for connecting the smart things to the servers by communicating their generated raw data to the application layer through different types of networks such as 4G, LAN, RFID, Internet, etc. The application layer is responsible for receiving the produced raw data via the network layer and assigning them to capable servers [16]. The application layer also provides various services for the users by employing different technologies such as cloud computing, databases, and big data processing modules [2].

2.3 IoT Applications

IoT provides various applications for different purposes, such as quality control, monitoring, and digitalization. This section reviews some of the most important IoT applications, such as Intelligent Transportation System (ITS), smart homes, and Smart Health.

- **Smart Home**

Smart homes are considered as an important application of IoT, in which sensors are utilized for different purposes such as automation of daily tasks, energy conservation, and security issues. In smart homes, the collected data by the sensors must be sent to an aggregator, which transmits the data to a context-aware service engine, and then this engine provides services based on the context [2]. For example, when the sensor data shows that the air humidity has increased, the engine turns the air conditioners on, or when a gas leak is sensed, the application turns the lights off.

- **Intelligent Transportation System**

ITS is another application of IoT that is emerged by the combination of IoT and transportation system [44], to evolve the traffic control and prediction systems. ITS collects data by exploiting various sensors such as GPS (to detect the location of vehicles), accelerometers (to measure the speed of cars), gyroscopes (to find the direction of vehicles), RFID (to identify the cars), and cameras (to record and monitor the movements of vehicles) [2], and then provides services for the users by considering the collected information. The main aim of ITS is to increase the efficiency of the transportation system (by minimizing traffic congestion), ensure safety and security (by avoiding accidents), and reduce fuel consumption.

- **Smart Health**

Thanks to the emergence of IoT and its combination with some other technologies such as fog and cloud computing, medical doctors can now provide faster and more efficient healthcare services by using healthcare applications [45]. Healthcare applications are categorized as the most important and attractive applications of IoT [46], that aim to reduce healthcare costs and improve the patient's quality of life.

In IoT-based healthcare applications, different types of medical devices such as sensors and diagnostic tools are utilized to provide healthcare services. One of the recent technologies that is being used in IoT-based healthcare applications is the Wireless Body Sensor Networks (WBSNs). As is shown in figure 2.2, WBSNs are composed of different wearable or in-body sensors, which continuously monitor the patient's health condition and transmit warnings to healthcare specialists in case of any abnormal situations. As the sensors do not have data processing capability, they transmit the sensed data to a cluster head or broker, and then the broker sends the data to capable servers for processing and storage. After the processing, the servers send the response to the user and store the results in databases for future data mining purposes. As any network failure or high latency impacts the patient's quality of life, these applications are considered to be real-time or delay-sensitive. Therefore provisioning of QoS for these types of applications is of great importance.

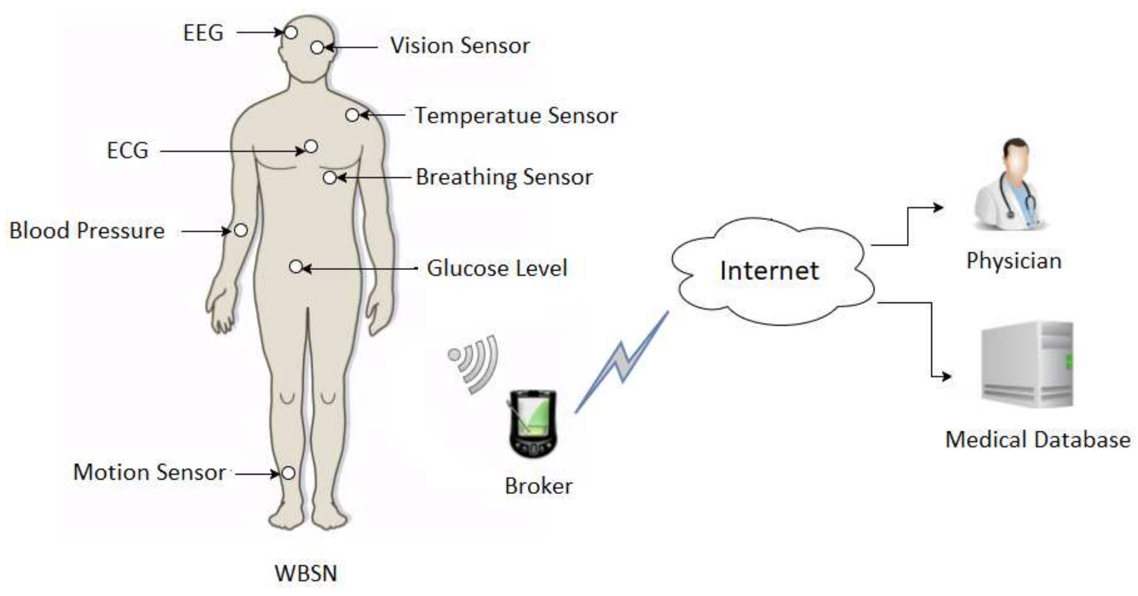


Figure 2.2: Wireless Body Sensor Networks

2.4 IoT Challenges

In this section, the main challenges of IoT systems are discussed as follows:

2.4.1 Limitation of IoT Objects

The IoT devices generate (or collect) data continuously. As the storage, communication, and computation capabilities of IoT sensors are limited for the huge amount of generated data (that is estimated to be 850 zettabytes by 2021 - overall by all devices) [47]; therefore, the compute-intensive tasks cannot be executed on them [48]. For this purpose, efficient resource provisioning algorithms must be designed to assign the generated tasks of the IoT devices to capable servers for processing and storage.

2.4.2 Communication

Communication between the IoT objects and servers is also another important challenge, which must be solved. The communication between the IoT devices is mainly wireless as the sensors are usually located in geographically dispersed areas. The challenge is that wireless channels are unreliable because of their high distortion [2], and the sensors need dedicated spectrums to communicate data. Concerning the limitation of spectrum availability, a dynamic spectrum allocation is required to provide the feasibility of data communication for billions of sensors [49].

2.4.3 Mobility Management

The mobility of IoT devices causes different challenges in the IoT networks, such as increasing the mobility signaling costs, latency, packet loss, and power consumption [50]. It must be noticed that, because of the limitations of IoT sensors in terms of processing capability, the current mobility protocols of mobile ad-hoc networks (MANETs) and sensor networks can not fulfill the requirements of an IoT based network with mobile objects. Therefore, mobility management is one of the most critical issues in the IoT that must be considered in future researches [48].

2.4.4 Security Issues

The potential security threats have been escalated in the IoT networks with the increase of IoT-connected devices. *"IoT devices are vulnerable due to lack of transport encryption, insecure Web interfaces, inadequate software protection, and insufficient authorization"*[51], and consequently they provide potential attack surfaces for the hackers. Therefore, to deal with security challenges, updated security measures must be ensured for the IoT networks to prevent data monitoring or interference [49].

2.4.5 Privacy

IoT devices provide and communicate data about users' location and movements, health conditions, etc., which increases privacy concerns. Therefore, in order to make IoT acceptable for the users, the protection of users' privacy must be considered for the development of IoT [49, 51].

2.4.6 Naming and Identity Management

In IoT, various devices are utilized to collect and communicate data over different types of networks. Each of these IoT devices needs a unique identity to receive and send data. Thus, IoT needs efficient dynamic naming and identity-management protocols in order to manage the vast amount of smart objects that communicate data in the IoT infrastructure [49].

2.4.7 Interoperability and Standardization

As we discussed, in IoT, billions of heterogeneous devices communicate data with each other by utilizing various technologies. The challenge is that these devices can not communicate with each other in some cases as they use different services, software, and hardware. Therefore, standardization of IoT is a crucial task that must be done for better interoperability of IoT devices [49].

2.4.8 Data Management

The IoT devices produce huge amounts of data that must be communicated over the network to be processed and stored. The challenge is that the current data centers cannot deal with the massive generated data. Therefore, enterprises prioritize data and make backups based on the needs and value, which affects the future data mining processes [51].

2.4.9 Data Mining

As IoT devices generate massive amounts of data, the utilization of data mining methods for discovering patterns and relationships between the data becomes more important. The data mining process helps enterprises to make better business decisions for future purposes. The IoT data consists of discrete and streaming data generated by different devices such as sensors, vehicles, industrial equipment, etc. Traditional data mining methods can not deal with the massive amount of heterogeneous data that gets generated by the IoT devices. Therefore, robust approaches are needed to be proposed and implemented to improve data mining in IoT [51].

2.4.10 Internet Traffic

According to the Cisco report, [52], by the end of 2030, almost 50 billion things will be connected to the Internet, and in the near future, the vast amounts of generated data by the IoT sensors will occupy the Internet Bandwidth remarkably [19, 24, 28]. As is mentioned in [53] till the end of 2020, 1.6 Zettabytes have been generated by the IoT devices, and transferring all these raw data from the sensors to the remote servers in the cloud (for processing) increases the Internet bandwidth utilization, which leads to increased latency, packet loss, and jitter [1].

2.5 The Requirements of IoT Applications

IoT applications have different requirements. For example, remote health monitoring applications require a stable Internet connection and reduced packet loss rate. On the other hand, some other applications that use costly technologies such as satellite or cellular communications, need to communicate the least possible amounts of data over the network to reduce the costs.

In terms of latency, IoT applications have different sensitivity levels, and therefore they are categorized into delay-sensitive and delay-tolerant groups [54]. Smart health and autonomous vehicles are two important examples of delay-sensitive applications as their performance can be affected by even the least possible delay. On the other hand, there are delay-tolerant applications such as smart parking, waste management system, and energy conservation applications, in which data can be communicated with large latency (higher than 100ms). These applications can be supported by the current protocols and existing network infrastructures, as the provisioning of reduced response time for these applications is not crucial. But obviously, the existing network protocols can not meet the requirements of delay-sensitive applications, and therefore, new networking algorithms must be designed [55].

2.6 Cloud Computing

Ian Foster defines Cloud computing as *"a large-scale distributed computing paradigm, in which a pool of abstracted, virtualized, dynamically scalable, managed computing power, storage, platforms, and services are delivered on-demand to external customers over the Internet"* [56]. The National Institute of Standards and Technology (NIST) also defines Cloud as *"a model for enabling ubiquitous, convenient, on demand network access to a shared pool of configurable computing resources that can be rapidly provisioned and released with minimal management effort or service provider interaction"* [57]. One of the latest definitions of Cloud has been presented by Elazhary in [58], who defines Cloud as *"a computing paradigm for providing anything as a service such that the services are virtualized, pooled, shared, and can be provisioned and released rapidly with minimal management effort"*.

2.6.1 Cloud Computing Advantages

Cloud computing provides flexible computational tools such as servers, databases, storage, software, data mining, machine learning, and visualization tools through the internet [59, 2]. Therefore, Cloud can be an excellent option for processing the generated data by the IoT devices, as combination of Cloud and IoT allows the limited data providers to use the processing and storage capabilities of Cloud, and assign the complex tasks to the cloud servers for processing [60]. From a technical point of view, Cloud provides the following advantages:

- **Powerful Computing Capabilities**: Cloud can use the power of thousands of computers to solve a problem, which leads to faster processing time and, consequently, reduced response time.
- **Great Storage Capacity**: the massive generated data by the IoT devices can be stored in the Cloud as it has huge amounts of storage capacity.
- **Suitable Accessibility**: the cloud resources are accessible from anywhere at any time, through the Internet.
- **Supporting the Mobile Users**: as cloud servers are accessible over the Internet, any mobile user (with Internet access) can also benefit from the advantages of Cloud.
- **Reduced Maintenance Costs**: as we discussed, Cloud provides a shared pool of computing resources, and therefore the organizations do not need to pay for hardware and software.

2.6.2 Cloud Computing Challenges

Cloud computing provides excellent services that can be used by the IoT devices, but it is worth mentioning that the cloud resources are centralized in Data Centers (DCs) that are far from the IoT devices in the perception layer [60]. The mentioned topological distance causes several challenges, such as increased response times that make the cloud unsuitable for delay-sensitive applications such as smart health [1]. In this subsection, we discuss the most important disadvantages of the cloud (mentioned in [60]), which cause different challenges for IoT-based applications.

- **Bandwidth Utilization**: In Cloud-based IoT systems, most of the computational processes occur in Cloud, which means that all the produced data and requests must be transmitted to the cloud for processing and storage [16]. It must be mentioned that unlike the processing power of servers, which has been risen, the Internet bandwidth has not increased considerably [61]. As is discussed in [53], by the end of 2020, more than 1.6 zettabytes are generated by the IoT devices [1]. The current Internet bandwidth can not deal with this huge amount of data, and therefore it is almost impossible to transfer the generated IoT data to the cloud [61]. This challenge leads to high latency and badly affects the performance of those delay-sensitive and real-time applications that need a quick response time.

- **High Latency**: There are different IoT applications that require predictable and quick response times. For example, road safety services need a delay of less than 50 ms, and Smart Factories can tolerate latencies varying from 0.25 to 10 ms. The topological distance between the IoT devices and Cloud servers leads to high communication delay, which is unsuitable for delay-sensitive applications [60].
- **Privacy and Security Issues**: As in IoT-based scenarios, data can be collected from anywhere; the user's privacy can be jeopardized [62]. In many cases, the IoT devices sense and transmit sensitive data (such as personal health data) that need protection. As the users have limitations to control the data path from an IoT device to the Cloud, and as the IoT devices do not have adequate computational capabilities to encrypt the data, communication of these sensitive data through the Internet (in order to be received by the cloud servers for processing) increases privacy concerns [63].
- **Hostile Environments**:

  Hostile environments are those places such as rural areas, in which Internet infrastructure is weak and stable access to the Internet is not guaranteed. Therefore, utilization of cloud services for delay-sensitive applications is not possible in hostile environments, as any internet disconnection or increased response time leads to the failure of these applications [64, 60].

2.7 Fog Computing and its Characteristics

Fog computing has emerged to complement the Cloud and tackle its challenges. Similar to Cloud, there are different definitions for Fog computing. From the theoretical point of view, authors in [20] defined Fog computing as a *"highly virtualized platform that provides compute, storage, and networking services between IoT devices and traditional cloud computing data centers, typically, but not exclusively located at the edge of the network"*.

Also, in [65] Fog computing is considered as *"a scenario where a large number of heterogeneous (wireless and sometimes autonomous) ubiquitous and decentralized devices communicate and cooperate to perform storage and processing tasks without the intervention of third parties. These tasks can be for supporting basic network functions or new services and applications that run in a sandboxed environment"*.

Moreover, from the technical point of view, IBM [66] defines Fog as a technology which provides *"resources at the edge of the cloud, instead of establishing channels for cloud storage and utilization"*, while Open Fog Consortium (OFC) describes Fog *"as a horizontal, system-level architecture that distributes computing, storage, control and networking functions closer to the users along a cloud-to-thing continuum"* [67].

Fog computing extends the cloud services to the edge of the network and provides services such as data processing, temporary (or permanent) data storage, and data communication. The most important characteristic of Fog computing is its proximity to the data providers (or users), which makes it completely different compared to Cloud. The proximity of fog servers to the users reduces the data transmission time and Internet traffic [68]. Therefore, fog computing can meet the requirements of delay-sensitive applications. Generally, Fog has the following important characteristics and advantages.

- **Low Latency**: Fog is specifically designed for those delay-sensitive applications that need rapid response time [69, 61, 70]. Despite Cloud, which communicates data over the Internet, in fog computing, the generated raw data by the IoT devices are sent to the fog servers (that are located at the edge of the network) to be processed, or stored [16]. This data processing at the edge of the network eliminates data transmission over the Internet, reduces the latency, and makes fog computing suitable for real-time applications [20]. There are several researches in which the performance of Fog and Cloud are compared in terms of latency. For instance, in [71] authors utilized fog computing for face identification purposes, and in [27] fog computing was utilized for processing the healthcare-related data. Both researches indicate that the response time of Fog is significantly less than the cloud.
- **Saving Internet Bandwidth**: In fog computing, servers are located at the edge of the network, between the data providers and remote servers in the cloud. These fog servers can locally provide different services such as preprocessing, redundancy removing, data aggregation, data compression, and cleaning [61]. Therefore, most of the generated data by the IoT devices do not need to be communicated over the Internet to be received by the cloud servers. In addition, in some other scenarios, the whole processing procedure is

done by the fog servers, and no data needs to be transmitted to the cloud [16, 30]. In these situations, fog computing effectively reduces the IBU, as all the processing tasks are performed at the edge of the network. Different researches such as [71, 27] compare fog and cloud, and the results show the positive impact of fog computing on the reduction of IBU. It is worth mentioning that this advantage of Fog becomes more important as the number of IoT devices increases [61].

- **Mobility Support**:

  There are fixed and mobile data providers in fog computing-based scenarios, such as vehicles, smartphones, and traffic cameras. Similarly, fog servers can also be fixed, or mobile computing resources, which can be deployed at home and airport, or on the mobile vehicles and trains [72, 73, 74]. Fog computing utilizes routing, addressing, and communication protocols to provide the feasibility of interaction between the mobile devices and fog servers [61]. It must be mentioned that in fog computing, the administrators can control the mobile users to see how they join the network and access the information [75].

- **Decentralized Architecture**:

  Cloud servers are mostly centralized in data centers and provide services for the user through the Internet. Compared to Cloud, which is more centralized, Fog consists of a huge number of distributed and heterogeneous nodes that provide computational services in the proximity of the users. This decentralized architecture of fog at the edge of the network provides better location-based services, quick analysis of data, and real-time decision making [61].

- **Heterogeneity**: Generally, any device with computational and storage power can play the role of a fog server. For example, a powerful computing device, a router, or an access point can be used as a fog server. Obviously, these different devices have different form factors, computation and storage capabilities, operating systems, and workloads. It must also be mentioned that different wired and wireless communication technologies can be utilized in fog-based network scenarios. Therefore, fog computing environments are heterogeneous in different terms.

- **Interoperability**: We discussed that fog servers are provided by different companies, and the fog computing environment is heterogeneous. It must be noticed that in fog computing based scenarios, a wide range of heterogeneous devices need to collaborate with each other to meet the requirements of IoT-based applications [76]. In order to provide the possibility of interaction between the heterogeneous IoT devices and fog servers, a policy-based resource management method is proposed in [77, 78] that supports the requirements of a fog-based network and provides a stable collaboration between the devices in the heterogeneous, dynamic, and distributed environment of fog.

- **Energy Saving**: In Fog-based networks, servers are distributed, and therefore, no heat will be generated because of the concentration of servers, and consequently, no cooling system is required [61]. Also, the short-range data communication in Fog reduces the energy consumption for communications [79]. The impact of fog computing on energy consumption has been deeply investigated in [80, 81], and the achieved results show that compared to Cloud, utilization of fog computing reduces the energy consumption significantly.

- **Data Protection**: As in Fog computing, data does not need to be communicated through the Internet; access to data is more restricted compared to Cloud. Besides that, fog servers also support the data by encryption and therefore increase the security of data.

- **Failure in High Workloads**: Generally, fog servers have less computing power than cloud servers. Therefore when the workloads of fog servers increase, they provide higher response times because of their limitations. Authors in [28, 30] investigated the performance of fog and cloud in different workloads, and the achieved results show that when the workloads of servers increases, fog servers fail and Cloud provides a better response time.

- **Volatility of Service Availability and Reliability issues**: As the fog servers are highly mobile, they are intermittently present in the network architecture. Therefore, the created infrastructure in fog-based network scenarios is temporary, which leads to the volatility of service availability and reliability issues [82].

- **Security Issues**: As discussed earlier, in fog computing, there is no centralized architecture, and therefore, there is no strict control from a service provider to guarantee the security of data communication. Hence, fog computing can be more vulnerable than Cloud in terms of data security [82].

2.8 Fog vs. Cloud

In this section, we compare fog and cloud computing technologies in different terms. For this purpose, we summarized the previously discussed characteristics of Fog and Cloud in Table 2.1 to show the differences between these two technologies.

Table 2.1: Fog vs. Cloud

	Fog Computing	**Cloud Computing**
Operational Cost	Low	High
Energy Concumption	Low	High
Location of Servers	Near to the user	Far from the users (accessible through the Interent)
Latency	Low	High
Geographical Distribution	Less Centralized	More Centralized
Possibility of Real-Time Interaction	Possible	Impossible
Failure Rate	High	Low
Processing Time of Servers	High	Low
Storage Capacity	Low	High
Server Mobility	High	Low
Cooling Cost	Low	High
Internet Bandwidth Utilization	Low	High
Computational Power	Low	High
Accessibility	Limited (geographically)	Global

2.9 Combined Fog-Cloud Scenarios

In the previous sections, we presented the characteristics of Fog and Cloud computing technologies and discussed about their advantages and disadvantages. Fog emerged to tackle the challenges of Cloud, but it is also limited from different aspects. Therefore, it can be generally said that Fog and Cloud are complementary technologies, and none of them can singly be a good solution for processing the generated data of the IoT devices. Therefore the combined fog-cloud computing emerged and got discussed in [83, 84, 85, 86, 87].

The architecture of fog-cloud networks consists of heterogeneous devices with completely different form factors; therefore, a set of standards is required to ensure the interoperability between the devices [88]. In combined fog-cloud scenarios, a task can be processed either by Fog or Cloud servers, and as these servers are heterogeneous in terms of hardware and workload, new algorithms must be designed to distribute the tasks between the fog and cloud servers.

3 Previous Works

In order to deal with the limitation of IoT sensors for processing and storage of their generated data, different solutions have been proposed in the literature, which will be reviewed in this section.

3.1 Benchmarks

3.1.1 Utilization of Cloud Computing

The first efforts for processing the generated raw data by the IoT sensors were based on the utilization of remote servers in the cloud. For instance, authors in [31] proposed utilization of cloud computing for processing the IoT data in agricultural applications. This approach had advantages and disadvantages that got discussed in section 2.6. It is clear that the remote servers in the cloud can provide powerful computing resources (in terms of processing and storage capabilities) for the IoT applications, which lets them store huge amounts of data and experience reduced processing times. However, it must be considered that the cloud is based on a Pay-as-you-go model, so it is costly. Also, if the IoT applications utilize the cloud servers to process their generated raw data, the Internet bandwidth would be occupied by the massive generated IoT data, leading to increased response times and lack of enough bandwidth for the other applications. Furthermore, as the remote servers are only accessible through the Internet, the latency of data communication over the Internet leads to increased response time because of the high latency of the cloud servers [1].

3.1.2 Utilization of Fog Computing

In order to overcome the high latency of data communication with the cloud servers, in different papers such as [89, 27], authors proposed the utilization of fog computing for delay-sensitive applications. However, as we discussed in section 2.7, the limitation of fog servers in terms of computational power leads to high response times in high workloads. It has been illustrated in [30], where the fog server provides a response time that is 50% higher than the cloud for processing 10 batches of healthcare data. It is worth mentioning that the more data gets assigned to both cloud and fog servers, the more differences in response times can be seen. Therefore, exploiting the fog servers is not always a proper solution.

3.1.3 Simple Methods for Utilizing both Fog and Cloud Servers

In order to employ both Fog and Cloud computing resources for processing the generated tasks of the IoT devices, authors in [33] proposed the Random Fit method, which selects one of the fog or cloud servers randomly for job allocation. Moreover, in [90], Greedy Assignment (better workload) method is proposed, which assigns the tasks to servers with the least workload to be processed. Furthermore, in [34], another task distribution method is proposed, which can be used for task distribution between the fog and cloud layers. This method selects two random nodes from the available servers and then assigns the generated task to the server with fewer tasks in its queue. In [91] also another task placement method is proposed, which sends the task to the home fog (nearest fog servers which has the least latency to the user) and if the waiting time was greater than a threshold, it assigns the task to the next fog server (a neighbor of the home fog). In addition, in [92] Latency-Aware task placement is proposed, which considers the overall latency of servers at the time of task distribution, and sends the tasks to the server with the least latency.

It must be mentioned that the performance of the discussed methods can be affected by different parameters. For example, the random fit method might assign a delay-sensitive task to a server with high latency, or the better workload method might assign a task to an idle server with high latency [1]. In both situations, the response time increases, and the requirements of delay-sensitive applications can not be met. It is also worth mentioning that the discussed simple methods, in addition to the cloud-based and fog-based methods, have been used as benchmarks in several recent researches.

3.2 Recent Complex Methods

3.2.1 Optimization Based Methods

- **Virtual Fog Resolver**

In [11], finding the most suitable server at each time slice is investigated, and a two-step resource management method called Virtual Fog Resolver (VFR) is proposed for distributing the tasks of IoT applications between the fog and cloud servers. This method provides a quick recovery approach in case of failure of fog servers and aims to improve the QoS by reducing the response time. As mentioned earlier, the

VFR method functions in two steps. In the first step, the fog servers around the user are categorized into two groups; home and backup fogs. For this purpose, at first, the user sends a similar task to all the fog servers and then notes their response times. The fastest fog server is considered as the home fog, and the second-best fog server will be considered as the backup fog.

After selecting the home and backup fogs, the user sends the generated request to both home and backup fogs (it is assumed that the home fog still performs better than the backup fog and provides a faster response) and notes their response times. Then, the delay is considered as the absolute difference of these two response times. The next request will be sent to the home fog; if the response of the home fog does not arrive before the delay, the request will be sent to the backup fog; on the other hand, the delay is reduced. The reason for reducing the delay is to ensure that when the home fog performs excellently, the request is not sent to the backup fogs. Assuming that the home fog responds after the delay, two different situations can happen:

- the home fog responds before the backup fog
- the backup fog responds before the home fog

In both situations, the results will be sent to the user, and the delay is updated.

While the request gets distributed between the fog servers, the probability of task assignment to the cloud increases. If this probability reaches a maximum amount, the requests will be sent to the cloud servers. In this case, if a fog server responds before the cloud server, then the probability of task assignment to the cloud reduces. Therefore, in this approach, the cloud servers only get used when the fog servers perform poorly, and the probability of task assignment to the cloud depends on the performance of the fog servers.

Authors finally compared the performance of VFR with PO-2 [93] and MinDelay [91] algorithms. The results show that VFR reduces the delay and resource utilization compared to the mentioned methods.

- **LBP-ACS: A Laxity and Ant Colony System Algorithm for Task Scheduling in Cloud-Fog Networks**

In order to save energy, provide green computing, and protect the environment, authors in [12] proposed a task scheduling method for combined fog-cloud scenarios, which reduces the energy consumption and consequently the production costs. In this paper, the challenge of task scheduling in IoT-based applications is investigated. To solve the challenge of scheduling complex tasks with priority constraints, a novel task scheduling algorithm is proposed, which utilizes laxity-based and ant colony algorithms, aiming to reduce the total energy consumption. This method considers both the priority and the finishing deadlines of the tasks and handles the delay sensitivity of the IoT tasks by using the laxity-based priority algorithm for making a task scheduling sequence. Also, in this method, the ant colony algorithm is used to provide optimal scheduling. Analyzing the performance of LBP-ACS shows that this method reduces the energy consumption and failure ratio of the system compared to similar methods.

- **CAG: Cost Aware Genetic-Based Method for Task Scheduling in Combined Fog-Cloud Scenarios**

For efficient task placement in combined fog-cloud scenarios, a cost-aware genetic-based algorithm is proposed in [13], which is suitable for real-time (hard deadline) applications as it improves the QoS, reduces the costs, and increases the data security. This method decides where to process a task to reduce the response time in scenarios where fog servers collaborate with rented servers in the cloud for efficient large-scale IoT task processing.

In order to provide a suitable trade-off between cost reduction and meeting the deadline of IoT tasks, this method uses a genetic-based algorithm in a central scheduling node. In the first step, the generated task is sent to the Master Fog Node, which is responsible for running the scheduling algorithm. The scheduler estimates the response time by considering the network capabilities, current computational resources, and task requirements. With regard to the estimated response time, if it was possible to process the task in the fog layer, the scheduler would forward it to one of the fog servers; on the other hand, the task would be sent to the cloud.

The experimental results show that this method reduces the costs (in terms of utilizing Internet bandwidth for communication of requests and responses) and resource utilization, and increases the success rate of task processing before the specified deadline.

- **Utilization of Analytic Hierarchy Process (AHP) for Dynamic Resource Allocation in Fog-Cloud Networks**

The delay sensitivity of some of the IoT tasks, the irregular changes in traffic patterns, and the heterogeneous nature of combined fog-cloud scenarios increase the need for efficient resource allocation policies. The resource allocation must be precise, and also different important parameters must be considered at the time of resource assignment. For this purpose in [14], Analytic Hierarchy Process based policy for resource allocation in fog-cloud scenarios has been proposed.

In order to reduce the delay of task processing, this method considers the network and computing load at the time of task assignment and then by evaluating the overall data, dynamically assigns weight to each criterion of optimization and then selects the most suitable server for processing the generated tasks. The achieved results show that this method outperforms some of the previous resource allocation approaches.

- **A Contract-Based Task Scheduling Algorithm in Combined Fog-Cloud Scenarios with Mobile Fog Servers**

The mobility of fog servers in combined fog-cloud scenarios badly affects the response time and consequently, user satisfaction. In [94], a task scheduling algorithm is proposed, which considers the mobility of fog servers and aims to exploit the computational power of fog servers efficiently, reduce the network load, meet the requirements of delay-sensitive tasks, and process most of the tasks at the edge of the network.

In order to reduce the negative impact of mobile fog servers on the network, this method at first detects the critical fog servers and then by considering different parameters such as communication delay, computational power, and betweenness centrality, finds the most suitable fog servers in a cluster. The critical fog servers will be responsible for creating the operational domain, and the other fog servers will be used for processing the generated tasks by the IoT devices.

The experimental results indicate that this method reduces the average service time and costs and increases the utilization of fog servers and the success rate of tasks.

- **Using Multi-Objective Genetic Algorithm for Workload Allocation in Fog-Cloud Scenarios**

In combined fog-cloud scenarios, there are different parameters that are related to each other. For example, if we assign a task to the fog layer, delay decreases but energy consumption increases. On the other hand, if a task goes to the cloud layer for processing, delay increases, but energy consumption reduces [95]. Therefore, workload allocation algorithms in these environments must consider available bandwidth, delay, energy consumption, and the computational capabilities of each layer to improve the QoS for the users.

For this purpose, authors in [95], proposed a method that utilizes the Multi-Objective Genetic Algorithm [96] for workload allocation in Fog-Cloud Scenarios.

This method formulates a trade-off between energy consumption and delay and then distributes the generated task of the IoT-based applications between the fog and cloud resources. The achieved results show that this task distribution algorithm can improve the delay and energy consumption by assignment of 25% of the tasks to the fog layer.

- **GPRFCA: Gaussian Process Regression for Fog-Cloud Allocation**

Authors in [97] proposed the Gaussian Process Regression for Fog-Cloud Allocation (GPRFCA) for resource allocation in combined fog-cloud scenarios. This approach utilizes a Gaussian process regression to estimate future demands to improve the resource utilization of the fog layer and prevent the requests from being blocked (especially those delay-sensitive ones). This method considers the availability of resources in the fog layer and decides where to process the generated tasks of the IoT devices. In this paper, the system model is considered to be in three layers. There are user devices in the first layer, and in the second and third layers, there are fog and cloud servers. The servers receive the tasks from the IoT-based applications, which collect data from the environment by utilizing the sensors. The tasks are categorized into two groups, namely delay-sensitive and delay-tolerable tasks. The final aim of GPRFCA is to process the delay-sensitive tasks in the fog layer to improve the QoS for the user. For this purpose, the GPRFCA mechanism considers and analyzes the history of sent requests to the servers for the prediction of future demands, and therefore it can reserve the fog resources for future delay-sensitive tasks. The delay-tolerable tasks can be processed in the cloud servers, so the blocking ratio of requests decreases and resource utilization of fog servers increases. In this approach, after assigning the first batch of the tasks to the fog servers, the remaining RAM and CPU resources of the fog layer are calculated, and then by utilizing Gaussian Process Regression, the arrival of future requests can be predicted.

The performance of GPRFCA approach is compared with cloudwards and fogwards methods in terms of blocking ratio of requests, response time, and energy consumption. The results indicate that the energy consumption rate of GPRFCA is always between the rate of fogwards and cloudwards methods and near to the less amount. In addition, this method reduces the blocking ratio of requests and reduces the response time for delay-sensitive tasks.

- **DAOWA: Delay Aware Online Workload Allocation in Combined Fog-Cloud Scenarios**

Dynamic traffic and heterogeneous environment of fog-cloud scenarios cause different challenges for workload allocation. In IoT-based systems, tasks are generated stochastically, and different tasks require different amounts of computation to be processed. Therefore, robust workload allocation methods need to be proposed to be usable in IoT-fog-cloud scenarios with the mentioned characteristics.

For this purpose in [98], an online workload allocation method is proposed, which aims to reduce the task service delay. This method investigates the stability of queuing systems of fog and cloud servers by Lyapunov drift plus penalty theory and then distributes the workload between the fog and cloud servers based on the performed analyses.

The achieved results show that this method reduces the average task service delay compared to similar workload allocation algorithms. The results also indicate that as the task arrival rate and instruction length increase, the DAOWA method performs much better than the competitor methods.

3.2.2 Other Complex Methods

- **F2C: Fog to Cloud Architecture**

In order to improve the response time for the IoT-based applications, authors in [82] considered the challenges of Fog and proposed the F2C architecture as a solution. In this approach, the edge devices utilize the capabilities of Cloud, and play the role of fog servers for the users by providing different services. The F2C architecture benefits from a layered management structure that combines the fog and cloud layers and offers service parallelization. The advantage of this approach is that it lets the user's device to choose the suitable fog or cloud server on the fly. The final goal of F2C is to provide parallel execution and improve execution time, resource utilization, and data security.

In order to investigate the performance of F2C, the authors used that for processing the tasks of a medical emergency application and compared its performance with a cloud-based scenario, in which the data is communicated between the user and cloud server through the Internet, and tasks are processed in Cloud. At first, in F2C architecture, an initial discovery process is performed for finding the closest fog server, and then the tasks are distributed between the fog servers to be processed. The achieved results show that F2C improves the performance of the cloud-based scenario in terms of execution time by 8.6 percent.

Furthermore, authors also discussed an optimized version of F2C (OF2C) in which the number of fog servers must not be equal to the number of tasks, as the fog servers have enough capabilities to process several tasks simultaneously. Moreover, in OF2C architecture, the tasks can be processed in parallel. The achieved results indicate that both F2C and OF2C perform better than the cloud-based approach, and OF2C improves the performance of F2C and reduces the execution time by 21 percent.

- **FBRC: Fog-Based Region and Cloud**

In order to reduce the response time for delay-sensitive applications, in [26], a task scheduling approach is presented. In this approach, fog-based regions and Cloud are considered for processing the generated raw data of the IoT devices. It is worth noticing that in this approach, the tasks can be processed not only by one region but in several regions when more resources are required. The regions can be composed of one or several fog servers, and their resources (and the location of these resources) change dynamically over time. In each region, a fog node is selected to handle the join/leave requests of fog nodes, send and receive the requests, and perform task scheduling for sending a request to the most suitable server at each time slice. Besides, the fog managers are also responsible for collecting information about the other regions to be informed about their resources. In this approach, the task scheduling algorithm works as follows:

1. the available tasks that must be processed are sorted with regard to their latency requirements in ascending order.
2. the computational resources are allocated based on a policy, which considered the delay sensitivity of the tasks.
3. the newly arrived and remained tasks are again sorted with regard to their latency requirements in ascending order, and step 2 is repeated.

Comparing the results with the cloud-based and fog-based methods shows that FBRC reduces the response time.

- **A Policy for Minimizing Delay in Fog-Cloud Networks**

In [91], Yousefpour et al. proposed a general framework for IoT-Fog-Cloud scenarios. Also, to reduce the delay of fog devices, they proposed a policy that aims to reduce the service delay for IoT-based applications. This policy utilizes fog-to-fog communications for sharing the workloads. In order to offload the computational tasks, this policy considers the length of the queue and the different types of tasks (that have different processing times). It must also be mentioned that this method can be used in any network architecture.

Moreover, in this method, task allocation is made based on the response times of fog servers, which depend on different factors such as the computational power of servers and their workloads, queue status, and task type. If the estimated response time of a task was less than a threshold (this amount depends on the incoming traffic from the IoT devices to the servers [91]), the fog server accepts the task and processes it. On the other hand, the fog server offloads the task to another fog server or the cloud. The achieved results show that using this method for processing the light tasks in the fog servers significantly reduces the average delay.

- **Efficient Utilization of Fog and Cloud Resources by Using A Novel Module Mapping Algorithm**

In order to use the network resources efficiently, and for providing the QoS (by reducing the latency) for delay-sensitive applications, in [85] a module mapping algorithm is proposed for efficient task distribution between the fog and cloud resources, which considers the task requirements and the available computational resources of the network.

In this paper, the case study is a real-world delay-sensitive IoT application, which processes the collected data by the sensors and then sends the results to the users. In order to distribute the tasks of the mentioned application, the proposed task assignment algorithm works as follows:

- In the first step, the tasks and the fog servers are sorted in ascending order with regard to their delay requirements and computational capabilities, respectively.
- Then, the algorithm tries to find the most eligible server for processing a task (an eligible server can meet the requirements of the task).
- If there were an eligible server, the task would be assigned to that server.
- If the eligible fog server got exhausted, or if there were no eligible fog server, then the task would be assigned to the cloud servers.

As discussed, this method tries to distribute the tasks between the fog servers and only uses the cloud in specific situations. The achieved results show that the presented algorithm improves Internet bandwidth utilization, delay, and energy consumption compared to a cloud-based approach. As an advantage, it can be said that this method is generic and can be used in any network architecture.

- **A QoS-Aware Service Distribution Approach in Fog-Cloud Networks**

In order to reduce the delay and energy consumption, in [99] authors proposed a QoS aware service distribution method, which efficiently utilizes the edge devices and considers the available computational resources in the network as well as the service requirements.

In this method, data communication between the user and fog/cloud servers is based on the services that a server can offer to the user. For example, if a server can provide service types A and B and process approximately two tasks simultaneously, the user must send only two types A or B tasks, or one type A and one type B task to the server. This policy prevents failures that are caused by fully occupied servers. The experimental results show that this method optimizes the service allocation in combined fog-cloud scenarios and prevents server under-utilization.

- **ReRaP: Resource Ranking and Provisioning algorithm for IoT Applications in Combined Fog-Cloud Scenarios**

The network users might change their behaviors for different reasons, such as their budget, workload, etc. This dynamic user behavior has not been discussed in the previous works. In order to handle the dynamic requirements of IoT-based applications, authors in [100] proposed a method for resource provisioning in combined fog-cloud scenarios, which aims to reduce the costs, delay, and processing time.

Firstly, this method ranks the current resources of fog and cloud layers with regard to their limitations and then provides computational resources for the users to meet their dynamic requirements. This method aims to use the fog devices in the proximity of the users, and if the fog servers were not eligible to process a task, then the task would be forwarded to the edge servers, and if they were not also eligible, then the task would be sent to the cloud servers for processing. An eligible server must have high computational power, be connected to the user with high bandwidth, and provide a suitable response time for delay-sensitive tasks of the users.

In order to rank the computing resources, their processing time, available bandwidth, and response time will be considered. This method ranks the servers with regard to the required time unit for processing a task. For example, if the required computational power by the user was less than the power of a resource, then the rank of that resource is equal to 1, and if that was more than the power of that resource, then the rank will be rounded up to the nearest value. After the ranking process, this method checks the ranking of resources in each layer (fog, edge, and cloud) and tries to find eligible resources in the fog, edge, and cloud layers, respectively. The experimental results show that the ReRaP algorithm reduces the costs, processing time, and delay compared to similar methods.

- **A Fuzzy Logic Approach for Task Scheduling in Combined Fog-Cloud Scenarios**

The fog-cloud networks are scalable, and therefore huge numbers of users can communicate data with the fog or cloud servers. These users have some preferences, while the servers are limited in terms of computational power. In order to do a trade-off between the user preferences and limitations of servers, in [101] authors presented a fuzzy logic based method, which ranks the servers by using linguistic and fuzzy quantified proposition and aims to assign the tasks of the IoT-based applications to the most suitable servers. The achieved results show that in comparison to other proposed algorithms, this method improves the user sanctification and energy consumption but fails in terms of execution time.

3.3 Investigating the Previous Works

This section reviewed the proposed task distribution algorithms, which can be used in combined fog-cloud scenarios. Table 3.1 summarizes the previous works. As can be seen, in most of the earlier researches, the delay has been considered an important parameter. Resource utilization, energy consumption, cost, and blocking rate of the tasks were other measured parameters by the researchers.

In section 2.4.10, we discussed about the huge amount of data that are generated by the IoT devices and the problem of Internet bandwidth utilization by the IoT devices. Moreover, in section 2.5, we discussed about the response time as the most important requirement of the delay-sensitive applications. As shown in Table 1.3, none of the reviewed papers have considered these two parameters simultaneously, and the impact of variation of response time on Internet Bandwidth Utilization has not been investigated. In addition, the discussed methods are limited, and they can only be used for specific purposes. For example, the proposed method in [11, 91] only aim to reduce the response time, and therefore they are not suitable for a delay-tolerable application, which aims to reduce the IBU, costs, or energy consumption. As another example, the proposed method in [12] is only suitable for improving energy consumption, so it is not useful for an application that needs to reduce the bandwidth utilization. Moreover, most of the proposed methods do not consider the dynamic behavior of servers and the fog-cloud network, which directly impacts the performance of any task placement technique.

As the combined fog-cloud scenarios are heterogeneous, as there are various types of applications in the world of IoT (which have different requirements), and as the number of IoT-based applications is increasing, a robust technique for distributing the IoT tasks with regard to the application requirements is needed. In this dissertation, we aim to propose a scalable approach that can distribute the tasks with regard to the application requirements, and be used in different network architecture with different number of servers and users.

Table 3.1: The reviewed methods in this section

Name of Method	**Solution**	**Discussed Variables**
F2C [82]	Service Parallelization in Fog-Cloud Servers	Execution Time
GPRFCA [97]	Using Gaussian Process Regression for predicting the future demands of the network	Delay, Energy Blocking Rate
FBRC [26]	Service Parallelization in Fog Layer and giving more priority to delay-sensitive tasks	Response Time,
VFR [11]	Utilization of Cloud, only when the best fog server fails	Response Time, Resource Utilization
LBP-ACS [12]	Utilization of Laxity based priority algorithm and Ant Colony for optimal scheduling	Failure Rate, Energy
Delay Minimizing Policy [91]	Sharing the workloads between the fog-cloud servers by considering the length of queue and the types of tasks	Delay,
Module Mapping Algorithm [85]	Considering the available omputationl resources and task requirements for workload distribution	Delay, Energy, Network Usage
DAOWA [98]	Investigating the stability of the queuing systems of servers by Lyapunov drift and penalty theory	Delay
QoS Aware Service Distribution [99]	Considering the available computing resources and service requirements, for task distribution	Delay, Resource Utilization
CAG [13]	Using Genetic Algorithm for trade-off between the cost and meeting the deadlines of IoT tasks	Cost, Success Rate
ReRap [100]	Resource Provisioning by considering the dynamic behaviors of the users	Delay, Cost Processing Time
Dynamic Resource Allocation [14]	Considering the network and computing load at the time of task distribution	Delay, Resource Utilization
Contract-based Task Scheduling [94]	Task distribution by considering the mobility of fog servers	Cost, Delay, Resource Utilization
Multi-Objective Genetic Algorithm [95]	Using multi objective genetic algorithm for optimizing the delay and energy consumption	Delay, Energy
Fuzzy Logic Approach for Task Scheduling [101]	Using fuzzy logic for trade-off between the user preferences and limitations of servers	Delay, Energy

4 Case Study and Primary Experiments

4.1 Case Study

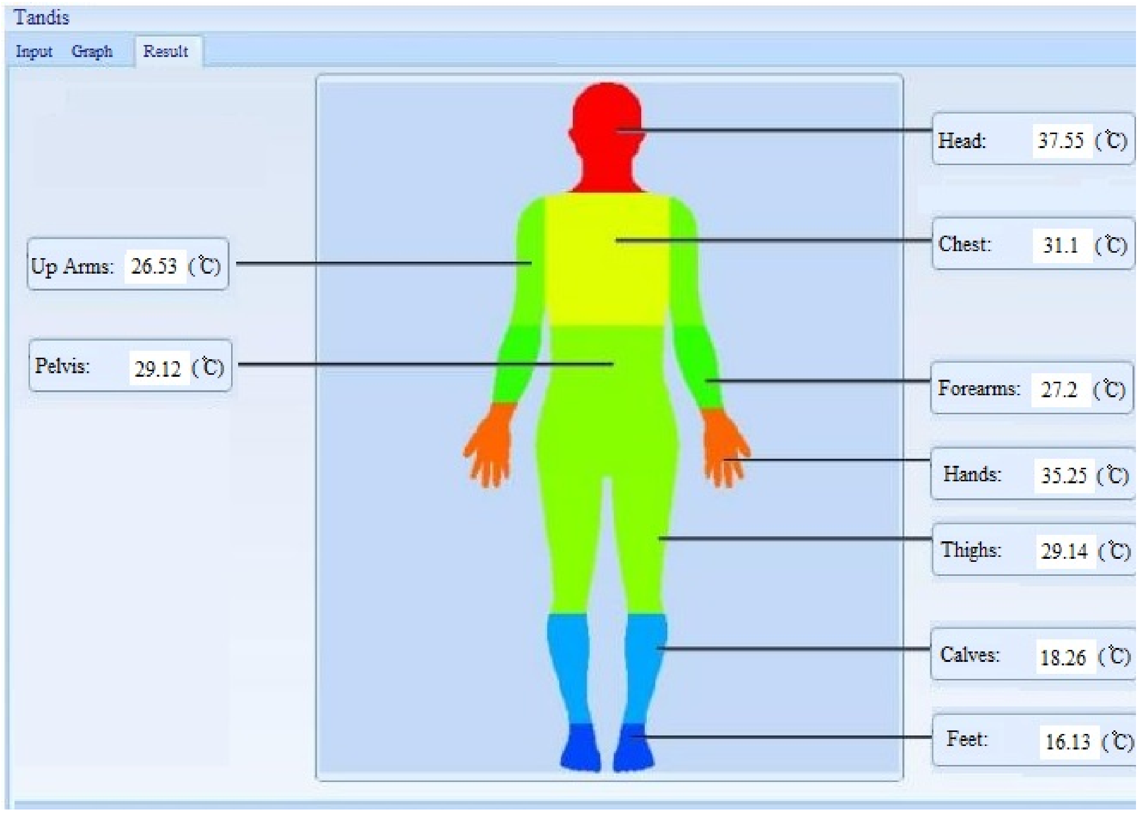
As is reported in my publication [27]; analyzing the reaction of the human body to different environmental factors like humidity, wind velocity, and temperature, in order to identify the best condition in which the body is in the heat comfort condition is not easy and many complicated calculations and models are needed.

There are various researches in the State of the Art, which proposed some models to simulate the heat transfer in the human body. One of the first models was presented by Stolwijk [102] in which the impact of cloth on the body temperature was not considered. This problem was resolved later by Haslam and Parsons in [103, 104]. In 2006, Salloum [105] proposed a complex model, which considers the effect of the respiratory system, sweating, blood perfusion in the tissue, heat transfer in the artery and vein, and contraction of blood vessels in different conditions [27].

As presented in [27], the case study that we have used in our experiments is an online delay-sensitive healthcare application (TANDIS) produced by Rayan Tahlil Sepahan Company. This application combines Stolwijk [102] and Salloum [105] approaches and is specially designed to analyze the effect of different working conditions on the body of workers who work in environments with high amounts of heat stress, such as iron and steel industries. Using the mentioned application, we can estimate the change of water level and the temperature of skin, core, vein, and artery in different segments of the body by using WBSNs and environmental sensors. As a result, workers will be informed about their health status and specific working schedules for any worker can be prepared, aiming to keep them safe and healthy in their working environment. In the following, we explain how this application works.

In this application, the inputs can be set manually and automatically. The environmental sensors and WBSNs are responsible for providing the inputs (information about the human body and environmental parameters) such as skin temperature, sweat rate, heart rate, air temperature, relative humidity, wind speed, and activity of the user, and sending them to the application.

Other information such as type of clothes, type of textile, the thermal resistance of cloth, body surface area, radiation temperature, metabolic rate, gender, weight height, layers of cloth, solar flux, and exposure time must be set manually. In the next step, the application processes the received data, and then as is shown in figure 4.1, it estimates the temperature of skin, core, vein, and artery in different segments of the body, and also provides some information like the change of water level and so on [27].



Head: 37.55 (℃)
Chest: 31.1 (℃)
Up Arms: 26.53 (℃)
Pelvis: 29.12 (℃)
Forearms: 27.2 (℃)
Hands: 35.25 (℃)
Thighs: 29.14 (℃)
Calves: 18.26 (℃)
Feet: 16.13 (℃)

Figure 4.1: Illustration of the temperature of different parts of body in TANDIS application

4.2 Comparing the Performance of Fog and Remote Servers

In order to investigate the performance of Fog and Cloud servers in terms of response time and IBU, we utilized both servers for processing the tasks of the discussed healthcare application in section 4.1. The results are presented in [27], and discussed in the following subsections.

4.2.1 Research Questions

In [27], our main purpose was finding the answer to the following questions:

- How long does it take for the fog and Cloud servers to process a task?
- How much is the latency between the user and the remote server?
- How long does it take for the user to receive the response from the servers?
- How much bandwidth can be saved by utilization of Fog servers instead of remote servers?

4.2.2 Assumptions

In this experiment, we assumed that:

- tasks are sent to the servers one by one and not in batches.
- there is no power restriction.
- data is transmitted reliably, and there is no radio frequency interference.
- the fog servers compress the data after sending the response to the users.

4.2.3 The Experimental Results

As reported in [27]; in our experiment, we considered a person who is equipped with several wearable sensors that are responsible for activity recognition and measuring the temperature, sweat rate, and heart rate in different parts of the body. We also assumed that several environmental sensors are utilized to measure the temperature, humidity, wind velocity, and other parameters of the working environment. Since in our experiments, utilization of real sensors was not necessary (because we only needed the raw data to be assigned to the servers), we generated the sensors data based on a random approach. Moreover, there was also a broker in the network architecture, which was responsible for receiving, aggregating, and sending the raw data (collected by wearable and environmental sensors) to the service providers (Fog server or a remote server in Cloud). It must be mentioned that we used real-world devices for this experiment. As the main differences between Fog and Cloud servers are their computational power and latency to the user, we used different computers in terms of hardware and used the more powerful one as the cloud, and the weaker one as the fog server. In addition, we put the Fog server in the local area network so that it was accessible through an ad-hoc communication, but the cloud server had a public IP address, which was only accessible through the Internet.

Regarding [27], as the architecture and technical details of WBSN and environmental sensors do not affect the response time and the Internet traffic, we only considered the broker, Fog and the remote server (with equal hardware and computing power) in the network architecture. In the first scenario, we installed both the client and server sides of the application on the server to measure the run time, delay, and response time on the localhost, which were equal to 0.50, 1.40 and 1.90 seconds respectively. The run time is the time that CPU spends for processing a batch of data, and the response time is considered as the run time together with I/O, connection, transmission, and propagation delays between the user and the server. In the second scenario, as illustrated in figure 4.2, there is a broker which is responsible for receiving the raw data from WBSNs and environmental sensors, performing some preprocessing on the received data, and sending the preprocessed data to a configured remote server in Cloud through the Internet by using TCP/IP communication protocol. After this step, the remote server starts processing the raw data, stores it, and sends the result to the user.

As is discussed in [27]; in the fog-based scenario as depicted in figure 4.3, the broker sends the raw data to a Fog server, which is one of the idle computers around the user in the local area network (LAN). Generally, Fog servers could be located anywhere between the cloud and user, but in this experiment the fog server is accessible via one-hop ad-hoc communication in LAN and could be discovered as it announces its computation power and address by voluntary advertisements in short periods of time. In this scenario, the broker receives the raw data from the sensors, does some pre-processing tasks, and sends the pre-processed data to a Fog server.

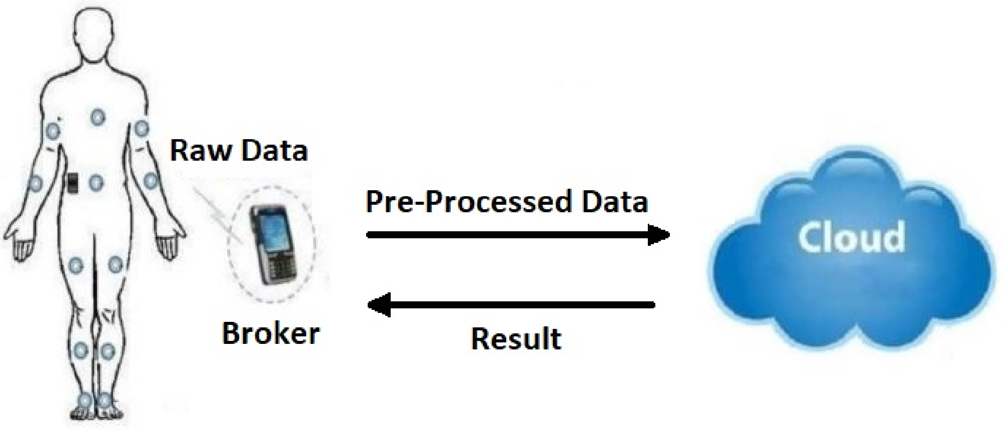


Figure 4.2: Cloud-Based Scenario

Table 4.1: Specifications of the used devices in [27]

Specification of Devices	**Broker**	**Fog Server**	**Remote Server**
Type of Device	Windows tablet	PC	PC
OS	Windows 8	Windows 8	Windows 8
CPU	Intel Core i5-6300U	Intel Core i5 8250U	Intel Core i5 8250U

Table 4.2: Details of the real-world experiment in [27]

Implementation Details	**Cloud Computing**	**Fog Computing**
Communication Protocol	TCP/IP	TCP/IP
Location of Server	Internet	LAN
Available Bandwidth	4.2 Mbps	300 Mbps
Distance Between User and Server	13 hops	1 hop
Average Response Time	3.55 sec	1.9 sec

The fog server starts processing the received data, sends the result to the user and also transfers the compressed result to the cloud for further processing and permanent storage.

As reported in [27], Cloud provides much more powerful resources compared to the fog, but as in our case study only a few amounts of raw data must be processed in the server, and as we aim to investigate the effect of the distance between the user and server on the response time, we have used servers with similar computational power in both scenarios. Additional specifications of the used devices in this experiment are presented in Table 4.1, and implementation details can be found in Table 4.2.

In all scenarios that are considered in [27], delay, response time, and run time are measured, and results are presented in figures 4.4 and 4.5 as well as Table 4.2. It must also be mentioned that in all scenarios the amount of the produced data by the sensors, which are aggregated, pre-processed and available in the broker is almost 2 KB and the amount of the processed data in the server is almost 90 KB. Therefore, in the cloud-based scenario, 2 KB during the upload and 90 KB during the download process is communicated between the user and remote server through the Internet. In contrast, in the fog-based scenario, both produced and processed data are communicated in the local area network and only the compressed results are communicated between the fog and remote server over the Internet. The applied compression method in Fog server is LZW [21], which is able to reduce the amount of the processed data up to 77% in 0.03 seconds (from 90 KB to 21KB) in our case study. Therefore, in the fog-based scenario, only the compressed result is sent to the cloud, which consequently reduces Internet traffic.

Furthermore, the average results in [27] show that using Fog computing also improves the response time by 46% in comparison to Cloud. As is illustrated in figure 4.5, in the fog-based scenario, no significant variation in response time is seen, and results are actually the same in all the experiments, while in the cloud-based scenario, tiny variations were observed. The reason is apparent, as in the cloud-based scenario, both request and response are communicated through different routers over the Internet, unlike the fog-based approach in which data is communicated in LAN. The average response time in the fog-based scenario is 1.90 seconds, while Cloud provided an average response time of 3.55 seconds. It is worth mentioning that the discussed results are provided by real measurement. The bandwidth utilization is the amount of communicated data between the users and servers, and the response time is the sum of run time (on the server) and the delay between the user and server.

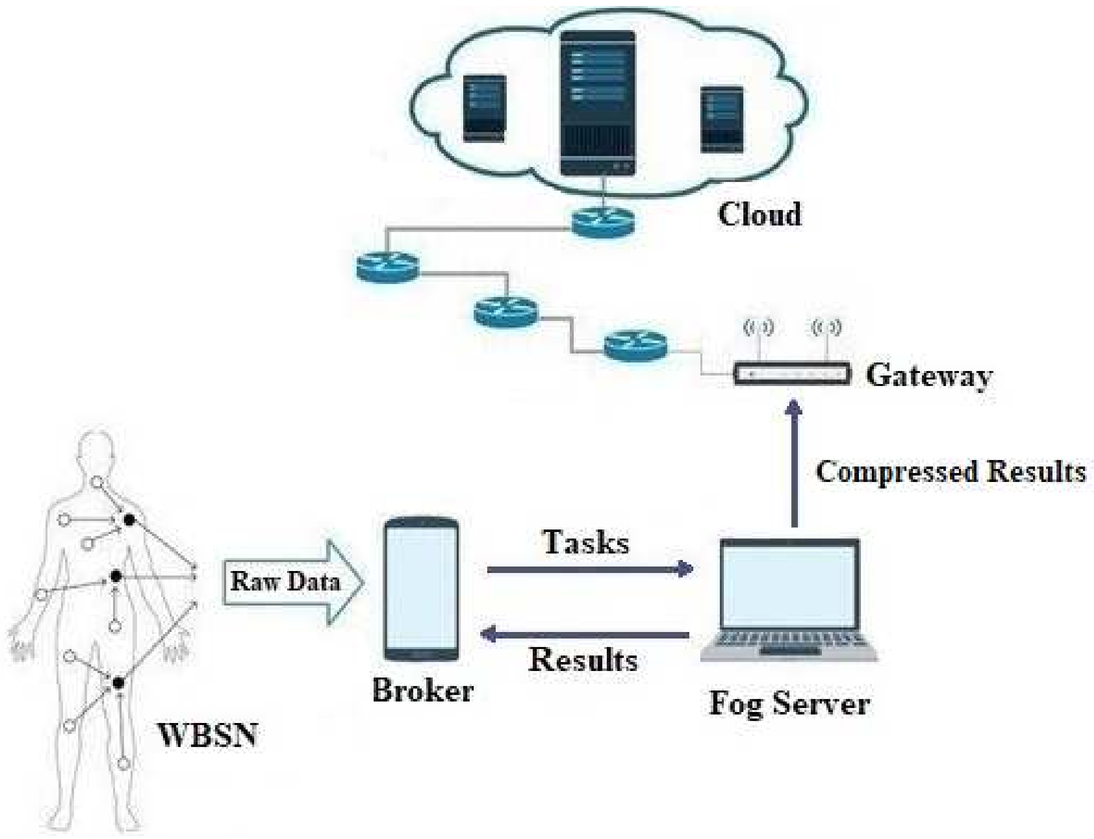


Figure 4.3: Fog Computing Based Scenario

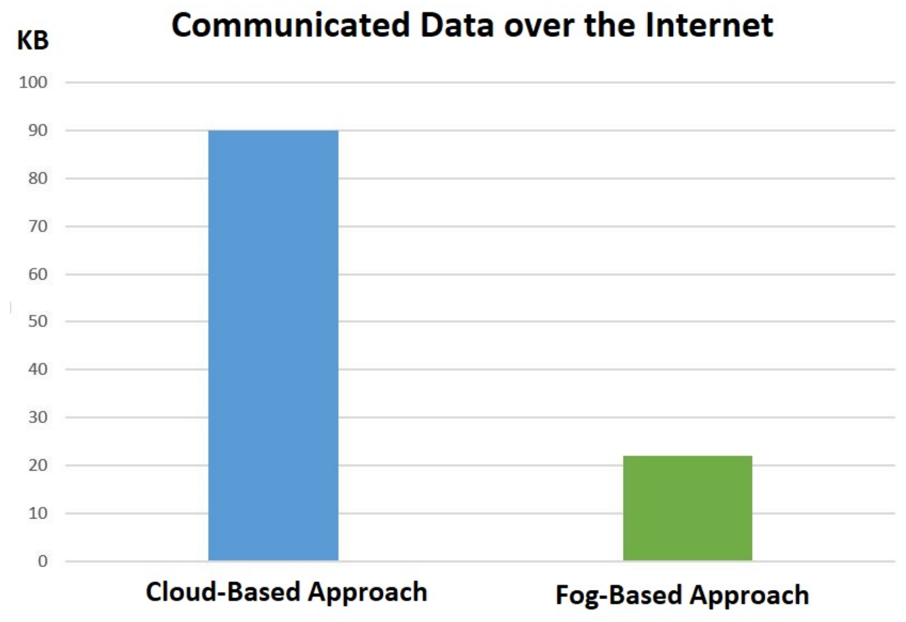


Figure 4.4: Comparing the performance of Cloud and Fog in terms of Internet traffic, published in [27]

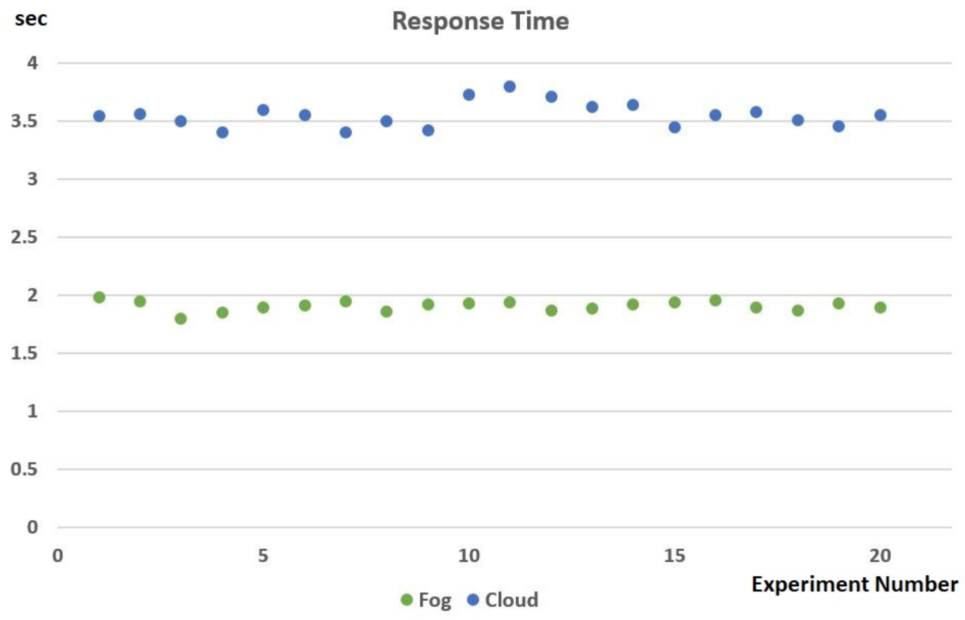


Figure 4.5: Comparing the performance of Cloud and Fog in terms of response time, published in [27]

In addition, we also performed another experiment with different servers and observed that the fog server processed the generated raw data by the IoT devices in 1 second, and the remote server provided a processing time of o.5 seconds. In this experiment, the delay of Fog and Cloud servers was also 1.4 and 2.6 seconds, respectively. Figure 4.6 compares the performance of Fog and Cloud servers in terms of response time.

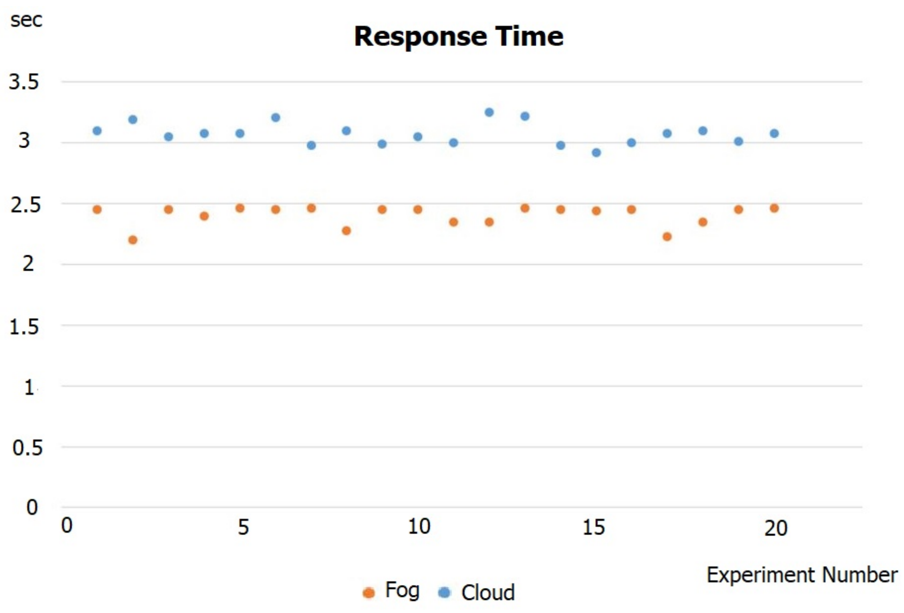


Figure 4.6: Comparing the performance of Cloud and Fog in terms of response time

Regarding [27], and as is depicted in figure 4.5-6, Fog computing provided a better QoS compared to Cloud by providing an improved response time and reduced Internet traffic. Reducing the network traffic improves data transmission and relieves bandwidth; therefore, the bandwidth can be reserved for other important users and applications. In addition, providing reduced response time is always required by delay-sensitive applications, especially healthcare applications in which even seconds are important.

But, it must be mentioned that when we use a more robust remote server (or a very weak fog server), the response times of Cloud and Fog layers get closer to each other. Generally, for one by one task processing, as is discussed in the state of the art, Fog servers provide a better response time, and Cloud servers fail because of their high latency to the users.

5 Intelligent Task Placement in Combined Fog-Cloud Scenarios

In section 1.2, we discussed about the task distribution challenges in combined Fog-Cloud scenarios. In this chapter, we raise several research questions about resource allocation and task placement in Fog-Cloud networks and propose solutions for the discussed research problems.

Firstly, we start with a network architecture in which Fog and Cloud servers have similar workloads, different computational powers, and latencies. We also assume that different numbers of tasks can be available in the broker at each time slice. In the following subsections, we discuss about the task distribution problems in such a network scenario and propose solutions for them. After each step, we make the network architecture more complex and develop our primary method to be usable in different network architectures.

5.1 Smart Task Distribution Between Fog and Cloud Servers with Similar Workloads

5.1.1 Network Architecture and Assumptions

As we discussed in the previous sections, Fog and Cloud servers are different in terms of computational power and latency to the user. Cloud servers are more powerful than Fog servers, but the fog layer is closer to the user, which means it has less latency. Therefore, in order to do experiments, we considered the network architecture that is depicted in figure 5.1 as our test bed. As can be seen, this network contains a user (equipped with WBSNs), broker, Fog server, and a remote server in the cloud. We set the computing capability of the cloud server to be almost twice that of the capability of the fog server [30]. Moreover, in our experiment, the fog server is accessible in the local area network through wireless ad-hoc communications, and the cloud server has a public IP address, which can be accessed via the Internet. We also assumed that:

- there is no power restriction, and nodes are homogeneous.
- data is transmitted reliably, and there is no radio frequency interference.

- servers in both Fog and Cloud have similar (light) workloads.

More details about this network are presented in table 5.1.

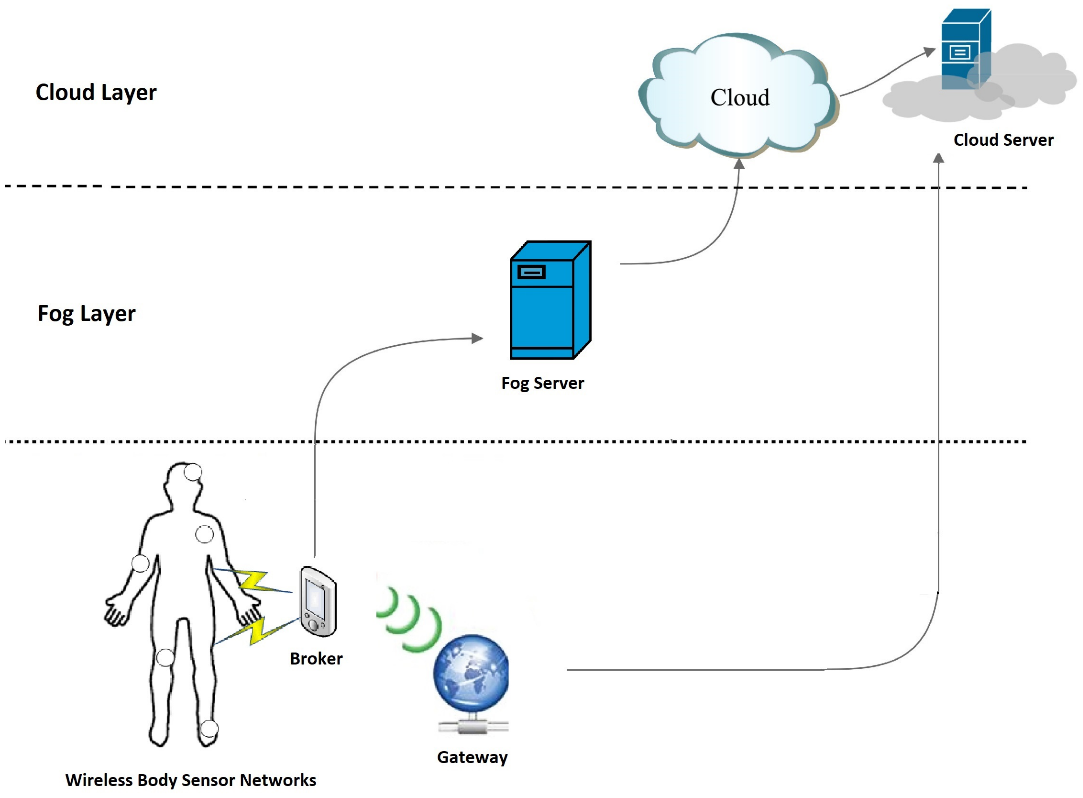


Figure 5.1: The Network Architecture

Table 5.1: The Implementation Details of our Experiment in [16]

	Cloud	**Fog**
Communication Protocol	TCP/IP	TCP/IP
Communication Between User and Server	Through the Internet	Ad-hoc
Location of Server	Internet	LAN
Available Bandwidth	4.2 Mbps	400 Mbps
Distance to User	13 hops	1 hop

5.1.2 Problem Description and Research Questions

In order to process the provided raw data by the IoT sensors, as discussed in [30], on one side, there are Cloud servers which have powerful computing resources but are located far from the users (with high latency to the users), and on the other side there are Fog servers which have limited computing capabilities but are located closer to the users (with low latency to the users).

Regarding [30], where both Fog and Cloud resources exist, the first challenge is that the broker is not able to distinguish the best resource (between the fog and cloud servers) for processing a batch of data. Also, when the number of available tasks in the broker increases, using only one resource (cloud or fog) is not efficient, as the best response time can only be achieved when both Fog and Cloud servers have the least idle time. Therefore, the second problem is that the broker is not able to distribute the available tasks in the most efficient way for reducing the response time and the Internet bandwidth utilization. Thus, an appropriate task distribution algorithm in combined fog-cloud scenarios is required to utilize the fog and cloud resources in the most efficient way, and reduce the response time and IBU as much as possible.

So as reported in [30], the research questions can be formed as follows:

- When is it efficient to use Fog servers?
- When do the cloud servers perform better than Fog servers?
- How is it possible to make the broker intelligent to predict the response times of servers?
- How should the broker distribute the tasks between the fog and cloud servers?
- How much does the intelligent task distribution reduce the response time and IBU?

5.1.3 AITDA: An Artificial Intelligence Based Task Distribution Algorithm

As is presented in [30], in order to deal with the discussed challenge in section 5.1.2, we propose an Artificial Intelligence based Task Distribution Algorithm (AITDA), which makes the broker smart and able to predict the processing time and the size of the result of a received task (from the user). This prediction process can be performed by using one of the function approximation methods (FAMs) [30]. As discussed in [1], the reason for using the FAM in such a scenario is that some applications (like the healthcare application in our case study) provide an infinite state of inputs with those continuous variables that their variation affects the output. In these scenarios, using other methods such as decision tables is not possible, and we need to use the FAMs to predict the output. In this research, we chose the ANNs [106] as the FAM because they have been used in the literature for resource allocation purposes [107, 108] and provided acceptable results. They are also ready to use and only need to be trained [30]. However, other similar methods such as Support Vector Machines [109] can also be used in these scenarios, but as comparing the performance of different FAMs is out of the scope of this thesis, we discuss about their performance in our future works. In the following steps (as discussed in [30]), we explain how the ANNs can be set up in the broker:

1. User sends different tasks to the broker.
2. The broker randomly sends the received tasks to the fog and cloud servers.
3. Fog and cloud servers process the tasks. For each task, they log the received data from the broker, as well as the processing time and the size of the result of the task.
4. Each of the servers train an ANN in which the input is the received data from the broker and the output is the processing time and the size of the result.
5. The ANNs will be sent to the broker (then, for each server, the broker has an ANN, which makes the broker able to predict the processing time and the size of the result when a task arrives).
6. The broker distributes the tasks based on the estimated response times and sizes, with regard to the pre-defined policies.

With regard to my publication [30]; different policies can be considered for the distribution of the available tasks in the broker. The policy that we define in this section aims to reduce Internet bandwidth utilization and response time. For this purpose, the available tasks in the broker must be sorted with regard to their predicted size of results in ascending order. The smaller tasks will then be assigned to the cloud, and larger ones will be sent to the fog servers for processing. In order to reduce the response time as much as possible, fog and cloud servers must process the tasks in parallel so that each of them experiences the least idle time.

As reported in [30], to check the performance of AITDA, we randomly generated 1000 different tasks, of which 800 tasks were used for the training process of the ANNs (in fog and cloud servers), and 200 tasks were used for the main experiment.

Each task consists of almost 400 numbers that can be provided manually or automatically by the sensors. It must be mentioned that we used multi-layer perceptron ANNs in MATLAB, which were composed of three layers and using the Levenberg-Marquardt training algorithm. In order to check the accuracy of the ANNs, we compared the predicted amounts with the actual amounts. The results showed that the ANNs could predict the response time and the size of the results with an error of ±0.09 seconds (13%) and ±250 bytes (0.27%), respectively.

The processing time of Fog and Cloud servers for 200 different tasks have been depicted in figure 5.2, which shows that the average processing time of cloud and fog servers are almost 0.5 and 1 seconds, respectively [30]. It is worth mentioning that the average size of results is almost 90 KB.

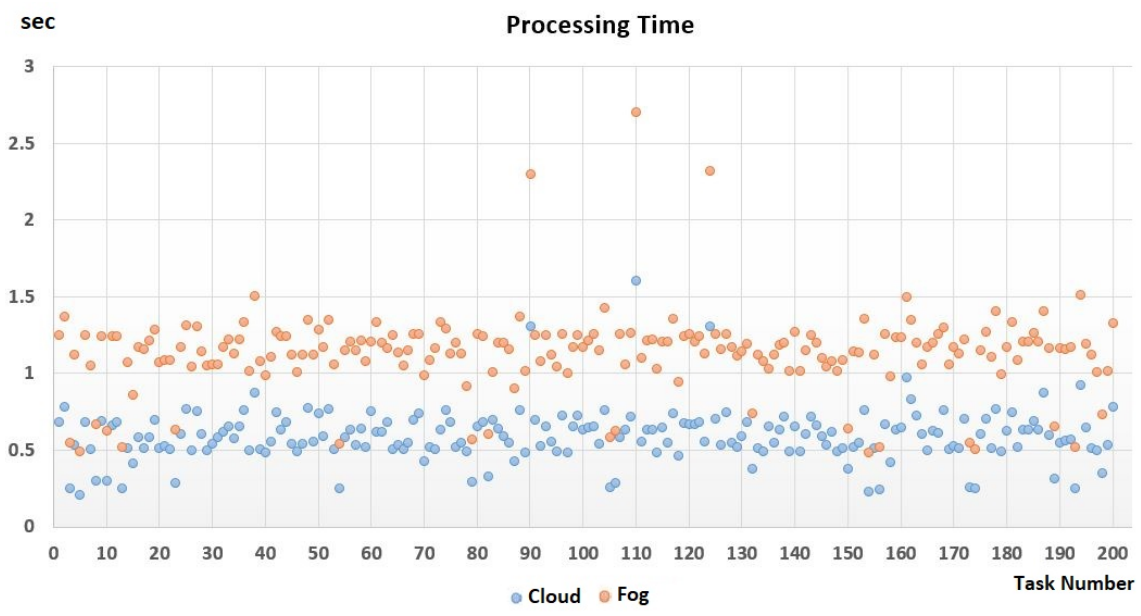


Figure 5.2: Comparing the Processing Time of Fog and Cloud Server

Regarding [30]; in the next step, we set up the trained ANNs in the broker and made the broker able to estimate the response time of the fog and cloud servers as well as the size of the results. Afterward, the broker distributed the tasks based on the discussed policy. It is worth noticing that we assumed both Fog and Cloud servers always work properly, and their workloads were considered to be equal (30%) at the time of task assignment process. Furthermore, the delay is considered to be constant (1.2 seconds for the fog and 2.4 seconds for the cloud servers).

5.1.4 Evaluation of Results

As is published in [30], and depicted in figures 5.3 and 5.4; we compared the performance of AITDA with two competitors, namely cloud-based and fog-based approaches, in terms of response time and Internet traffic. Figure 5.3 shows the fact that by increasing the number of the available tasks in the broker, Cloud performs better than Fog in terms of response time. The reason is the computing capability of the cloud servers, which is more than the power of the fog servers. Therefore, when several tasks are sent to both fog and cloud servers, the processing time of the cloud is less than the fog. This difference in processing time is too high and makes the delay of cloud servers insignificant. As illustrated in figure 5.3, AITDA improves the response time compared to fog-based and cloud-based approaches. The reason is obvious as the AITDA distributes the tasks between the fog and cloud resources. This improvement becomes more significant when the number of available tasks in the broker increases. In addition, as is shown in figure 5.4, AITDA performs better than the cloud-based method and worse than the fog-based approach in terms of Internet Bandwidth Utilization. The reason is that AITDA assigns some of the tasks to the Cloud, but the cloud-based method communicates all the data and responses over the Internet, unlike the fog-based method, which does not communicate any data through the Internet, as Fog servers are located in the local area network.

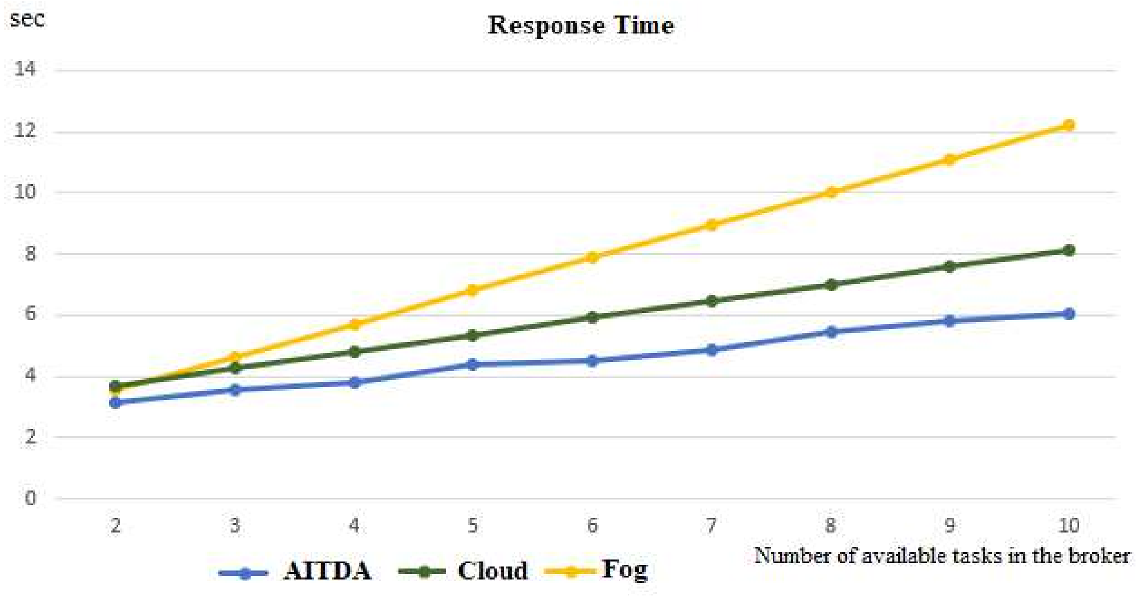


Figure 5.3: The performance of different approaches in terms of response time, published in [30]

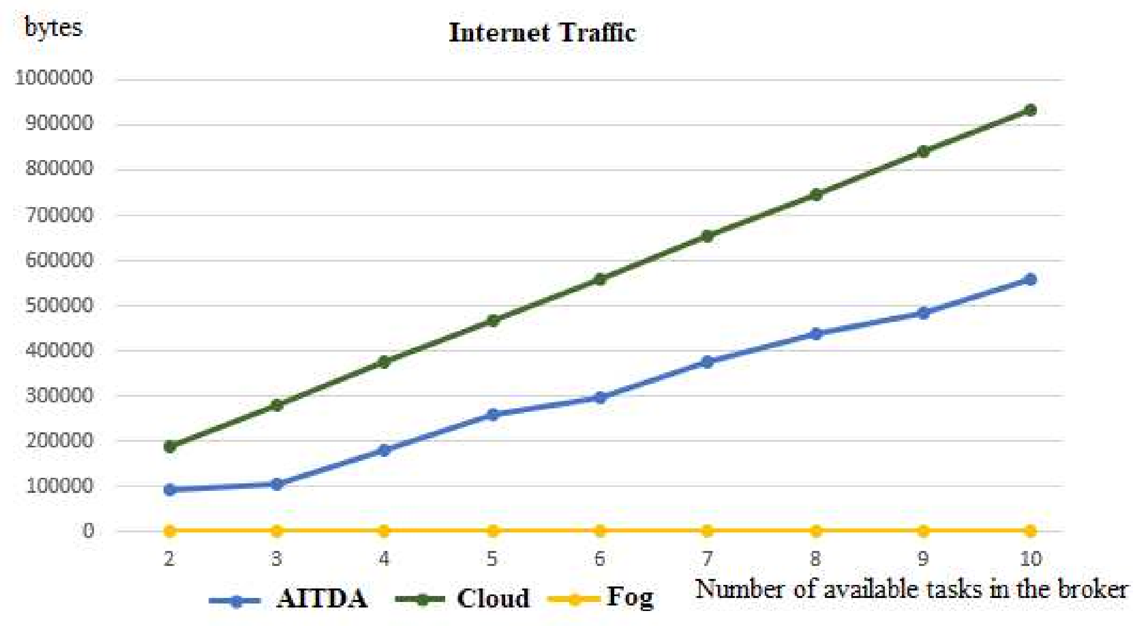


Figure 5.4: The performance of different approaches in terms of Internet traffic, published in [30]

5.2 Fog-Cloud Smart Task Distribution by Considering the Application Requirements

5.2.1 Problem Description

In the previous section, we discussed about AITDA, which its main aim was to communicate the least possible amount of data through the Internet. Therefore, AITDA can be considered as a suitable task distribution approach for delay-tolerant applications. But, as we discussed in section 2.6.2, there are delay-sensitive applications in which data must be communicated between the user and servers with the least possible amount of delay. In order to provide suitable task distribution methods for both delay-tolerant and delay-sensitive applications, we propose an updated version of AITDA, which is called Fog-Cloud Smart Task Distribution (FCSTD) method that considers the application requirements at the time of task distribution.

5.2.2 Proposed Approach

As discussed in [28]; the FCSTD method starts running in broker after the following steps:

1. The data providers generate data and send the tasks to the distributor unit

(broker).

2. The broker randomly assigns the produced tasks to the fog and remote servers for processing in a specific period of time (P_k).
3. The broker logs the tasks that have been assigned to the fog server, in addition to the processing time and the size of the results (provided by the fog server).
4. After P_k, the broker sends the logged data to the remote server.
5. The remote server trains two ANNs: a) In the first ANN, the assigned tasks to the fog server are set as input, and the response time and the amount of the provided data by the fog server are set as output. b) In the second ANN, the received tasks by the remote server (from the broker) are set as input, and the response time, in addition to the amounts of the provided data by the remote server, are set as output.
6. The remote server sends both of the ANNs to the broker after the training process.
7. Then, the broker can utilize the ANNs, such that when a task arrives, the broker sets the task as the input of the ANNs, and then the outputs of the ANNs are the response time of fog and cloud servers, and also the size of results.
8. The broker assigns the received tasks to suitable servers with regard to application requirements.
9. The learning process (steps 3-6) will continue in P_{k+1} for improving the training process of ANNs.

Regarding [28]; when the broker receives the tasks, it must be able to distribute them with specific policies. For instance, a policy can be assigning all the tasks to the fog servers. This can be a good approach for non-delay-sensitive applications as the high response time of the fog servers (because of their limited resources) can be tolerated. Another policy can be based on reducing the response time, which is suitable for delay-sensitive applications. This policy must consider an efficient task distribution between the existing fog and cloud servers. Our proposed method distributes the tasks with two different policies, namely FCSTD traffic-based and FCSTD time-based, that aim to reduce the IBU and response time, respectively.

5.2.2.1 FCSTD Time-Based

As reported in [28]; the FCSTD time-based method is usable for delay-sensitive applications. With regard to figure 5.5, when the broker receives a batch of tasks (N tasks) from an application, the FCSTD time-based method considers the variables in Table 5.2 and then sorts the tasks with regard to their predicted processing times in ascending order.

Table 5.2: Notations (published in [28])

Variable	Description
N	The number of available tasks in the broker
F	The processing time of the fog server, predicted by the ANN
C	The processing time of the cloud server, predicted by the ANN
S	The size of the result provided by the fog or cloud server (the size of results are the same in both fog and cloud servers)
D_c	Delay of communication with the cloud server
D_f	Delay of communication with the fog server
TR_c	Total response time of the cloud server, predicted by the ANN
TR_f	Total response time of the fog server, predicted by the ANN
RP	The Maximum Response time

Regarding [28]; there is always a relation between the F and C (F is always greater than C); the tasks can be sorted by considering either F or C. Then, the possible distribution conditions will be checked. In the first possible distribution, the first task in the queue (N1) is assigned to the cloud server, and N2, N3,... Nn to the fog server. Then TR_c , TR_f , and RP are calculated as follows:

$$TR_c = C_1 + D_c \quad ; \qquad TR_f = F_2 + F_3 + \ldots F_n + D_f$$
$$RP1 = max\,\{TR_c, TR_f\}$$

In [28], it is mentioned that C_1 and F_2 are the processing time of N1 and N2, provided by cloud and fog servers, respectively. In the second possible distribution, the first two tasks of the queue (N1 and N2) are assigned to the cloud server, and N3, N4, ... Nn to the fog server. Therefore, TR_c , TR_f , and RP will be equal to:

$$TR_c = C_1 + C_2 + D_c \quad ; \qquad TR_f = F_3 + F_4 + \ldots F_n + D_f$$

$$RP2 = max\,\{TR_c, TR_f\}$$

This process continues until the last possible distribution in which all of the tasks will be assigned to the cloud server [28]. Therefore, TR_c , TR_f , and RP are calculated as follows:

$$TRc = C_1 + C_2 + \ldots\ldots + C_\mathrm{n} + D_\mathrm{c} \qquad ; \qquad TR_\mathrm{f} = 0$$

$$RPn = max\,\{TR_\mathrm{c}, TR_\mathrm{f}\}$$

Obviously, FCSTD time-based method selects the distribution possibility in which RP has the minimum possible amount [28].

5.2.2.2 FCSTD Traffic-Based

As is discussed in [28], and also in section 5.1.3, after receiving a batch of tasks from the user, the FCSTD traffic-based method sorts the tasks with regard to their sizes in ascending order. In the next step, it checks the possible distributions and selects the best one (similar to the FCSTD Time-Based method).

5.2.3 Evaluation of Results

In this experiment, the assumptions, network architecture, and implementation details are similar to the previous experiment discussed in section 5.1. The technical details of the broker and servers are presented in table 5.3.

Table 5.3: The technical details of the used devices in our experiment

	Broker	**Fog Server**	**Cloud Server**
Type of Device	Laptop	Laptop	Server
CPU	Intel Core i5 6300 U	Intel Core i5 8250 U	Intel Core i7 8700
RAM	6 GB	8 GB	16 GB

In this research (as published in [28]), we investigated the performance of different task placement policies for distributing the different types of tasks. The discussed healthcare application in section 4.1 generates tasks with almost similar sizes (89-94 KB) and processing times (nearly 0.5 and 1 seconds by cloud and fog servers, respectively). In the following, these types of tasks will be called SSP tasks (because they have Similar Sizes and Processing times). In order to apply different distribution policies to this case study, at first, we randomly generated 1000 SSP tasks (800 tasks for the training of the neural networks and 200 tasks for the main experiment). After training the multi-layer perceptron neural networks (composed of 3 layers) with the Levenberg-Marquardt training algorithm in MATLAB, we checked their performance accuracy to predict the response time and the size of the processed data (results). Neural networks were able to predict the response time and size of the results with the error of 0.1 seconds and 250 bytes, respectively.

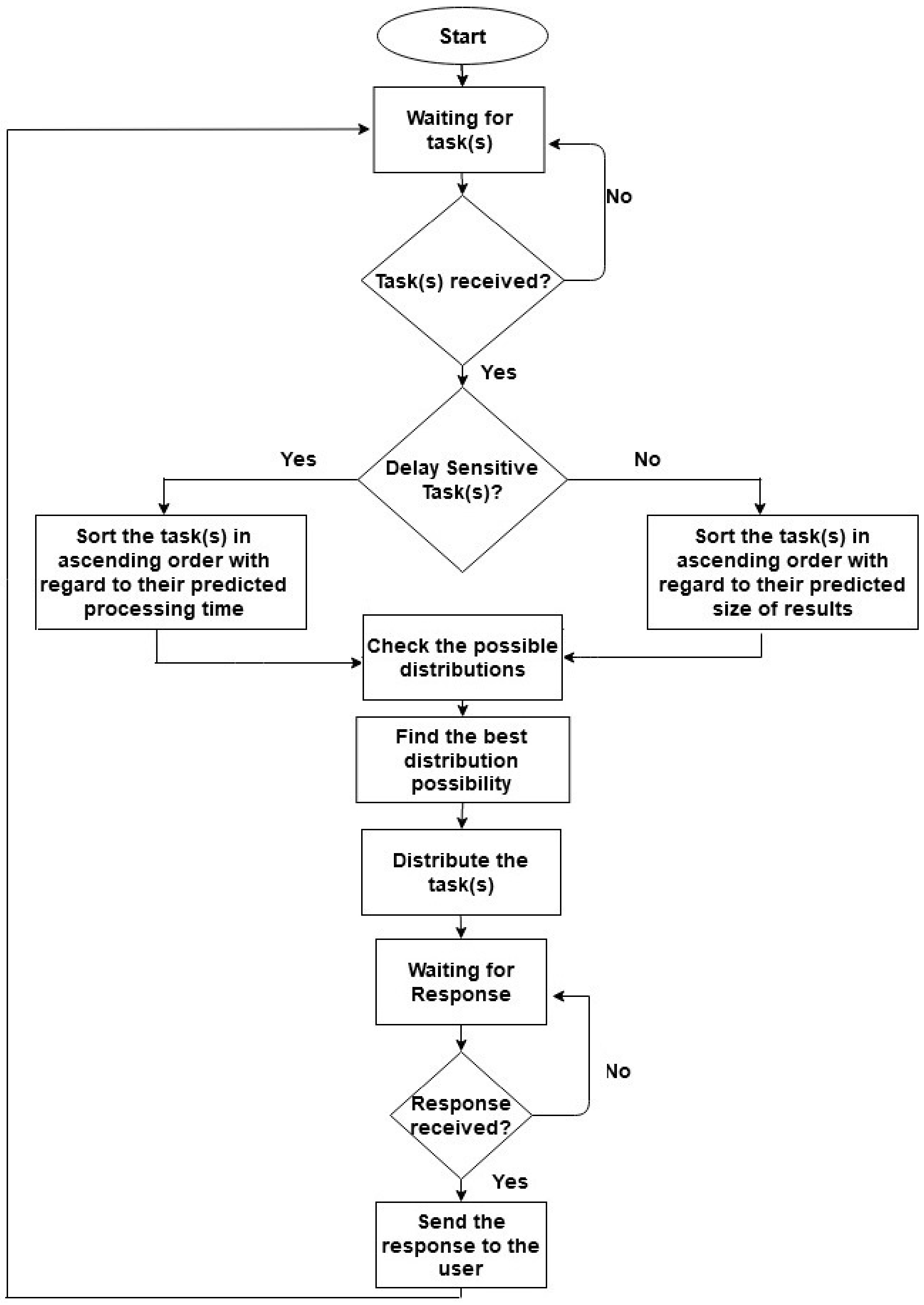

Start
Waiting for task(s)
Task(s) received?
No
Yes
Delay Sensitive Task(s)?
Yes
No
Sort the task(s) in ascending order with regard to their predicted processing time
Sort the task(s) in ascending order with regard to their predicted size of results
Check the possible distributions
Find the best distribution possibility
Distribute the task(s)
Waiting for Response
Response received?
No
Yes
Send the response to the user

Figure 5.5: The Data Flow in FCSTD

As reported in [28], for a thorough investigation of the performance of different distribution methods, we also generated tasks with:

- almost similar sizes but with different processing times (SSDP)

- different sizes but with almost similar processing times (DSSP)
- different sizes and processing times (DSP)

In this experiment [28], the broker is responsible for distributing the tasks between the fog and cloud servers with five different policies, namely:

- Fog-based: that utilizes the fog resources for processing the produced data by the IoT devices.
- Cloud-IoT: the proposed method in [31] which allocates all of the generated data to the cloud for processing.
- Random Fit: that is proposed in [33] that randomly assigns the tasks to the fog or cloud servers.
- FCSTD Traffic-based: that uses the ANNs to distribute the tasks and aims to reduce the IBU.
- FCSTD Time-based: that exploits the ANNs for task distribution between the fog and cloud resources, aiming to reduce the response time.

and after the distribution, it waits for the response of the servers to forward them to the user. It must be noticed that in this experiment, response time is the sum of the processing time, transmission, propagation, connection, and application delay [28]. The results are also based on an average of 20 different experiments with different tasks. The performance of the mentioned distribution methods in terms of IBU and response time are presented in [28] and also figures 5.6- 5.13.

For the SSP tasks, as is depicted in figure 5.6, both versions of the FCSTD method provided the best response time compared to the other methods. The difference between the FCSTD method and the other task placement techniques increases as the number of available tasks in the broker increases. For example, when there are only three tasks available in the broker, our proposed method responds in 3.6 seconds, and the cloud-IoT, random fit, and fog-based methods provide the response in 4.25, 4.6, and 4,63 seconds receptively. But, when there are 10 tasks available in the broker, the response time of FCSTD is almost 6 seconds, and the response times of Random Fit, Cloud-IoT, and fog-based methods are 7, 8.11, and 12.19 seconds, respectively.

It can be generally said that both versions of our proposed method provided similar response times that were much better than the other methods. The Random Fit and Cloud-IoT also performed similarly, and after FCSTD, they were the second-best approaches in terms of response time. Moreover, we also observed that the fog-based method was the worst. The reason is that in the fog-based method, all the tasks get processed in the fog servers (which are limited in terms of computational power), and by increasing the number of available tasks in the broker, fog servers perform poorly.

However, the ranking of methods in terms of IBU is entirely different. As shown in figure 5.7, in the fog-based method, the amount of communicated data over the Internet is always equal to zero. Therefore, this method is the best task placement approach in terms of IBU. The reason is that this method exploits the fog servers accessible in the local area network (in our experiment). Moreover, figure 5.7 also indicates that the Random Fit method communicates fewer amounts of data over the Internet in almost all situations compared to FCSTD and Cloud-IoT methods. It is also clear that the traffic-based version of FCSTD performs slightly better than its time-based version, which is ranked as the second-worst approach in terms of IBU for the distribution of SSP tasks. Moreover, we also witnessed that the Cloud-IoT method is the worst policy in terms of IBU, as in this method, all the requests and responses are communicated over the Internet between the users and the remote servers in the cloud. All in all, it can be said that for the distribution of SSP tasks, our method is always faster than the other task placement techniques, but in terms of IBU, Random Fit performs better in exchange for a significantly higher response time.

For the distribution of the SSDP tasks, as is visible in figure 5.8, both versions of our proposed method provided the best response time compared to other methods in all situations. Moreover, Cloud-IoT and Random Fit methods are ranked as the third and fourth-best approaches for distributing SSDP tasks. However, they performed similarly in most of the situations. It is also clear from the data that the limitation of fog servers in terms of computational power caused the fog-based method to be ranked as the worst method in terms of response time. In terms of IBU, the fog-based and Random Fit methods are ranked as the first and second-best methods for distributing the SSDP tasks. In figure 5.7, we observed that both versions of our proposed method performed similarly, but in figure 5.9, it is apparently seen that the traffic-based version of FCSTD performs much better than its time-based version in all the situations and gets ranked as the third-best approach in terms of IBU. We can also see that the time-based version of FCSTD is ranked as the fourth-best method, and the Cloud-IoT performs worse than all the discussed task distribution techniques. Moreover, as shown in figure 5.10, it is explicitly observed that FCSTD is the fastest method for distributing the DSSP tasks in terms of response time. The diagram also reveals that the Random Fit and Cloud-IoT methods perform similarly, and the fog-based method is the worst task placement approach compared to the other discussed techniques. However, in terms of IBU, the fog-based method is again the best, and the traffic-based version of our method performed better than the Random Fit (which is ranked as the third-best distribution approach). It is also clear that Cloud-IoT is the worst method, and the time-based version of FCSTD performs better than Cloud-IoT and worse than Random Fit in terms of IBU for distribution of DSSP tasks.

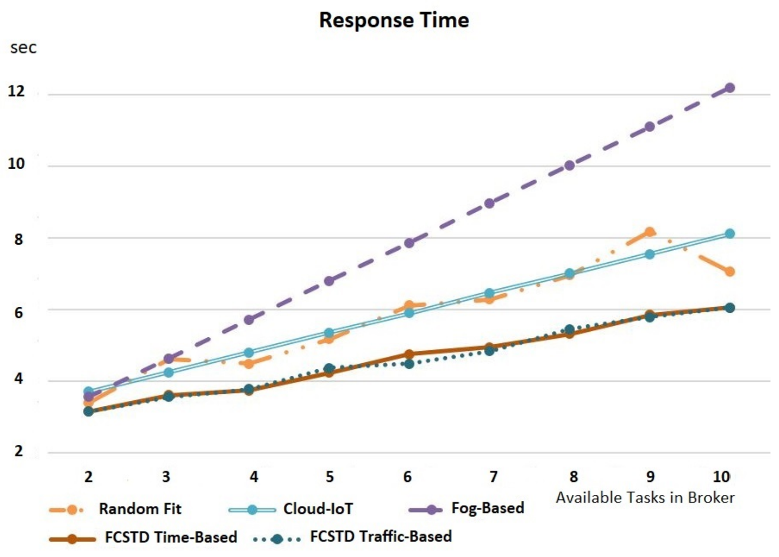


Figure 5.6: The performance of different distribution policies in terms of response time for SSP tasks, published in [28]

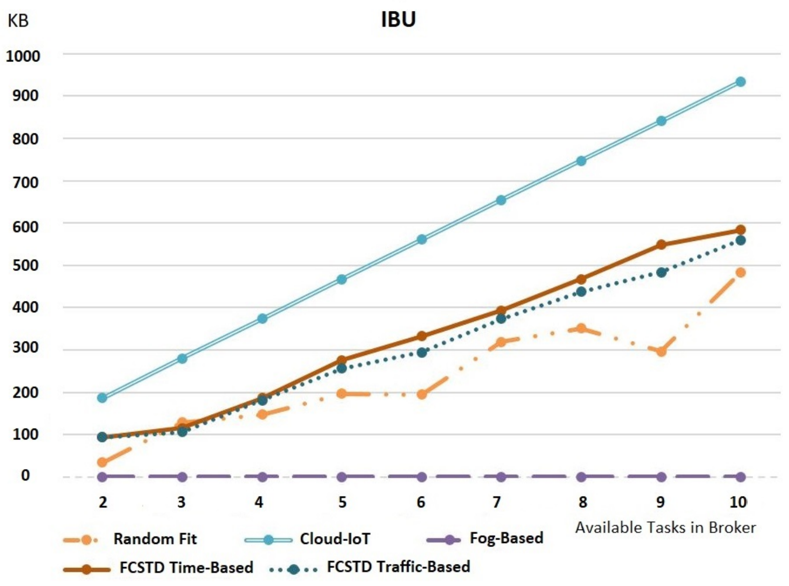


Figure 5.7: The performance of different distribution policies in terms of IBU for SSP tasks, published in [28]

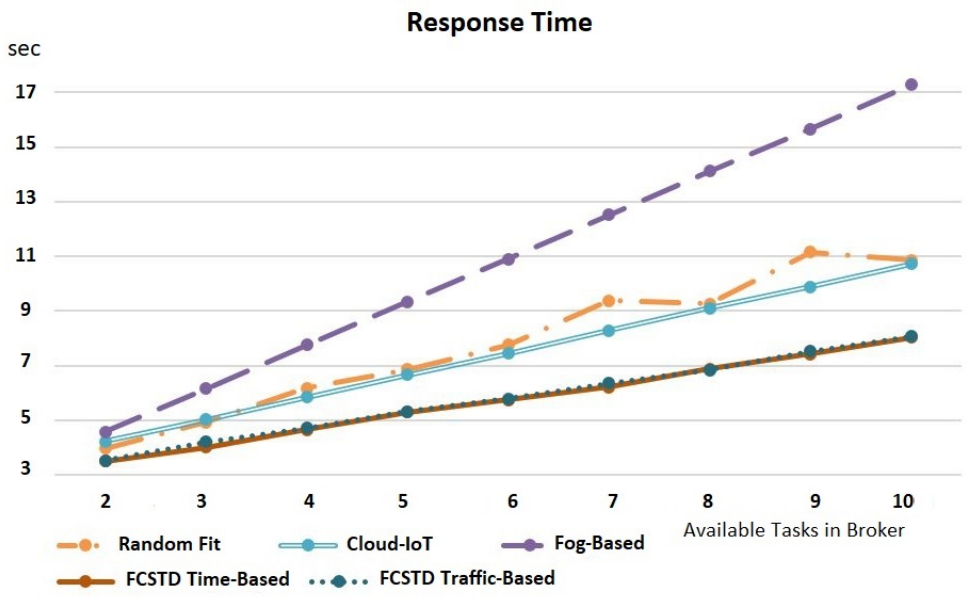


Figure 5.8: The performance of different distribution policies in terms of response time for SSDP tasks, published in [28]

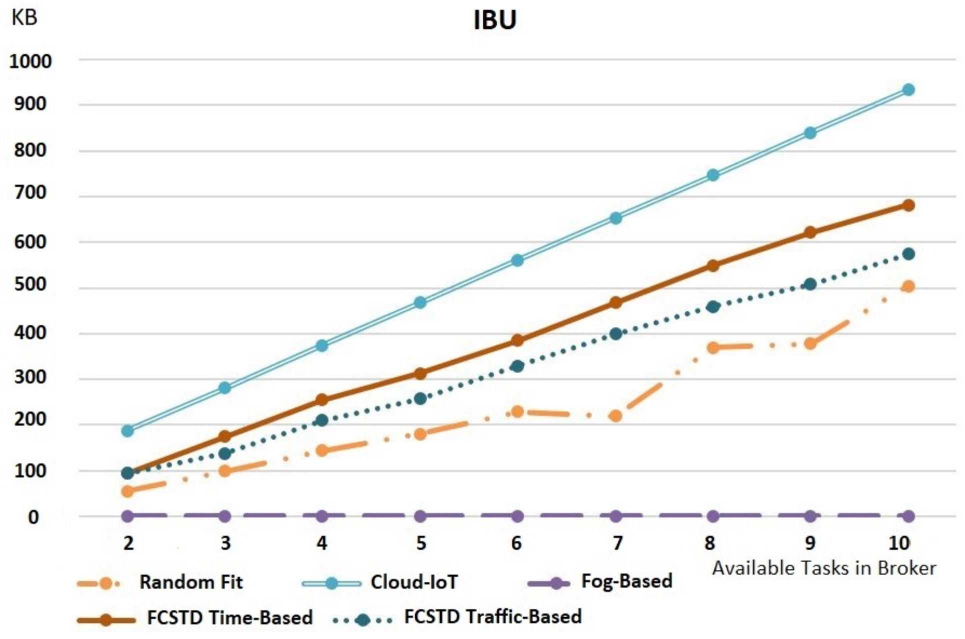


Figure 5.9: The performance of different distribution policies in terms of IBU for SSDP tasks, published in [28]

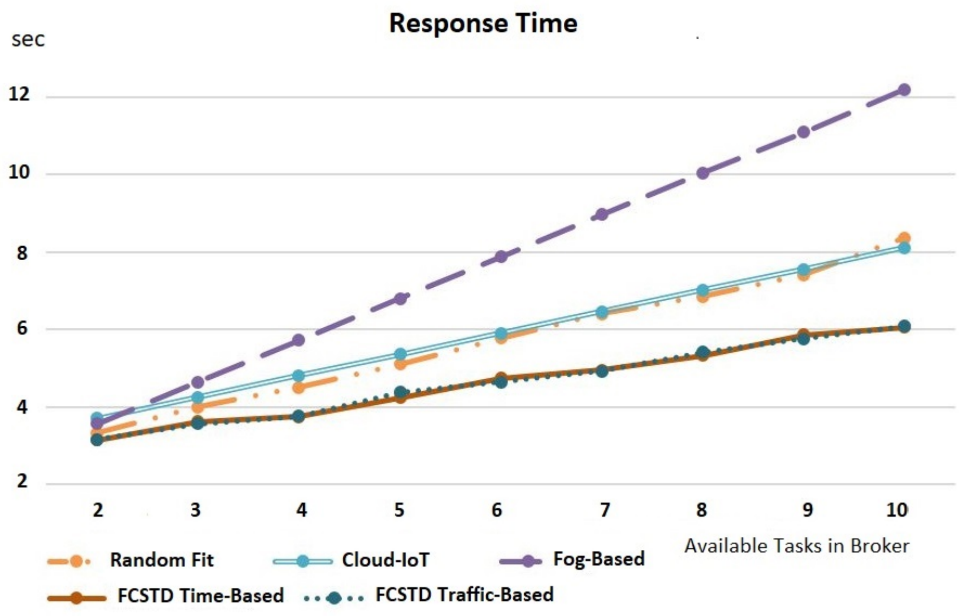


Figure 5.10: The performance of different distribution policies in terms of response time for DSSP tasks, published in [28]

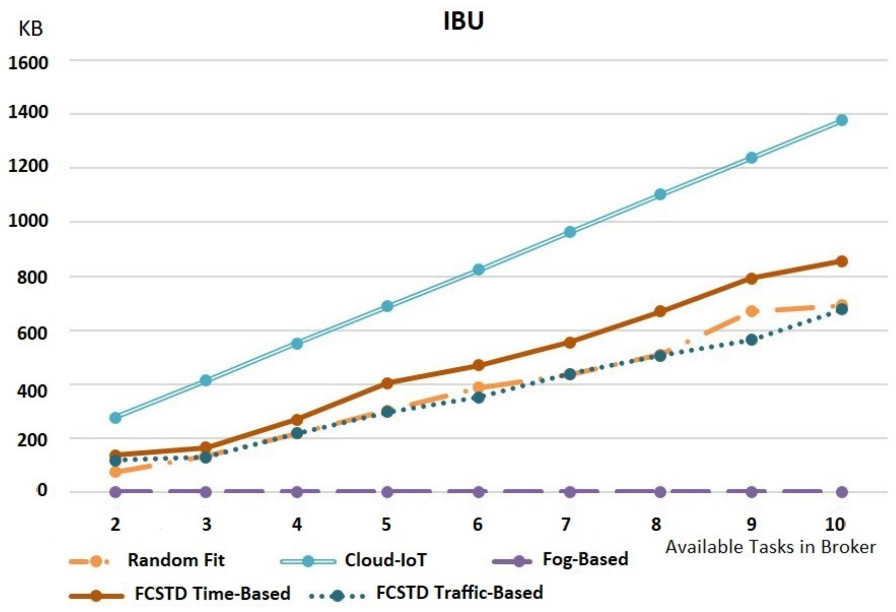


Figure 5.11: The performance of different distribution policies in terms of IBU for DSSP tasks, published in [28]

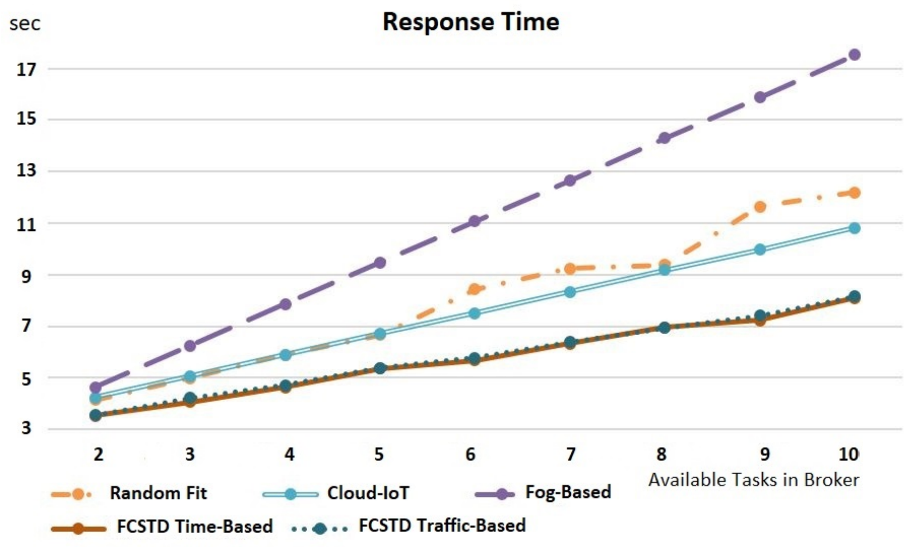


Figure 5.12: The performance of different distribution policies in terms of response time for DSP tasks, published in [28]

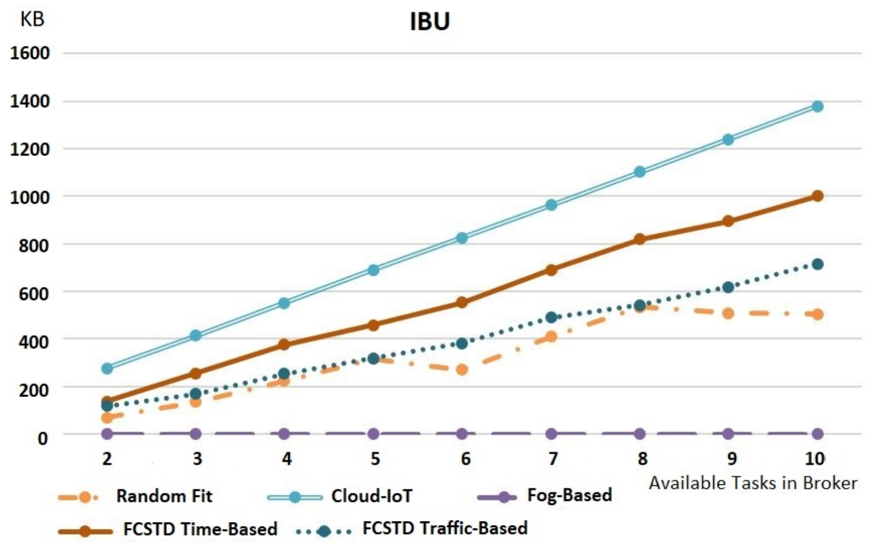


Figure 5.13: The performance of different distribution policies in terms of IBU for DSP tasks, published in [28]

As is depicted in figure 5.12, both versions of our proposed method provide the best response time for distributing the DSP tasks. It can also be seen that the Cloud-IoT method is ranked as the third-best approach in comparison to the other methods in terms of response time. We also observed that the Random Fit approach failed in this situation (and got ranked as the second-worst approach) and only performed better than the fog-based method. In terms of IBU, after the fog-based method, which does not communicate any data over the Internet, Random fit is the best method as it utilized less Internet bandwidth than both versions of FCSTD and Cloud-IoT methods.

Generally, with regard to figures 5.6-13, it can be said that the fog-based method is always the best approach for delay-tolerant applications, as it utilizes no Internet bandwidth and provides high response times. We can also conclude that both versions of FCSTD always provide the fastest response time, which makes them suitable for delay-sensitive applications. However, it must be noticed that the traffic-based version of FCSTD always performed better than its time-based version in terms of IBU, so it can be a better approach for reducing both response time and IBU. Furthermore, we also observed that the Random Fit method, in some cases, performs better than FCSTD in terms of IBU in exchange for a considerably higher response time.

Regarding [28]; as a disadvantage, it must be discussed that our proposed method is only usable for those case studies in which the variation of application inputs has a direct effect on the output, where neural networks can perform excellently. For instance, we used another application that solves a mathematical problem, checking if a number (between 1 and 2^{50}) is prime or not. We trained the neural networks similar to our previous case study, and as the neural networks were not able to predict the response time by checking the inputs of the tasks, our proposed method performed poorly, as is presented in figure 5.14. It must be mentioned that in the following figure, communication delay of fog and cloud servers has not been considered as we only aim to show the wrong prediction of the neural networks when there is no relation between their input and output.

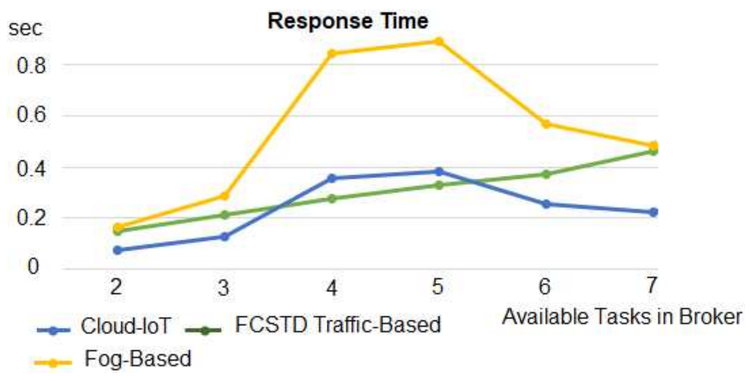


Figure 5.14: Comparing the achieved response times by using, Cloud-IoT, FogBased and FCSTD Traffic-Based methods, published in [28]

5.3 Smart Task Distribution Between Fog and Cloud Servers with Different Workloads

In sections 5.1 and 5.2, we assumed that the available servers in Fog and Cloud layers are always idle and ready to process the generated raw data by the IoT devices. We also assumed that different numbers of tasks could be available in the broker at the time of task distribution. Now we consider the situations in which the available Fog and Cloud servers have different workloads at the time of task arrival (and their workloads change over time). This assumption causes a problem for the broker because the broker is not aware of the workloads of servers, and it might forward a received task from the IoT devices to a server with a high workload, which increases the response time.

In this section, we present the Machine Learning Based Task Distribution (MLTD) algorithm (a new version of FCSTD), which can distribute the received tasks between the servers by considering their workloads.

5.3.1 Network Architecture and Assumptions

As reported in [16]; for evaluating the performance of MLTD, we consider the following network architecture, depicted in figure 5.15. As is presented, the network consists of a user who is equipped with the Wireless Body Sensor Network, which continuously generates healthcare-related data. Moreover, there is also a broker which is responsible for receiving the raw data from the user and assigning them to the fog or cloud servers for processing. Furthermore, the fog layer consists of two different fog servers (accessible in local area network through ad-hoc communications), which are responsible for receiving the tasks from the broker, processing, and sending them back to the user or cloud servers (for further processing and permanent storage of the results). Furthermore, there is also a powerful remote server in the cloud (accessible via a public IP address with a distance of 14 hops to the user), which is accessible through the Internet and responsible for processing the received tasks from the broker or storing the result of the processed tasks by the fog servers. The technical specifications of the fog and cloud servers are presented in Table 5.4.

Table 5.4: The technical details of the used devices in our experiment in [16]

	Fog Server 1	**Fog Server 2**	**Remote Server**
Type of Device	Laptop	Laptop	PC
CPU	Intel Core i5 6200 U	Intel Core i7 7600 U	Intel Core i7 8700 U
RAM	8 GB	8 GB	16 GB

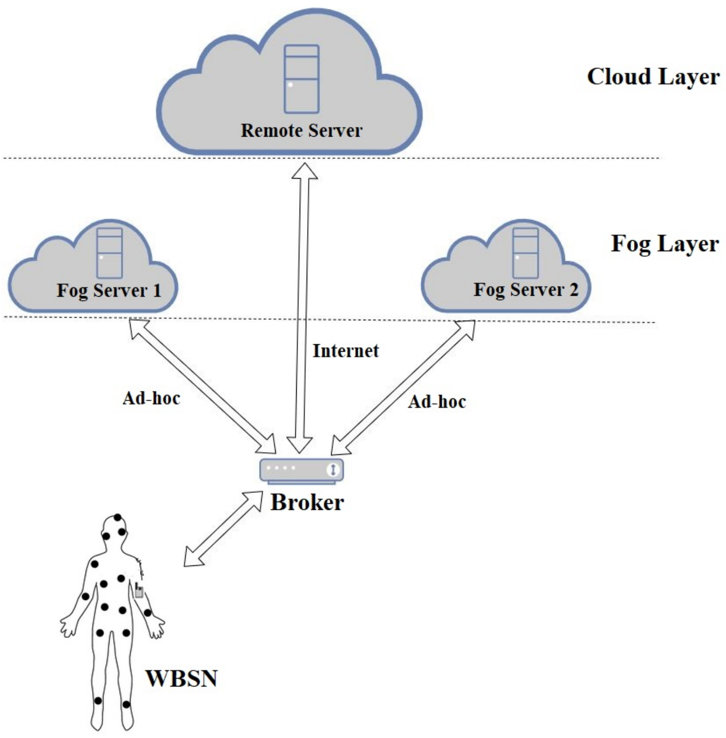


Figure 5.15: Network Architecture

As is presented in [16]; similar to [97] we consider the workloads of servers to be the occupation percentage of CPU and RAM, and we assume that the servers can be averagely, 30%, 50%, and 75% occupied. It is also assumed that the servers inform the broker about their workloads continuously. Table 5.5 shows the effect of server workload on the processing time. It must be mentioned that the presented processing times in table 5.5 are the average processing time of 200 different tasks that we ran on the mentioned servers with different workloads. As is evident, there is a direct relationship between the workload and the processing time, such that higher workloads lead to higher processing times. Similar to the previous experiments, we also assume that there is no power restriction, nodes are homogeneous, data is transmitted reliably, and there is no radio frequency interference.

Table 5.5: The Effect of Workload on the Processing Time of Servers (published in [16])

Workload	Fog Server 1	Fog Server 2	Remote Server
30%	1.51 sec	1.63 sec	0.57 sec
50%	1.64 sec	1.73 sec	0.7 sec
75%	1.77 sec	1.86 sec	0.88 sec

5.3.2 Problem Description

As reported in my publication [16]; in a combined fog-cloud scenario, there are fog and cloud servers which are heterogeneous in terms of workload, computing capabilities, and latency to the user, and all these three factors affect the response time directly. For instance, the cloud servers are more powerful than the fog servers in terms of computational capabilities, which causes the processing time of the cloud servers to be less than the fog servers. However, as remote servers in the cloud are far from the users and only accessible through the Internet, they have more latency to the users in comparison to Fog servers. Moreover, the different servers in both Fog and Cloud layers might have different workloads, which leads to different processing times and consequently, different response times. Considering these heterogeneous servers in both layers, when the data provider (an IoT device) sends a task to the distributor unit (broker) to be sent to one of the servers for processing, the distributor unit can not detect the fastest server at that time slice. Consequently, it might forward the received task to a server with a higher delay or workload, which leads to increased response times (that is not tolerable by delay-sensitive applications) or IBU.

In order to deal with the problem of task distribution in heterogeneous fog-cloud environments, we upgraded the FCSTD algorithm to be able to distribute the tasks by considering the different workloads of servers. The upgraded FCSTD has been called Machine Learning Based Task Distribution (MLTD) method in [16].

5.3.3 Machine Learning Based Task Distribution (MLTD)

As is presented in [16]; in order to train the neural networks, we generated 1000 different tasks and ran 800 tasks on all three servers with the three different workloads (so we ran the tasks for nine times) and noted the provided response times. Consequently, we know the inputs (the 800 tasks) and the results (the processing times) in all of the nine situations. In the next step, we trained one neural network for each case (so the total number of neural networks is equal to 9), in which the input is the received task from the user, and the output is the response time of the server (as is shown in figure 5.16). We also checked the accuracy of the neural networks with the other 200 tasks, and the results show that the neural networks can predict the response time with the error of $\pm$ 0.1 seconds. In MLTD technique, all the servers must continuously inform the broker about their workloads so the broker is aware of the workloads of servers at the time of task distribution. Therefore, whenever a user sends a task to the broker, the broker checks the workloads of servers and chooses three neural networks. It sets the received task as the input, and the neural networks predict the response time. In the next step, with regard to the expected response times, the broker assigns the task to the server, which can provide the least possible response time at that time slice.

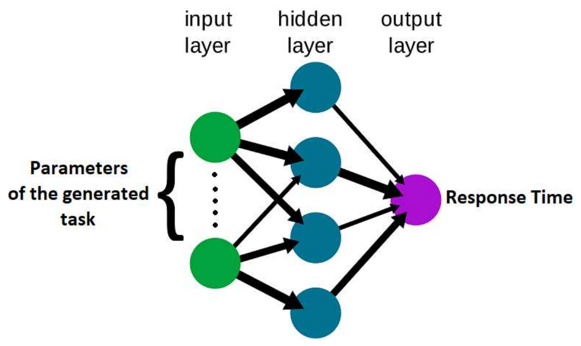


Figure 5.16: The training process of Neural Networks

5.3.4 Evaluation of Results

Regarding [16]; we used the MLTD technique for distributing the generated tasks (200 tasks) of the discussed healthcare application. We compared the achieved results with four competitors in terms of response time, IBU, and resource utilization. The implementation details of the experiment are presented in Table III, and the competitor methods (chosen from the state of the art) are described as follows:

- Random Fog: which is one of the discussed methods in [32]. In this method, the broker selects one of the fog servers randomly and assigns the received task to this server for processing.

- Virtual Fog Resolver: the proposed method in [11] which we discussed about it in chapter 3. This is the most similar method to MLTD as it has been designed for task distribution between different fog and cloud servers with different workloads.

- Random Fit: which is one of the proposed approaches in [33] that randomly selects one of the fog or cloud servers for task assignment purposes.

- Cloud-IoT: which is the proposed approach in [31] that exploits the cloud resources for processing the produced raw data by the IoT devices.

As reported in [16]; in order to evaluate the performance of MLTD and compare its results with the other four competitors, we used the shown network architecture in figure 5.15. In the following, we investigate the performance of all of the discussed methods in terms of response time, IBU, and resource utilization in different experimental conditions, in which there are:

- Different Fog servers with Different Workloads (DFDW)

- Different Fog servers with Similar Workloads (DFSW)
- Similar Fog servers with Different Workloads (SFDW)
- Similar Fog servers with Similar Workloads (SFSW)

Figures 5.17-28 show the performance of the discussed task placement methods in different conditions in terms of response time, IBU, and resource utilization. It is worth noticing that we will not discuss about the performance of Cloud-IoT and Random Fog methods in terms of IBU because Cloud-IoT is always the worst method in terms of IBU as it communicates all the requests and responses over the Internet. It is also clear that the Random Fog method is always the best in terms of IBU as it distributes the tasks between the fog servers, which are accessible in the local area network (in our experiments).

As is discussed in [16]; in figures 5.17, 5.18, 5.20, 5.21, 5.23, 5.24, 5.26, and 5.27, the Y-axis represents the time (in seconds), while the X-axis shows the number of Time Slices that the Workload Changes (TSWC). In this experiment, the broker receives one task at each time slice, so when TSWC is equal to 1, it shows that the workloads of servers are being changed at each time slice (randomly between 30%, 50%, and 75%). Therefore, during the distribution process of 200 tasks, the workloads of the servers change 200 times. Similarly, when the TSWC is equal to 2, it means that the workloads of servers change after each two time slices (100 times during the distribution process of 200 tasks) and so on.

DFDW is the most similar condition to reality. There are different fog servers in this condition, which their workloads change over time (in our experiment, it changes randomly between 30%, 50%, and 75%). As shown in figure 5.17, in DFDW condition, the MLTD method always provides the best response time. The reason is that this method is intelligent and can detect the fastest server at the time of task arrival. The diagram also reveals that the Random Fog method always performs worse than MLTD and better than all other task distribution techniques. Moreover, it is apparently seen that the Random Fit method performs better than VFR in terms of response time in DFDW condition because, in more than 90% of situations, it provided a reduced response time compared to the VFR method. It is also clear from the data that the Cloud-IoT method is always the worst approach for processing the generated tasks of the IoT-based applications as it performs worse than all the other techniques in DFDW condition. The reason is that this method assigns all the tasks to the remote servers, which are far from the users and have high latency.

In terms of IBU, as is illustrated in figure 5.18, we see that the MLTD method is the best method for reducing the IBU. We mentioned that MLTD performs intelligently and always selects the fastest server. As in DFDW condition, the fastest server in most of the situation is in the fog layer, so the MLTD assigns most of the tasks to the fog layer and reduces the IBU. As is shown in figure 5.19, the MLTD method has assigned almost 185 tasks to the fog layer and only 15 tasks to the remote server.

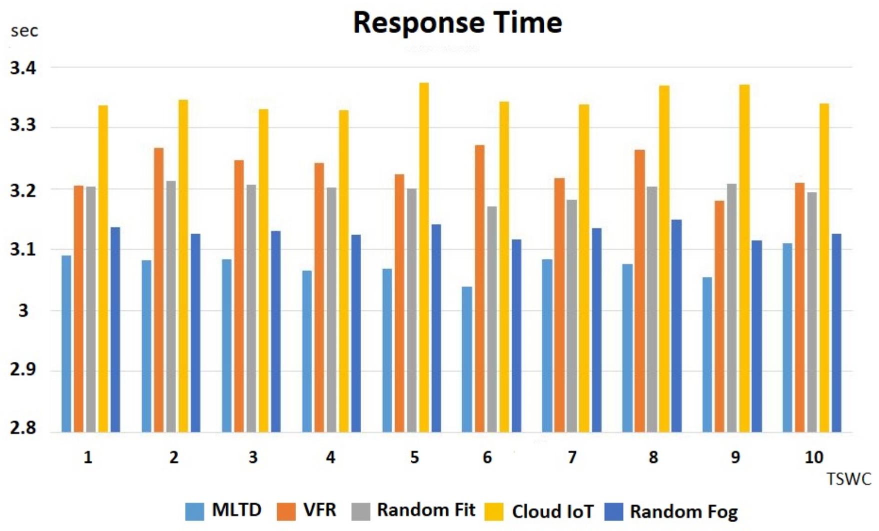


Figure 5.17: Evaluation of different methods in terms of response time in DFDW condition, published in [16]

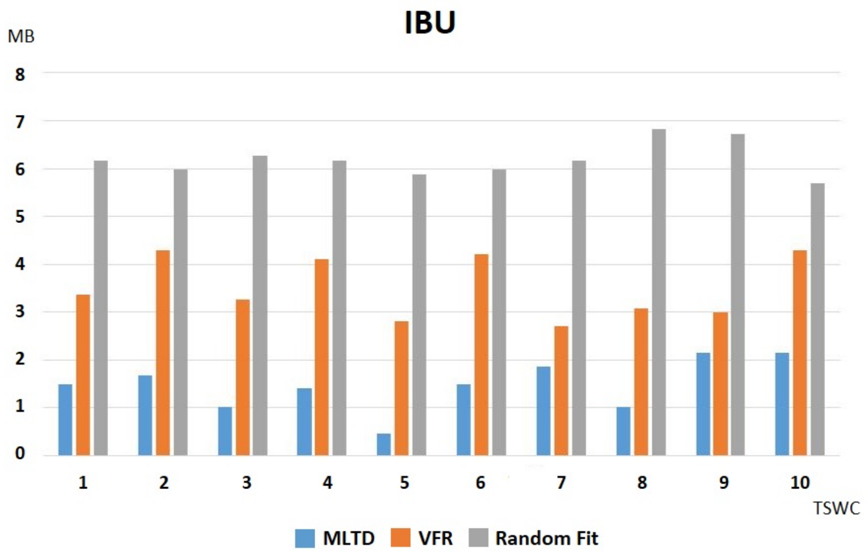


Figure 5.18: Evaluation of different methods in terms of IBU in DFDW condition, published in [16]

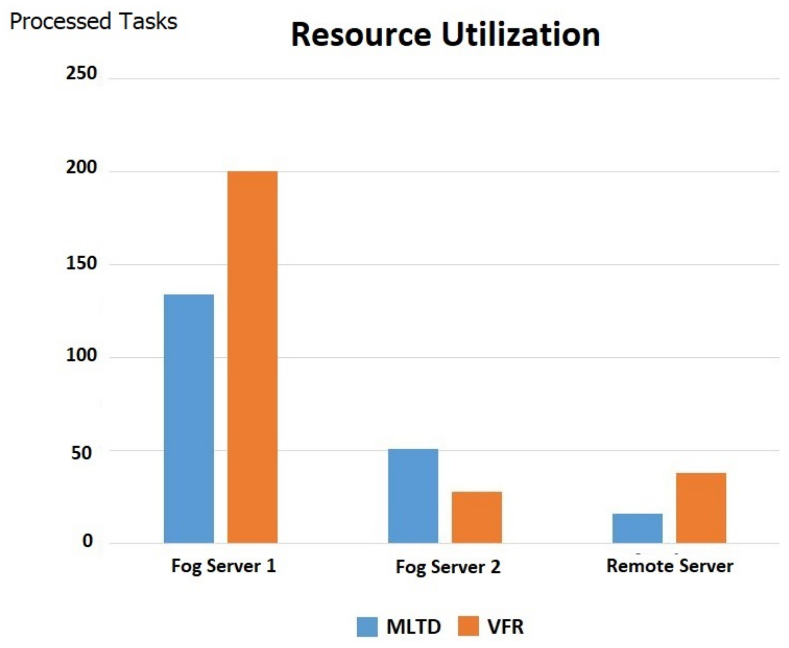


Figure 5.19: Evaluation of different methods in terms of resource utilization in DFDW condition, published in [16]

In SFDW condition, we assumed that there are similar fog servers (in terms of computational power) with different workloads. As is depicted in figure 5.20, in terms of response time, the MLTD method in almost 90% of the situations is the fastest task distribution method, and Random Fog is the second-best method, which performs very similar to the MLTD. Moreover, it could be noticed that in SFDW condition, the VFR and Random Fit methods perform similarly such that in 60% of situations, Random Fit performs better than VFR, and in 40% of the situations, VFR provides a better response time in comparison with Random Fit [16]. It is also clear that the Cloud-IoT method is again the worst approach in terms of IBU.

In terms of IBU, the performance of MLTD, VFR, and Random Fit are similar to their performance in DFDW condition. Figure 5.21 shows that MLTD communicates less data over the Internet than the VFR and Random Fit methods. As is depicted in figure 5.22, MLTD sends 16 tasks to the remote server, and VFR assigns more than 40 tasks to the cloud layer. However, VFR always performs better than Random Fit in terms of IBU.

So it can be generally said that in SFDW condition, MLTD performs better than all of the competitors in terms of response time and better than VFR and Random Fit in terms of IBU. It can also be said that VFR and Random Fit perform similar to each other in terms of response time, but VFR utilizes less Internet bandwidth in comparison to Random Fit, so generally, in this condition, VFR can be ranked as the second-best task placement technique.

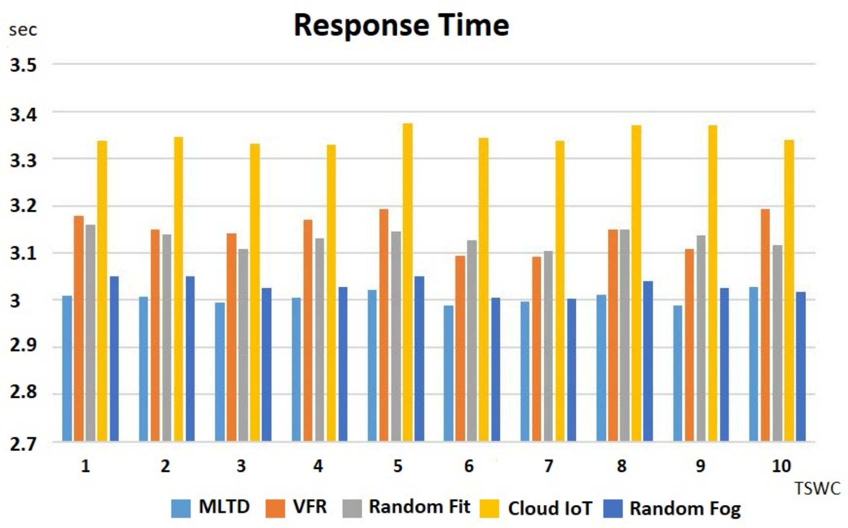


Figure 5.20: Evaluation of different methods in terms of response time in SFDW condition, published in [16]

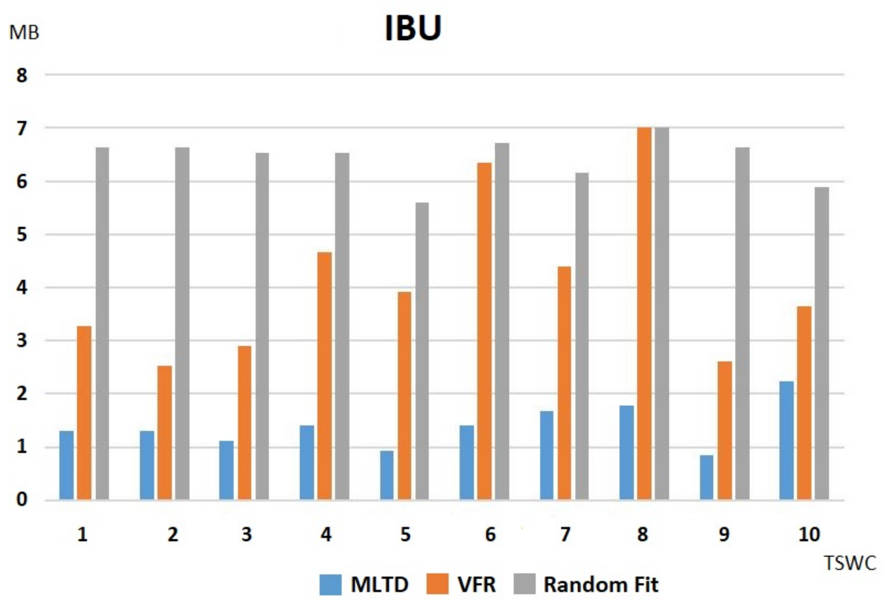


Figure 5.21: Evaluation of different methods in terms of IBU in SFDW condition, published in [16]

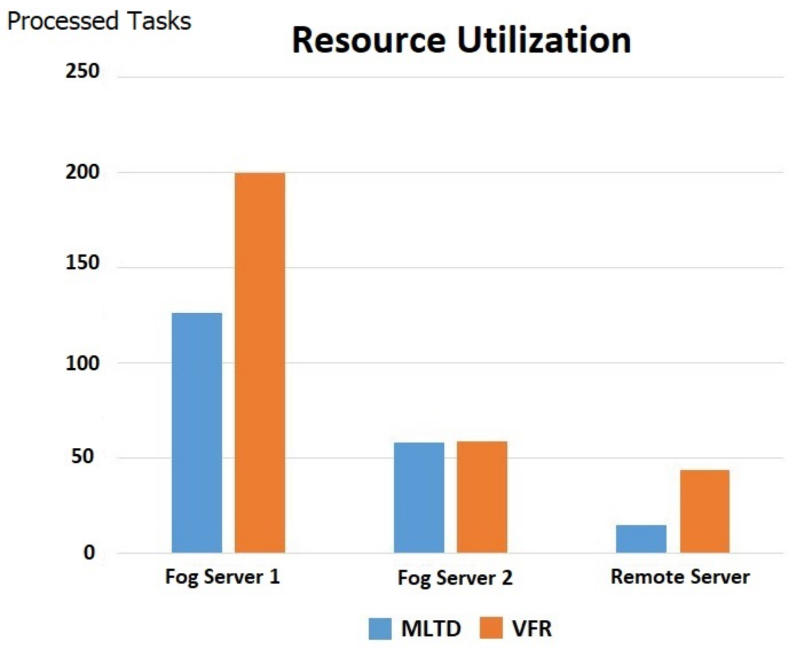


Figure 5.22: Evaluation of different methods in terms of resource utilization in SFDW condition, published in [16]

In DFSW condition, it is assumed that there are different fog servers in the fog layer, which are 75% occupied. As is illustrated in figure 5.23, in this condition, MLTD fails in terms of response time and gets rankled as the second-worst approach. The reason is that when fog servers are 75% occupied, they provide a response time, which is similar to the response time of the cloud server, and as the neural networks have an error of 0.1 seconds, the broker distributes the tasks unsuitably, which leads to increased response times [16]. Turning to the details, figure 5.23 also shows that Random Fog is the best method in terms of response time in DFSW condition [16].

It is also clear that VFR performs better than Random Fit by providing a response time of less than 3.1 seconds in all of the situations. In this situation, the third-best approach is Random Fit, which performs better than MLTD and Cloud-IoT and worse than VFR and Random Fog. It must also be mentioned that similar to previous conditions, the Cloud-IoT method is the worst approach for task distribution in DFSW condition.

As is shown in figure 5.24, we see that only in 30% of the situations, MLTD performs better than VFR in terms of IBU, and in 70% of the situations, VFR utilizes less Internet bandwidth in comparison to MLTD. Moreover, we also observed that Random Fit is the worst technique for task distribution in SFDW condition in terms of IBU. Figure 5.25 also shows that the MLTD has assigned more tasks to the remote servers than the VFR method, which caused the IBU of MLTD to be higher than VFR.

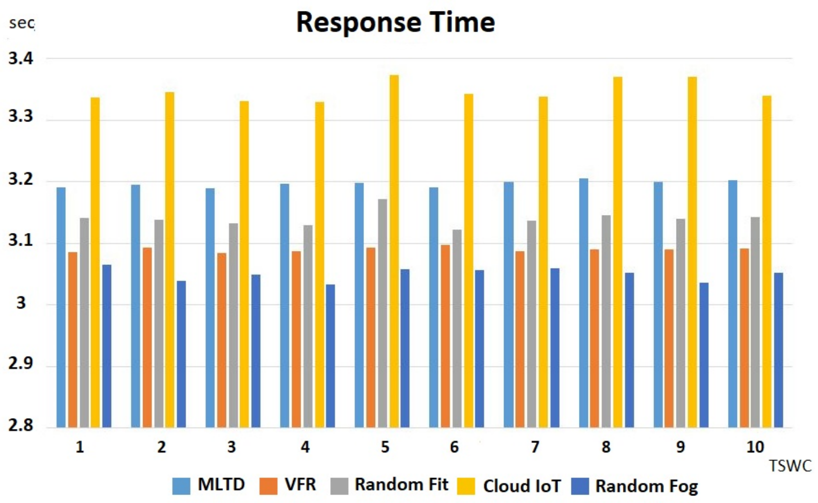


Figure 5.23: Evaluation of different methods in terms of response time in DFSW condition, published in [16]

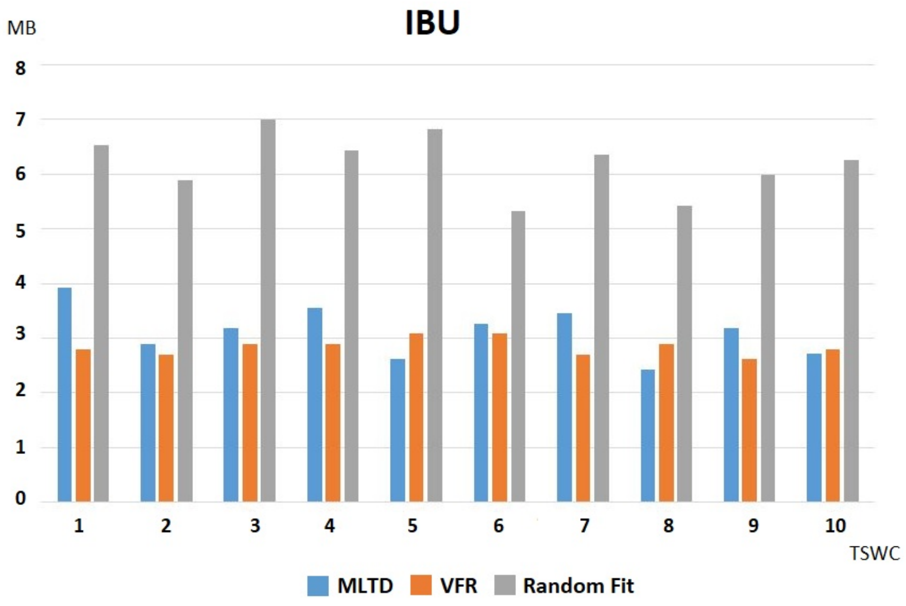


Figure 5.24: Evaluation of different methods in terms of IBU in DFSW condition, published in [16]

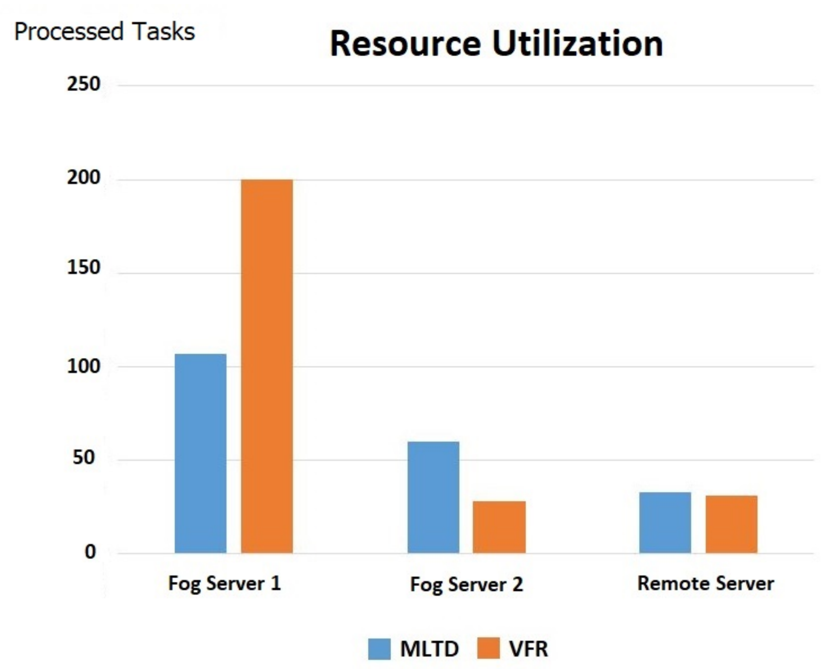


Figure 5.25: Evaluation of different methods in terms of resource utilization in DFSW condition, published in [16]

In the SFSW condition, it is assumed that the fog servers are similar in terms of computational power, and they are 75% occupied. Figure 5.26 shows that the MLTD method (similar to DFSW condition) fails in terms of response time compared to the other techniques and gets ranked as the second-worst approach after Cloud-IoT, which is the worst method. Again, the error of neural networks is the reason of wrong task distributions and high response times. It is also clear that in this condition, the Random Fog, VFR, and Random fit are the best, second-best, and third-best methods in terms of response time, respectively [16].

In terms of IBU, as is depicted in figure 5.27, we see that our proposed method is the best and performs better than VFR and Random Fit. Random Fit and VFR got ranked as the second and third methods in this condition. Figure 5.28 shows that the VFR method has assigned more tasks to the remote server in comparison with MLTD, which causes the IBU of VFR to be higher than MLTD. It must be mentioned that sending more tasks to the remote server does not necessarily lead to higher response times. As shown in Figures 5.28 and 5.26, we see that VFR assigns more tasks to the remote server compared to MLTD, but it reduces the response time. The reason is that in the VFR method, the task might be assigned to the remote server, but the user can receive its response from the fog servers, which have less latency.

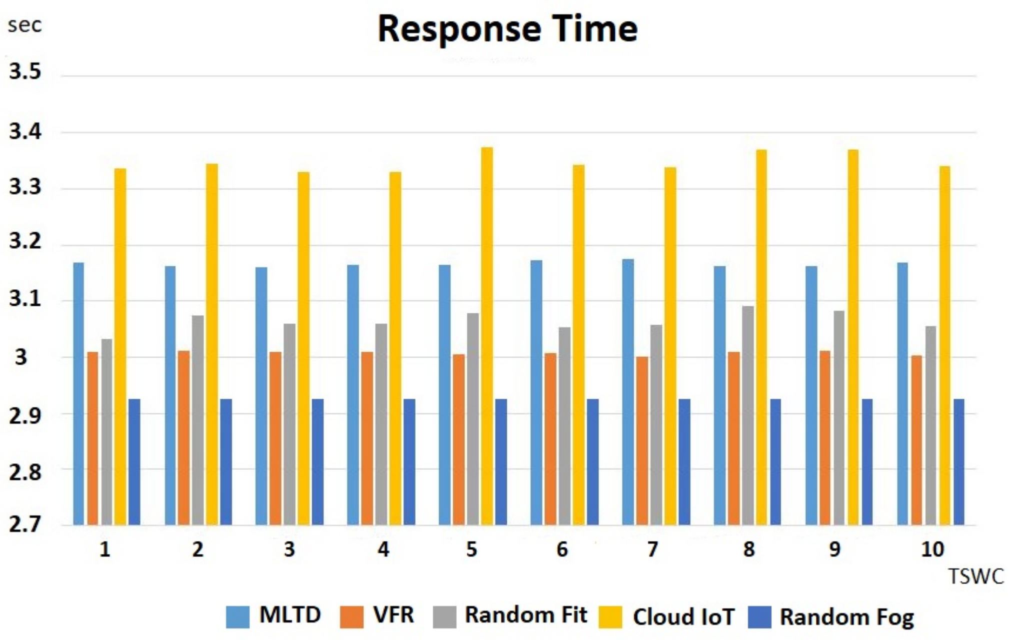


Figure 5.26: Evaluation of different methods in terms of response time in SFSW condition, published in [16]

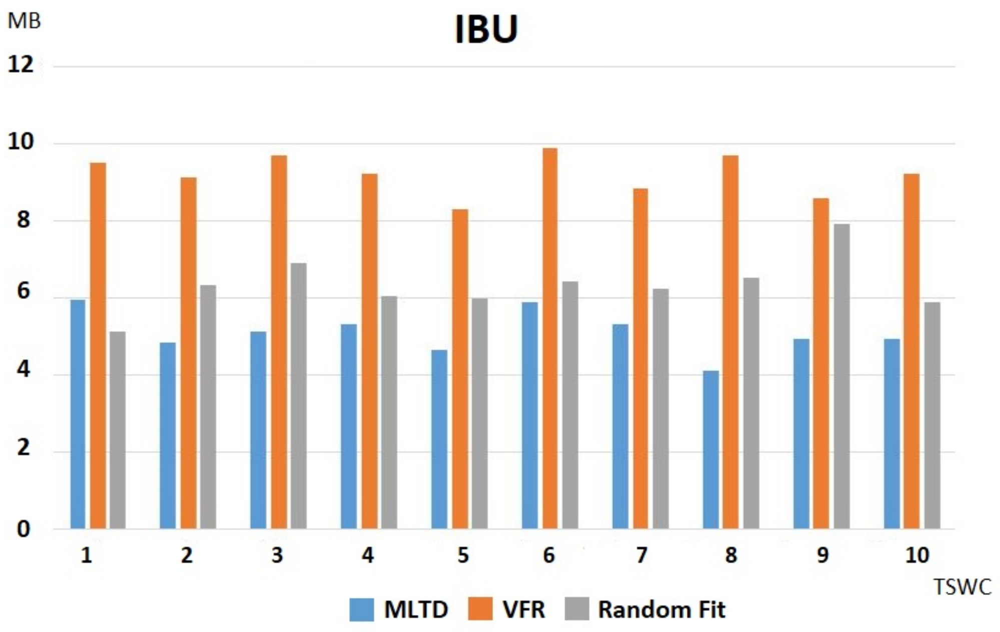


Figure 5.27: Evaluation of different methods in terms of IBU in SFSW condition, published in [16]

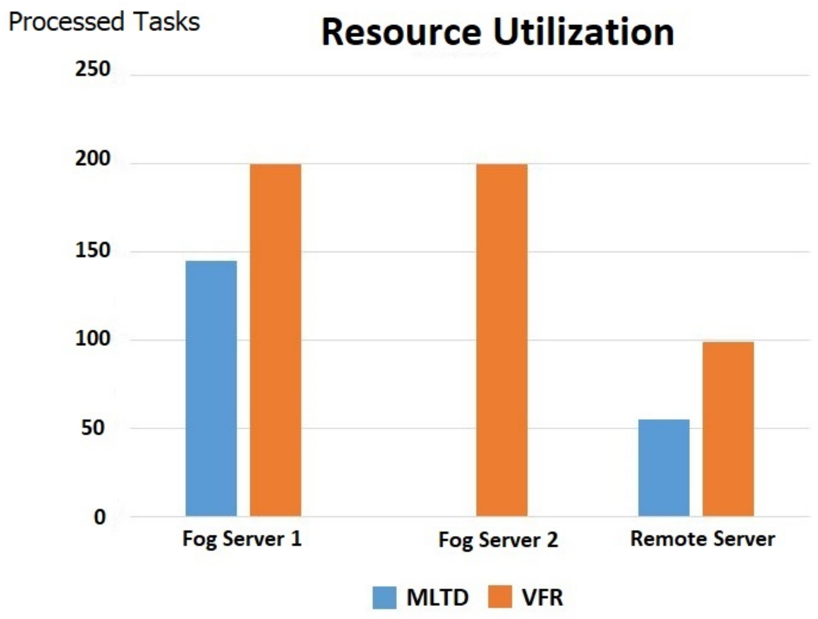


Figure 5.28: Evaluation of different methods in terms of resource utilization in SFSW condition, published in [16]

5.4 Investigating the Effective Parameters on the Performance of MLTD

In the previous section, we discussed about the final version of our proposed smart task distribution method (MLTD) and observed its performance in different experimental environments. We were witnessing that MLTD was better than other competitors in terms of reducing the response time and IBU when the neural networks were able to perform excellently for predicting the response times of servers.

5.4.1 Research Questions and Hypothesis

MLTD works based on predictions of the ANNs. The performance of ANNs depends on several factors, such as the utilized training method and the richness of training. In order to investigate the performance of MLTD intensely, in this section, we use different training algorithms and exploit different numbers of tasks for the training process of ANNs. In this step, our hypothesis is that utilization of different training methods leads to different results, and the richness of training directly affects the precision of predictions such that when more tasks are used for the training, better results can be achieved. As is presented in [1], the research questions that will be answered in this section are as following:

Table 5.6: The Accuracy of the ANNs, published in [1]

Training Algorithm	Number of Used Tasks for Training	Error
LM	200	±0.31 sec
LM	800	±0.1 sec
GA	200	±0.07 sec
GA	800	±0.05 sec

1- What can improve or deteriorate the performance of our proposed smart task assignment algorithm?

2- Does the utilization of more tasks for the training process of ANNs necessarily lead to better predictions? If yes, how much can it be improved?

3- Does the utilization of different training methods (for the ANNs) lead to different results?

5.4.2 Changing the Method and Richness of Training

In previous sections, we utilized the MLTD algorithm, which was trained by the Levenberg-Marquardt algorithm. We also used 800 different tasks for the training process of MLTD [1]. To see the impact of different training methods on the performance of MLTD, in [1], besides the Levenburg-Marquart (LM), we used Genetic Algorithm (GA) for the training process of ANNs, and we tried to train the ANNs by using 800 and 200 different tasks [1]. The accuracy of the ANNs is presented in Table 5.6.

5.4.3 Evaluation of Results

We used the network architecture that is shown in figure 5.15. The technical details of the experiment and the used devices are also similar to the previous experiment (section 5.3.1). Regarding [1]; in the next step, we set up the following task distribution methods in the broker:

- MLTD LM 800: The MLTD method in which the ANNs are trained with the Levenberg-Marquardt algorithm and the number of used tasks for the training process is equal to 800.
- MLTD LM 200: The MLTD method in which the ANNs are trained with the Levenberg-Marquardt algorithm and the number of used tasks for the training process is equal to 200.

- MLTD GA 800: The MLTD method in which the ANNs are trained with the Genetic algorithm and the number of used tasks for the training process is equal to 800.
- MLTD GA 200: The MLTD method in which the ANNs are trained with the Genetic algorithm and the number of used tasks for the training process is equal to 200.
- VFR: The proposed method in [11], which uses the best fog servers for task processing and also uses the cloud server in case of failure of the best fog server.
- Random Fit: The proposed method in [33] that randomly assigns the tasks to one of the fog or cloud servers.
- Better Workload: The proposed method in [90] which assigns the tasks to the server with less workload.
- Ideal MLTD: In this method, it is assumed that the MLTD can predict the response times of servers without any error. Although this method is unreal and impossible to implement, but we use it to show the difference between the prediction quality of our method and the ideal case.

As is presented in [1]; to compare the performance of the mentioned methods in different situations, we provided the following four conditions in which there are:

- Different Fog servers with Different Workloads that are changing randomly between 30%, 50%, and 75% (DFDW)
- Different Fog servers with Similar Workloads (75%) (DFSW)
- Similar Fog servers with Different Workloads that are changing randomly between 30%, 50%, and 75% (SFDW)
- Similar Fog servers with Similar Workloads (75%) (SFSW)

It must be mentioned that the presented results in this section are based on the average results of 20 experiments for processing of 200 different tasks [1].

5.4.3.1 DFDW and SFDW Conditions

As reported in [1]; figures 5.29-32 illustrate the performance of the mentioned methods in terms of response time and IBU. Moreover, figures 5.33-34 show the number of tasks that are served by each server in DFDW and SFDW conditions. The results show that MLTD methods (except MLTD LM 200) are the best methods in both conditions in terms of response time and IBU. As is shown, MLTD GA 800 is the best one among all of the MLTD methods, and the reason is that in this method, the ANN prediction accuracy is better than the other methods. It is also obviously seen that MLTD LM 200 is the worst method in both conditions. The reason is that this method predicts the response times badly (with an error of ±0.31 sec with regard to Table 5.6), which leads to the failure of the ANNs and wrong distributions (as can be seen in figures 5.33-34, the MLTD LM 200 has assigned 121 and 146 tasks to the remote server in DFDW and SFDW conditions, respectively). Evaluation of other methods also show that the Better Workload method performs better than VFR and Random Fit approaches in both conditions in terms of response time. It must be mentioned that in figures 5.33-34 and 5.39-40, the Random Fit method is eliminated because of its random policy for distribution. In this experiment, as there are two Fog servers and one Cloud server, the possibility of task assignment to the fog and cloud servers in the Random Fit method is almost 66% and 33%, respectively. Moreover, figures 5.33-34 show that the number of assigned tasks to the fog server 1 is always equal to 200 when VFR is the distribution algorithm. The reason is that the VFR algorithm always sends the tasks to the best fog server, and in case of failure of the best fog server, it resends the tasks to the fog server 2 or the remote server in Cloud.

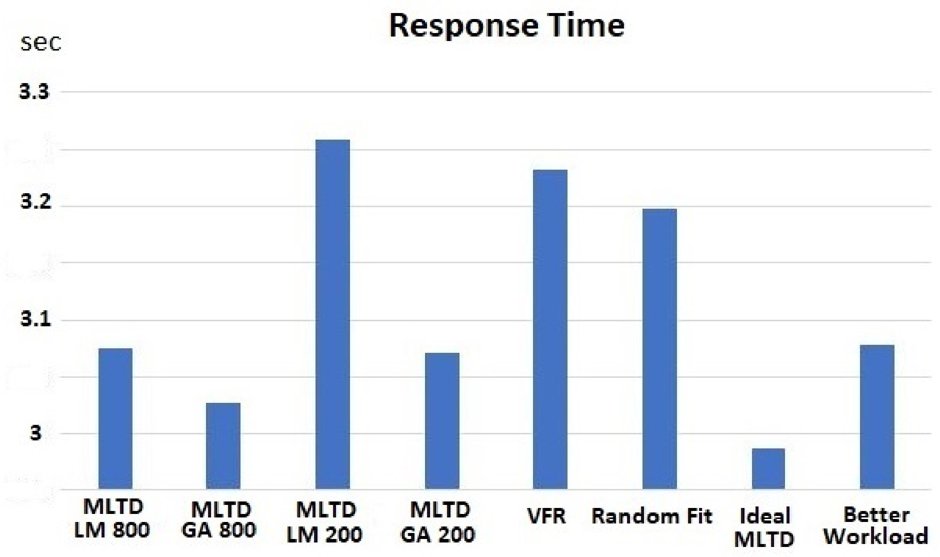


Figure 5.29: Evaluation of different methods in terms of response time in DFDW condition, published in [1]

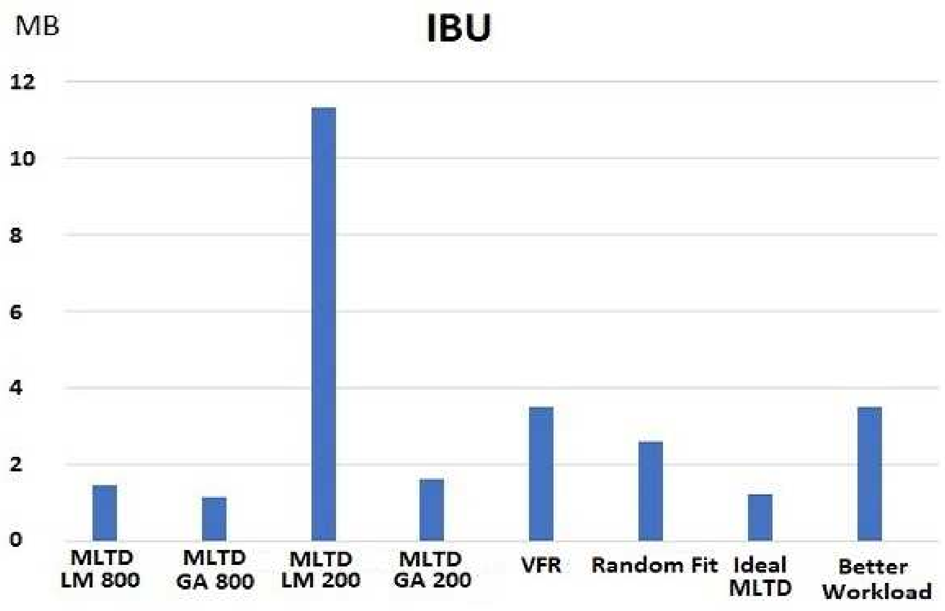


Figure 5.30: Evaluation of different methods in terms of IBU in DFDW condition, published in [1]

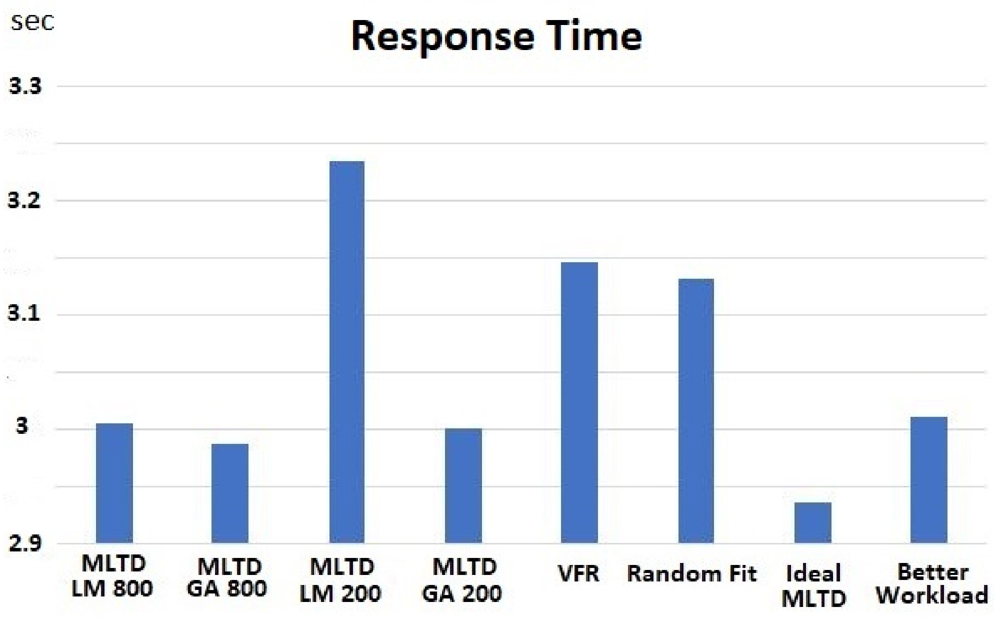


Figure 5.31: Evaluation of different methods in terms of response time in SFDW condition, published in [1]

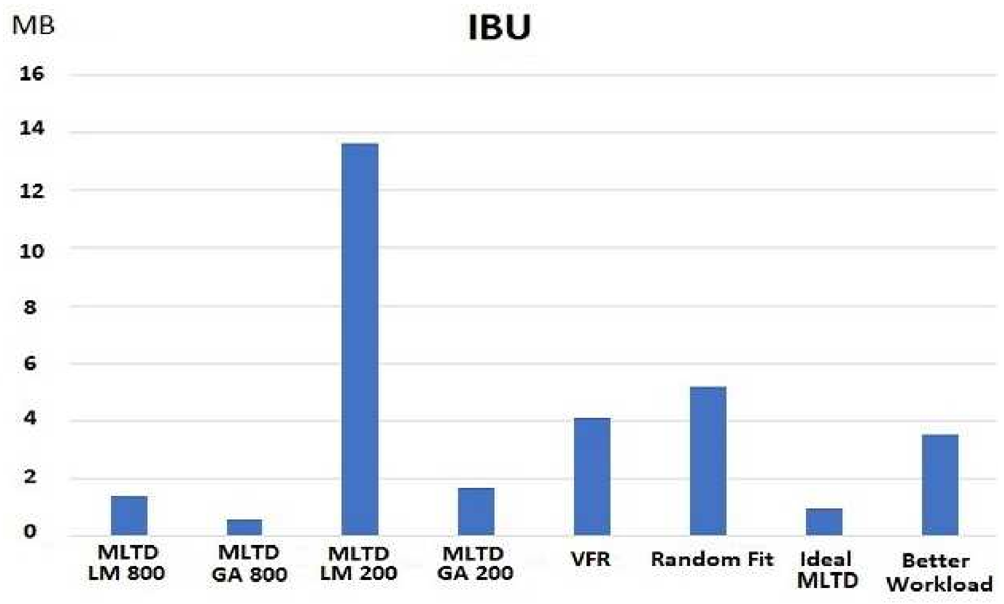


Figure 5.32: Evaluation of different methods in terms of IBU in SFDW condition, published in [1]

5.4.3.2 DFSW and SFSW Conditions

As presented in [1]; figures 5.35-5.38 show the performance of the discussed methods in terms of response time and IBU and also figures 5.39-40 indicate the number of tasks that are processed by the fog and cloud servers in each situation. It must be mentioned that the Better Workload method is eliminated in these two conditions because the workloads of servers are constant. As is depicted in the figures 5.35 and 5.37, the MLTD methods fail in DFSW and SFSW conditions in terms of response time, as the response time of the fog and cloud servers are too close to each other (because the fog servers are all 75% occupied). Among the intelligent methods, MLTD GA 800 and MLTD LM 200 are the best and the worst methods, which could have been expected because of their accuracy in the prediction of the response times. It is also clearly observed that the best method in terms of response time in these two conditions is the VFR method, which provides a response time of less than 3.1 seconds. In terms of IBU, the VFR method is also the best in DFSW condition by assignment of 31 tasks to the remote server and communicating almost 3 megabytes of data over the Internet. Moreover, as is indicated in figure 5.38, we see that the intelligent methods are again the best approaches for task distribution (for reducing the IBU) and the MLTD methods that are trained with 800 tasks communicate the least possible amount of data through the Internet.

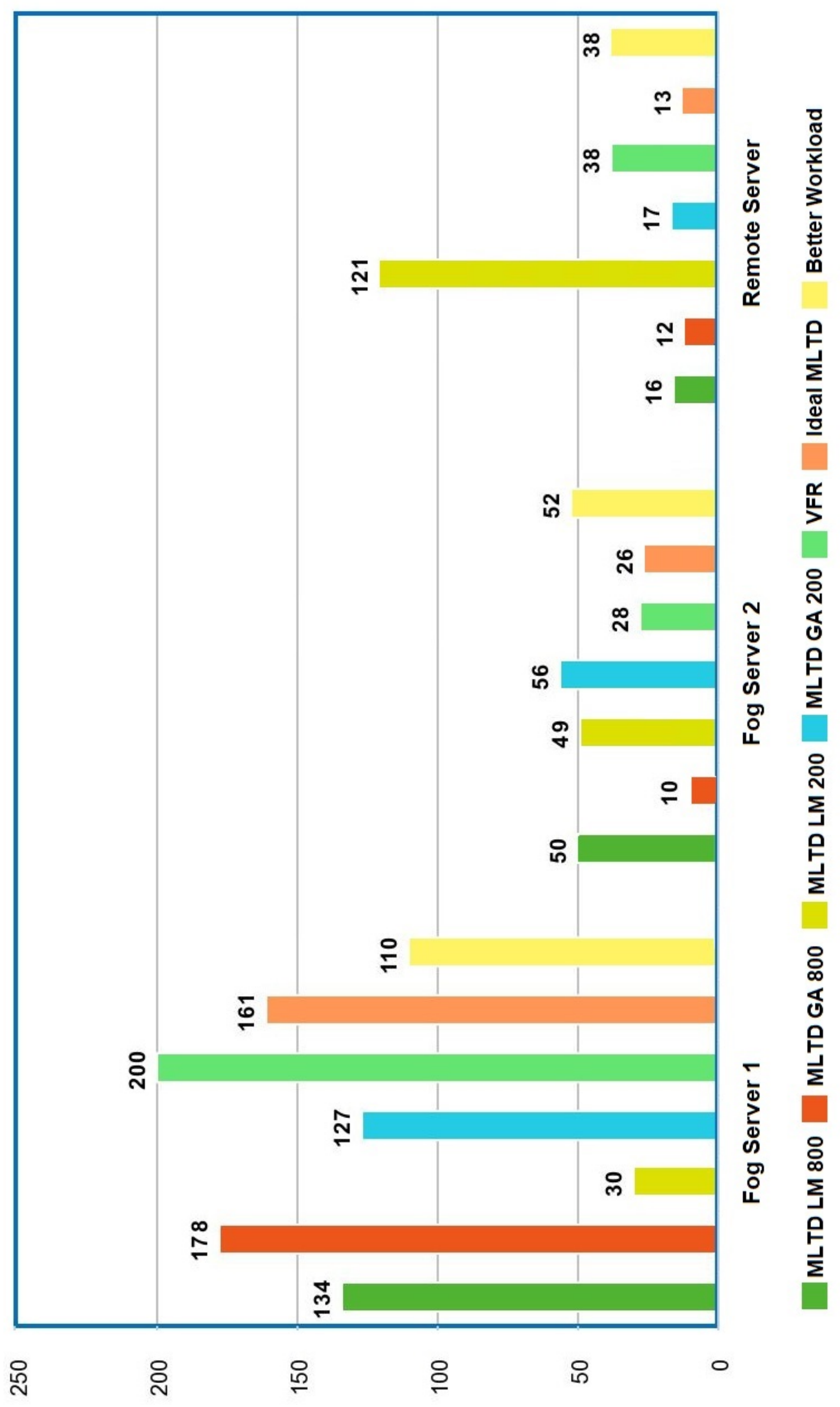


Figure 5.33: The number of processed tasks by each server in DFDW condition, published in [1]

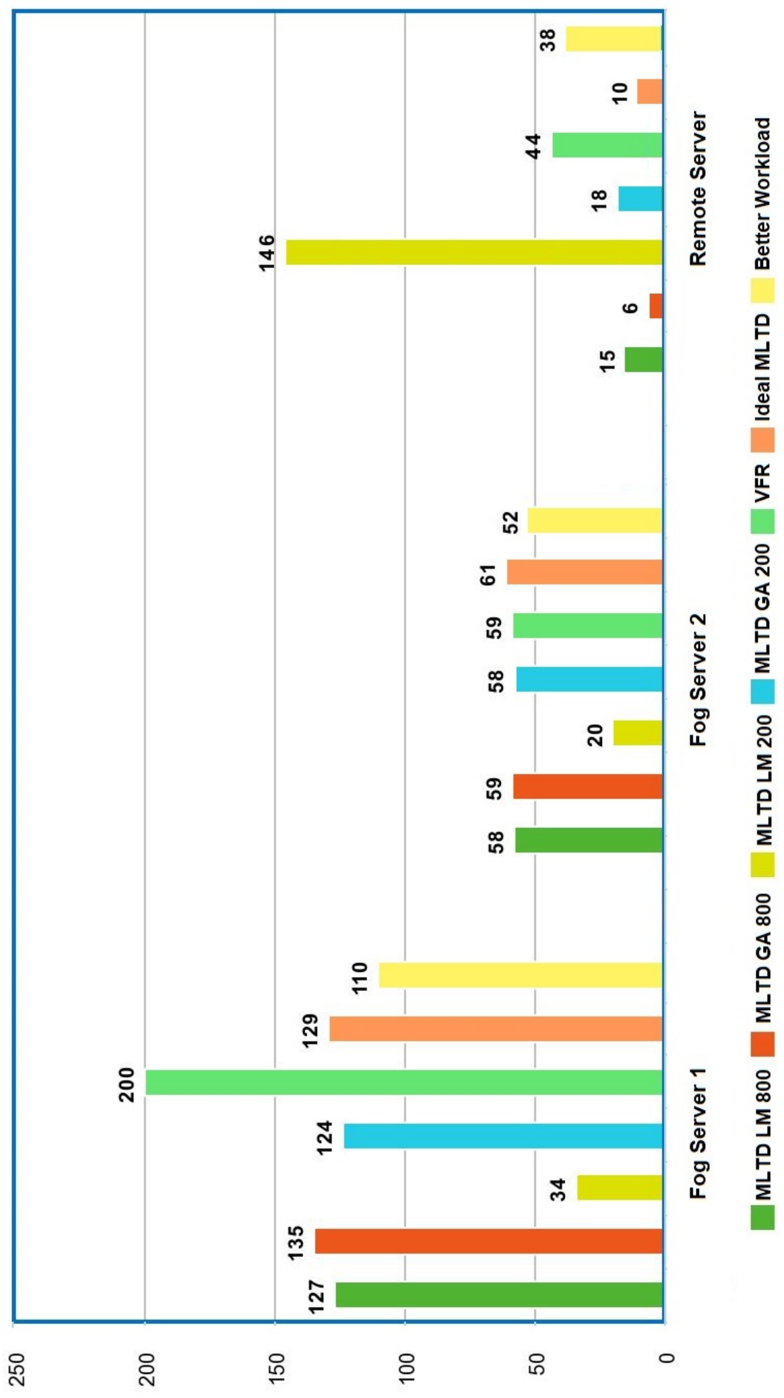


Figure 5.34: The number of processed tasks by each server in SFDW condition, published in [1]

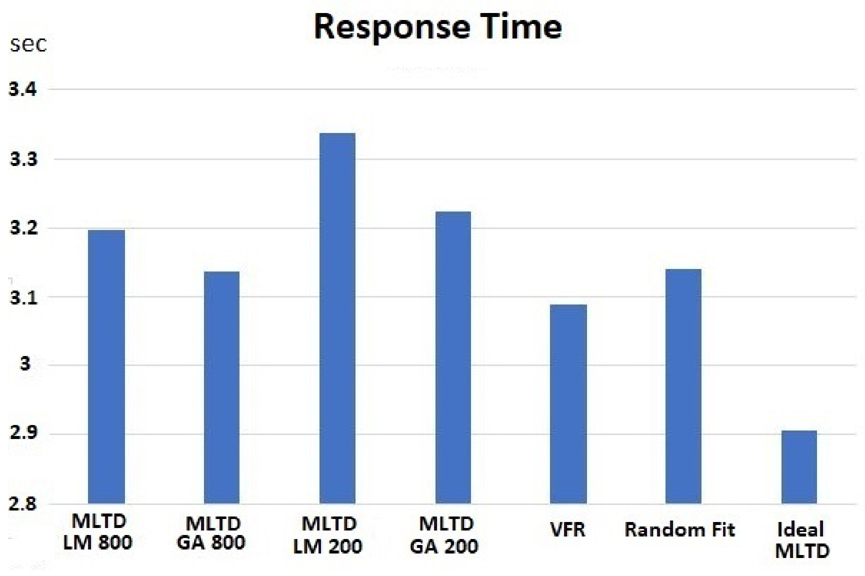


Figure 5.35: Evaluation of different methods in terms of response time in DFSW condition, published in [1]

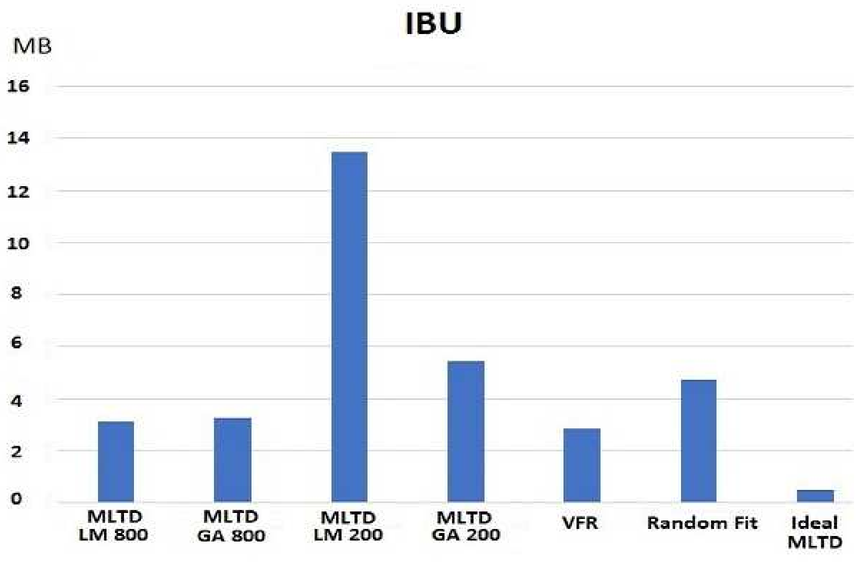


Figure 5.36: Evaluation of different methods in terms of IBU in DFSW condition,published in [1]

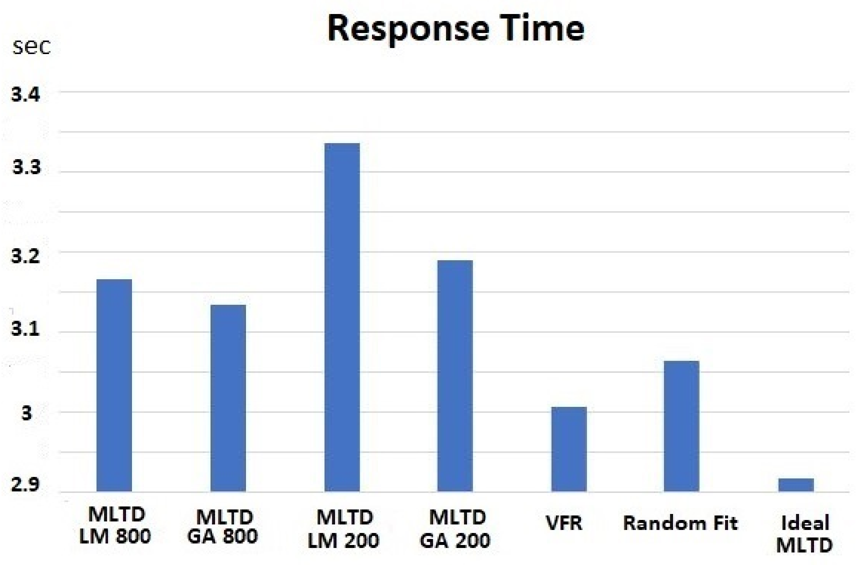


Figure 5.37: Evaluation of different methods in terms of response time in SFSW condition, published in [1]

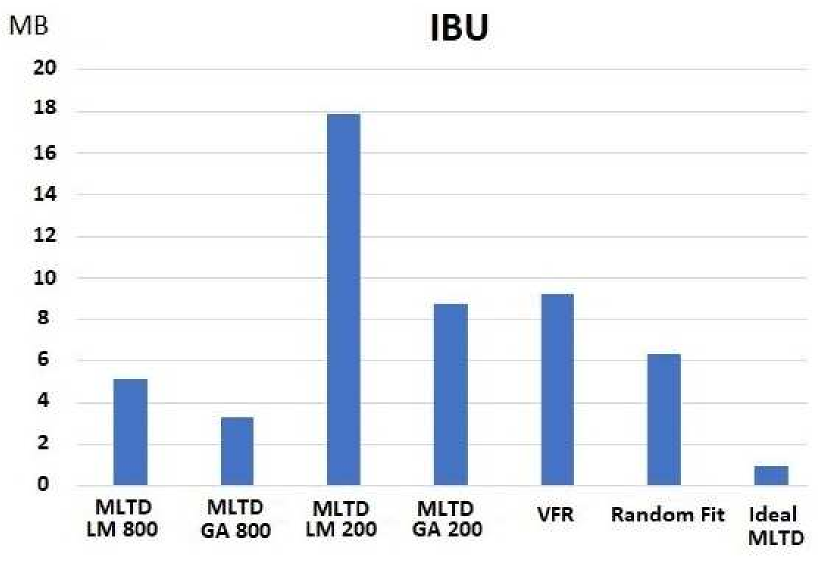


Figure 5.38: Evaluation of different methods in terms of IBU in SFSW condition, published in [1]

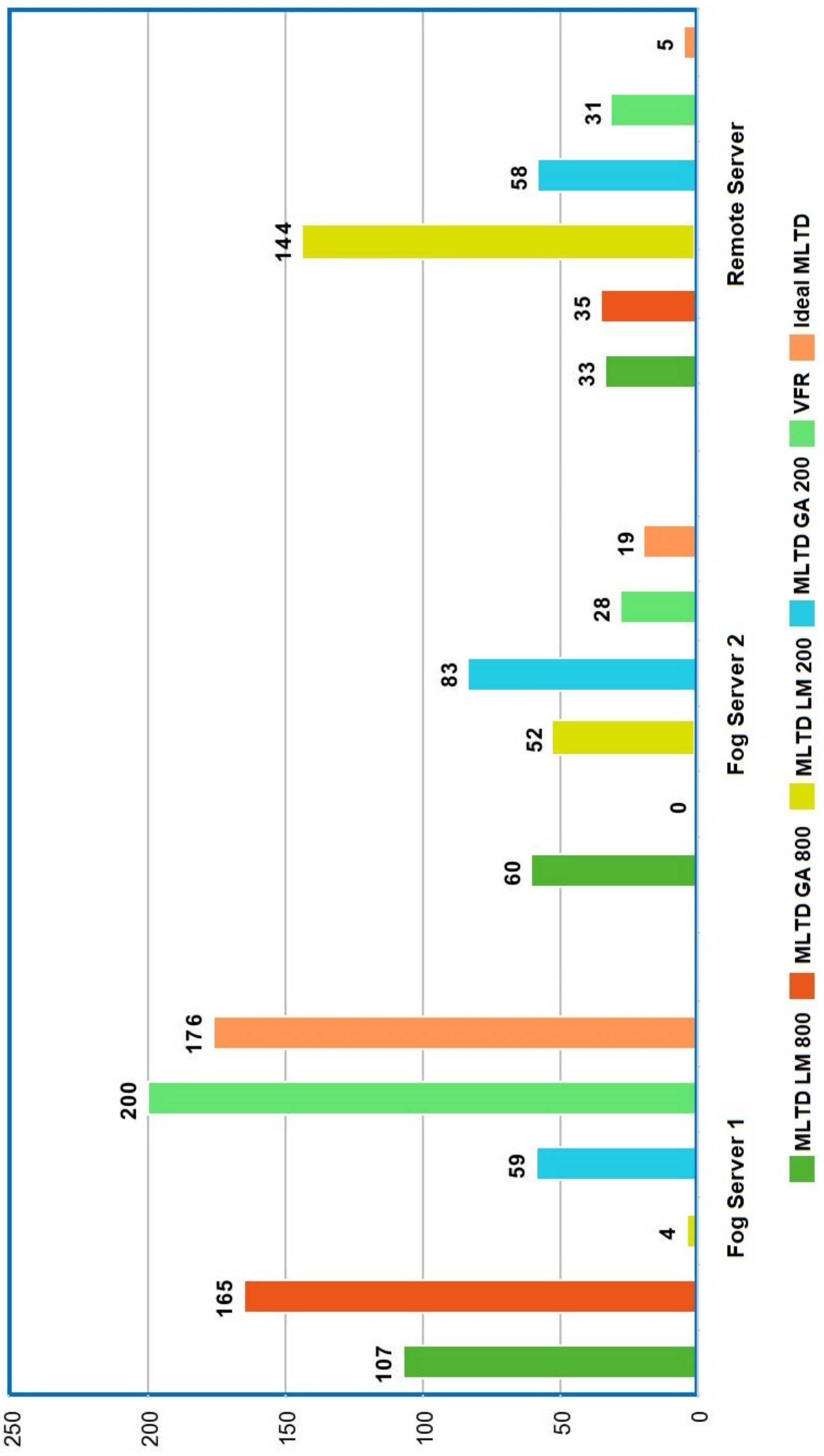


Figure 5.39: The number of processed tasks by each server in DFSW condition, published in [1]

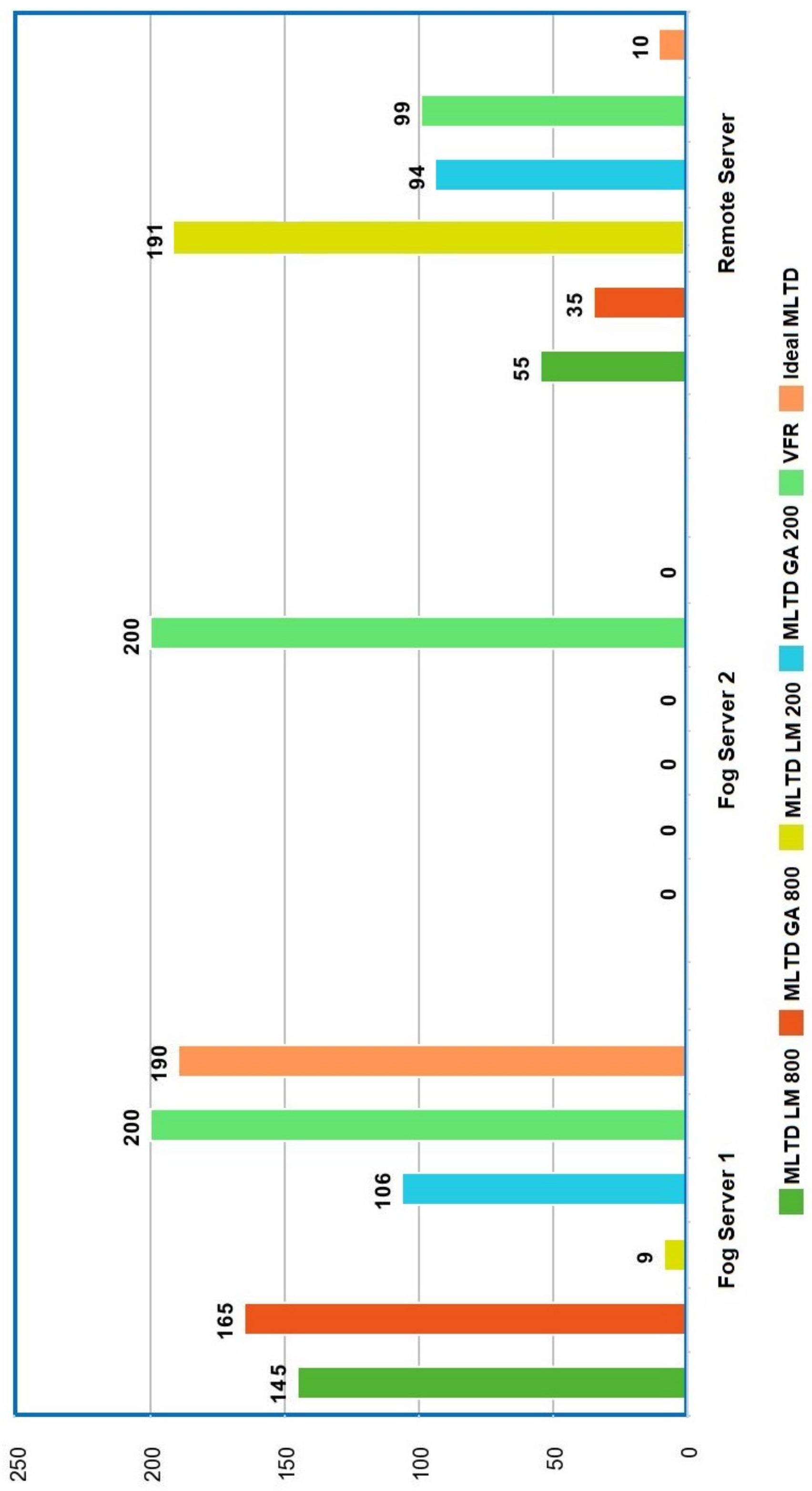


Figure 5.40: The number of processed tasks by each server in SFSW condition, published in [1]

6 Conclusion and Future Works

6.1 Conclusion

In the world of the IoT, various applications utilize sensors for data collection. These sensors generate raw data that needs to be processed (to be available as meaningful information for the users) and stored. However, the IoT sensors do not have enough computational capabilities to process and store their generated data.

Dealing with the above-mentioned problem, different technologies are available that can process the generated raw data by the IoT devices, such as Cloud and Fog computing. However, each of these technologies has disadvantages that make them unsuitable for specific types of applications. For example, cloud servers have considerable computational power, but using them is costly and increases the IBU; therefore, they are not suitable for delay-tolerable applications. Fog computing is also restricted in terms of computational power, and in the case of higher workloads, fog servers fail and provide high response times, which is not tolerable for delay-sensitive applications. Therefore, utilization of only Fog or Cloud servers is not a suitable approach.

In order to exploit both Fog and Cloud servers for processing the generated raw data by the IoT devices, different methods have been proposed in the literature, which we reviewed in chapter 3. We categorized the proposed methods into different groups, such as benchmarks, complex methods, optimization-based approaches, etc. Most of these methods aim to improve the QoS (by reducing the response time) and energy consumption. However, the proposed methods are suitable for specific types of network scenarios, and we discussed that their performance could be affected by different parameters that have not been considered. Therefore, by considering the shortcomings of the previous works, we proposed an artificial intelligence based approach for task distribution in combined Fog-cloud scenarios, which can be utilized in any network architecture. This method considers the application requirements and aims to reduce the response time, IBU, and resource utilization.

Our proposed method utilizes the ANNs for predicting the response times of servers and the size of the results. Then, by considering the predicted amounts, it distributes the tasks between the available servers. To train the neural networks, we ran different numbers of tasks on the servers and then used the parameters of the tasks as the input and the processing times of the servers as the output. In order to investigate the performance of the first version of our proposed method (AITDA), we used it for distributing the SSP tasks of an online delay-sensitive healthcare applications and assumed that the available fog and cloud servers are always idle and ready to process the IoT data. We also assumed that any number of tasks could be available in the broker. The achieved results showed us that AITDA performs better than the Fog-Based and Cloud-Based methods in terms of response time. However, in terms of IBU, the Fog-Based method performed better than AITDA. So, with regard to the first experiment, we concluded that:

- the performance of cloud and fog servers (in terms of response time) depends on the number of assigned tasks (workload) to them, so that when the workload increases, Cloud performs better than Fog.
- Fog performs better than Cloud for one by one task processing. The reason is that the delay of data communication with Cloud is higher than Fog, so when Cloud was used for data processing, the total amount of processing time and delay became more than the response time of Fog.
- smart task distribution helps and reduces the response time, which is an important requirement of delay-sensitive applications.
- AITDA performs better than the Cloud-based method in terms of IBU, and the impact of smart task distribution on IBU becomes more visible when the number of available tasks in the broker increases.

Although AITDA reduced the response time, but during the distribution process, it considers the IBU by sending the smaller tasks (in terms of size) to the cloud layer. So it can be said that AITDA is a suitable approach for delay-tolerable applications, as its main aim is to reduce the IBU. In the next step, we developed the AITDA to be usable for both delay-sensitive and delay-tolerable applications. The new versions of AITDA were called FCSTD time-based (which its main aim is reducing the response time) and FCSTD traffic-based (which its main objective is to reduce the IBU). We used these two methods for distributing the SSP, DSSP, DSP, and SSDP task. We observed that our method always provides a reduced response time in comparison to the other methods. However, we also observed that our approach failed in competition with the Random Fit method for reducing the IBU. In our experiments, Random Fit provided a better IBU compared to our method when we used it to distribute SSP, SSDP, and DSP tasks. But, it must be mentioned that Random Fit reduces the IBU in exchange for a considerably higher response time. So we can conclude that our method is suitable for delay-sensitive applications as it always provides the best response time, but for delay-tolerable applications, the task distribution method must be selected with regard to the types of the tasks of an application.

We used two different applications (in terms of delay sensitivity) for our experiments. The first one was a delay-sensitive online healthcare application in which the variation of inputs affects the output. In this situation, the neural networks can find a relation between the input (parameters of the task) and output (the response time or size of results), and therefore, they can perform excellently. In order to show the impact of different types of applications on the performance of ANNs for predicting the response time, we used the ANNs for distributing the tasks of a delay-tolerable application in which the parameters of tasks did not have any relation with the response times of servers, and we witnessed that ANNs failed. So we can conclude that our proposed method is suitable for distributing the tasks of those applications in which the variation of inputs of the tasks affects the processing times of servers.

In the third step, we assumed that the available servers in Fog and Cloud layers have different workloads that change over time. We developed the FCSTD method to distribute the tasks in such a network scenario by training the neural networks to predict the processing times of servers with different workloads. We compared the performance of our proposed method with four other task distribution techniques, and the results indicated that our method provides the best response time and IBU for task distribution between the servers with different workloads. However, it is worth mentioning that we also investigated the performance of our method for task distribution between the occupied fog servers and remote servers with different workloads. As in this situation, the servers provide similar response times (with a difference of less than the error of ANNs for prediction of response time); our proposed method failed and got ranked as the second-worst approach. So we can conclude that our method was suitable for task distribution in network scenarios where the difference of response times of servers is more than the error of ANNs.

After witnessing the positive impact of our proposed method on reducing the response time and IBU, we finally tried to investigate the impact of different training methods and different richness of training on the performance of ANNs for the selection of the fastest server at each time-slice. Our achieved results showed that the Genetic algorithm trains the ANNs better than Levenberg-Marquardt, and utilizing more tasks for the training of ANNs, increases the accuracy of ANNs for predicting the response times of servers. In addition, in our final experiment, we assumed that the ANNs could predict without error, and we observed that in this case, our proposed method is the best method in comparison to any other task distribution technique.

All in all, the utilization of ANNs in the broker for predicting the response times of fog and cloud servers reduces the response time and IBU only if the ANNs have high prediction accuracy. If this accuracy gets negatively affected by the utilization of unsuitable training algorithms or a few numbers of tasks for the training, our proposed method predicts the response time with a high error, which leads to wrong task distributions and increases the response times and IBU.

6.2 Future Works

6.2.1 Investigating the Performance of our Proposed Method in Scenarios with Mobile Servers

In all of our discussed experiments in this thesis, we assumed that the servers are fixed, and there are always specific numbers of servers available in Fog and Cloud layers. But, in real-world scenarios, the number of servers in each layer (specifically in the Fog layer) is variable. As we discussed in chapter 2, the fog servers can be mobile devices that can enter or leave the Fog network, which increases the number of ANNs in the broker and applies overload to the broker as it must train one neural network for each server and workload. This is a challenge that negatively affects the performance of the broker and leads to increased response times. Therefore, for future works, mobile fog servers must be considered in the experiments, and solutions need to be proposed to deal with this challenge.

6.2.2 Investigating the Impact of Number of Available Servers on the Performance of Broker

In our experiments, the number of servers and their workloads were limited. Therefore, we utilized only one broker to distribute the tasks. If the number of servers increases, the broker needs to train too many neural networks, which causes delay in the broker decision-making process. For future works, we suggest investigating the performance of brokers in network scenarios with higher numbers of fog servers to find out the number of required brokers for different sizes of networks.

6.2.3 Considering the Variation of Delay

We considered the communication delay of fog and cloud servers to be constant. However, in real-world scenarios, the delay in communication changes continuously because of different reasons such as congestion, collision, etc. The variation of delay significantly impacts the response time, and if the broker does not consider the variable delay, then it might assign the produced task to a server with high latency, which leads to an increased response time that is not tolerable by the delay-sensitive applications. Therefore, for future works, the variation of delay must be considered.

6.2.4 Increasing the Accuracy of Function Approximation Method

We used the ANNs as the function approximation method in our experiments. As we discussed earlier, the performance of our proposed method depends on the accuracy of the neural networks, which can be affected by the utilization of different training methods. For future works, we suggest using different function approximation methods (such as support vector machine) and training methods to find the most suitable ones for each type of application.

6.2.5 Considering Different Numbers of Available Tasks in the Broker

In sections 5.3 and 5.4, we investigated the performance of our proposed method for one by one task distribution between the fog and cloud servers. But, as we discussed in section 5.2, the number of available tasks in the broker might increase. Therefore, more than one task can be assigned to a server for processing. Therefore, as the ANNs can only predict the response time of one task, our method needs to be developed to become useful for the assignment of different numbers of tasks to a server. This issue must also be considered for the future works.

A Publications

Chapters 4 and 5 of this dissertation are based on the publications that are listed in the following. The rank of conferences has been written with regard to the CORE [110] and Qualis [111] lists of valid conferences.

1. **M. Pourkiani** and M. Abedi, ”Using Machine Learning for Task Distribution in Fog-Cloud Scenarios: A Deep Performance Analysis,” 35th International Conference on Information Networking (ICOIN), 2021, pp. 445-450, doi: 10.1109/ICOIN50884.2021.9333929.

   Location: Jeju Island, South Korea
   Indexing: IEEE, DBLP, Web of Science
   Sponsor(s): IEEE Computer Society
   Rank: B1

2. **M. Pourkiani** and M. Abedi, ”Machine Learning Based Task Distribution in Heterogeneous Fog-Cloud Environments,” 28th International Conference on Software, Telecommunications and Computer Networks (SoftCOM), 2020, pp. 1-6, doi: 10.23919/SoftCOM50211.2020.9238309.

   Location: Hvar, Croatia
   Indexing: IEEE, DBLP
   Sponsor(s): IEEE Comunication Society
   Rank: B1

3. **M. Pourkiani** and M. Abedi, ”FCSTD: Fog-Cloud Smart Task Distribution by Exploiting the Artificial Neural Networks,” 2020 11th International Conference on Network of the Future (NoF), 2020, pp. 38-42, doi: 10.1109/NoF50125.2020.9249167

   Location: Bordeaux, France
   Indexing: IEEE, DBLP
   Sponsor(s): IEEE Comunication Society
   Rank: B2

4. M. Abedi and **M. Pourkiani**, ”Resource Allocation in Combined Fog-Cloud Scenarios by Using Artificial Intelligence,” 2020 Fifth International Conference on Fog and Mobile Edge Computing (FMEC), 2020, pp. 218-222, doi: 10.1109/FMEC49853.2020.9144693.

   Location: Paris, France
   Indexing: IEEE, DBLP
   Sponsor(s): IEEE France Section
   Rank: —

5. **M. Pourkiani**, M. Abedi and M. A. Tahavori, ”Improving the Quality of Service in WBSN Based Healthcare Applications by Using Fog Computing,” 2019 International Conference on Information and Communications Technology (ICOIACT), 2019, pp. 266-270, doi: 10.1109/ICOIACT46704.2019.8938448.

   Location: Yogyakarta, Indonesia
   Indexing: IEEE
   Sponsor(s): IEEE Indonesia Section
   Rank: —

   Moreover, we also published two other papers that we did not cite in this thesis. These papers are listed in the following:

6. M. Abedi and **M. Pourkiani**, ”AIMCS: An Artificial Intelligence based Method for Compression of Short Strings,” 2020 IEEE 18th World Symposium on Applied Machine Intelligence and Informatics (SAMI), 2020, pp. 311-318, doi: 10.1109/SAMI48414.2020.9108719.

   Location: Herlany, Slovakia
   Indexing: IEEE, Web of Science
   Sponsor(s): IEEE SMC Society
   Rank: National

7. **M. Pourkiani** and M. Abedi, ”An Introduction to a Dynamic Data Size Reduction Approach in Fog Servers,” 2019 International Conference on Information and Communications Technology (ICOIACT), 2019, pp. 261-265, doi: 10.1109/ICOIACT46704.2019.8938494.

   Location: Yogyakarta, Indonesia
   Indexing: IEEE
   Sponsor(s): IEEE Indonesia Section
   Rank: —

In addition, chapters 2 and 3 of this thesis are part of the following paper:

8. **M. Pourkiani**, "Improving the Quality of Service in Combined Fog-Cloud Networks: A Survey," International Journal of Pervasive Computing and Communications (to be submitted).

   Indexing: DBLP, Web of Science

B Source Codes

B.1 Neural Networks

```
%% Start of Program
 clc
 clear
 close all
%tic



%% Data Loading
% This section is utilized for loading the data from the data sets
Data = xlsread('dataset/fogserver2/70Train.csv');
%Data=Data(1:200,:);
X=Data(:,1:end-6);
Y=Data(:,end-1:end);

Data1 = xlsread('dataset/fogserver2/70Test.csv');

X1=Data1(:,1:end-6);
Y1=Data1(:,end-1:end);


DataNum = size(X,1);
InputNum = size(X,2);
OutputNum = size(Y,2);

%% Normalization
% In this section we normalized the data between 0 and 1.
MinX = min(X);
MaxX = max(X);

MinY = min(Y);
MaxY = max(Y);
```

```
XN = X;
YN = Y;

XN1 = X1;
YN1 = Y1;

for ii = 1:InputNum
XN(:,ii) = Normalize_Fcn1(X(:,ii),MinX(ii),MaxX(ii));
XN1(:,ii) = Normalize_Fcn1(X1(:,ii),MinX(ii),MaxX(ii));
end


for ii = 1:OutputNum
YN(:,ii) = Normalize_Fcn1(Y(:,ii),MinY(ii),MaxY(ii));
YN1(:,ii) = Normalize_Fcn1(Y1(:,ii),MinY(ii),MaxY(ii));
end

%% Test and Train Data
% The data is divided into two sections
 for the training and test processes
 (which is not necessary for this case).
TrPercent = 100;
TrNum = round(DataNum * TrPercent / 100);
TsNum = DataNum - TrNum;

R = randperm(DataNum);
trIndex = R(1 : TrNum);
tsIndex = R(1+TrNum : end);

Xtr = XN(trIndex,:);
Ytr = YN(trIndex,:);

Xts = XN(tsIndex,:);
Yts = YN(tsIndex,:);

%% Network Structure
% This section define the structure of the neural networks.
pr = [-1 1];
PR = repmat(pr,InputNum,1);

Network = newff(PR,[4 OutputNum],
{'tansig' 'tansig' 'tansig' 'tansig' 'purelin'});
Network.divideFcn='divideint';%divideint
Network.divideParam.trainRatio=80/100;
Network.divideParam.testRatio=10/100;
Network.divideParam.valRatio =10/100;
%Network.trainParam.max_fail=5;

%% Training
% In this section the neural networks are trained.
%Networknet.trainParam.showWindow = false;

tic
Network = train(Network,Xtr',Ytr');

toc
%% Assesment
% In this section the performance of the neural networks is evaluated.
%YtrNet = sim(Network,Xtr')';
YN1Net = sim(Network,XN1')';
YN1NetUN=YN1Net;
%YtrNetUN=YtrNet;
%MSEtr = mse(YtrNet - Ytr)
MSEts = mse(YN1Net - YN1)

%YS1_1 = Unnormalize_Fcn(YtrNet(:,1),MinY(1),MaxY(1));
%YS1_2 = Unnormalize_Fcn(YtrNet(:,2),MinY(2),MaxY(2));

YSY1_1 = Unnormalize_Fcn(YN1Net(:,1),MinY(1),MaxY(1));
YSY1_2 = Unnormalize_Fcn(YN1Net(:,2),MinY(2),MaxY(2));
```

B.2 Genetic Algorithm

```
function [Network2] = TrainUsing_GA_Fcn(Network,Xtr,Ytr)


%% Problem Statement
IW = Network.IW{1,1}; IW_Num = numel(IW);
LW = Network.LW{2,1}; LW_Num = numel(LW);
b1 = Network.b{1,1}; b1_Num = numel(b1);
b2 = Network.b{2,1}; b2_Num = numel(b2);
```

```
TotalNum = IW_Num + LW_Num + b1_Num + b2_Num;

NPar = TotalNum;

VarLow = -1;
VarHigh = 1;
FunName = 'Cost_ANN_EA';

%% Algorithm Parameters
SelectionMode = 2; % 1 for Random, 2 for Tournment
PopSize = 100;
MaxGenerations = 120;

RecomPercent = 15/100;
CrossPercent = 50/100;
MutatPercent = 1 - RecomPercent - CrossPercent;

RecomNum = round(PopSize*RecomPercent);
CrossNum = round(PopSize*CrossPercent);
if mod(CrossNum,2)~=0
CrossNum = CrossNum - 1;
end

MutatNum = PopSize - RecomNum - CrossNum;

%% Initial Population
Pop = rand(PopSize,NPar) * (VarHigh - VarLow) + VarLow;

Cost = feval(FunName,Pop,Xtr,Ytr,Network);
[Cost Inx] = sort(Cost);
Pop = Pop(Inx,:);

%% Main Loop
MinCostMat = [];
MeanCostMat = [];

for Iter = 1:MaxGenerations
%% Recombination
RecomPop = Pop(1:RecomNum,:);

%% CrossOver
%% Parent Selection
SelectedParentsIndex = MySelection_Fcn(Cost,CrossNum,SelectionMode);

%% Cross Over
CrossPop = [];
for ii = 1:2:CrossNum
Par1Inx = SelectedParentsIndex(ii);
Par2Inx = SelectedParentsIndex(ii+1);

Parent1 = Pop(Par1Inx,:);
Parent2 = Pop(Par2Inx,:);


[Off1 , Off2] = MyCrossOver_Fcn(Parent1,Parent2);

CrossPop = [CrossPop ; Off1 ; Off2];
end
%% Mutation
MutatPop = rand(MutatNum,NPar)*(VarHigh - VarLow) + VarLow;

%% New Population
Pop = [RecomPop ; CrossPop ; MutatPop];
Cost = feval(FunName,Pop,Xtr,Ytr,Network);
[Cost Inx] = sort(Cost);
Pop = Pop(Inx,:);

%% Display
MinCostMat = [MinCostMat ; min(Cost)];
%     [Iter MinCostMat(end)]
MeanCostMat = [MeanCostMat ; mean(Cost)];
subplot(2,1,1)
plot(MinCostMat,'r','linewidth',2.5);
xlim([1 MaxGenerations])
%       hold on
%       plot(MeanCostMat,':b','linewidth',2)
%       hold off

subplot(2,1,2)
plot(Pop(:,1),Pop(:,2),'rp')
axis([VarLow VarHigh VarLow VarHigh])
pause(0.05)
```

```
end
%% Final Result Demonstration
BestSolution = Pop(1,:);
BestCost = Cost(1);
Network2 = ConsNet_Fcn(Network,BestSolution);
```

B.3 Virtual Fog Resolver

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;
using System.Windows.Forms;
namespace Failure_prevention
{
class Program
{
public static double FDelay = 1.4;
public static double CDelay = 2.6;
static void Main(string[] args)
{
string[] F1_30size = new string[200];
string[] F1_30time = new string[200];
string[] F1_50size = new string[200];
string[] F1_50time = new string[200];
string[] F1_70size = new string[200];
string[] F1_70time = new string[200];

string[] F2_30size = new string[200];
string[] F2_30time = new string[200];
string[] F2_50size = new string[200];
string[] F2_50time = new string[200];
string[] F2_70size = new string[200];
string[] F2_70time = new string[200];


string[] C1_30size = new string[200];
string[] C1_30time = new string[200];
string[] C1_50size = new string[200];
```

```
string[] C1_50time = new string[200];
string[] C1_70size = new string[200];
string[] C1_70time = new string[200];

string[] F1_30sizeR = new string[200];
string[] F1_30timeR = new string[200];
string[] F1_50sizeR = new string[200];
string[] F1_50timeR = new string[200];
string[] F1_70sizeR = new string[200];
string[] F1_70timeR = new string[200];

string[] F2_30sizeR = new string[200];
string[] F2_30timeR = new string[200];
string[] F2_50sizeR = new string[200];
string[] F2_50timeR = new string[200];
string[] F2_70sizeR = new string[200];
string[] F2_70timeR = new string[200];


string[] C1_30sizeR = new string[200];
string[] C1_30timeR = new string[200];
string[] C1_50sizeR = new string[200];
string[] C1_50timeR = new string[200];
string[] C1_70sizeR = new string[200];
string[] C1_70timeR = new string[200];

Read_DATA_All(Application.StartupPath + "\\In\\F1\\30.txt",
 F1_30sizeR, F1_30timeR, F1_30size, F1_30time, 1, 1);
Read_DATA_All(Application.StartupPath + "\\In\\F1\\30.txt",
 F1_50sizeR, F1_50timeR, F1_50size, F1_50time, 1, 1);
Read_DATA_All(Application.StartupPath + "\\In\\F1\\30.txt",
 F1_70sizeR, F1_70timeR, F1_70size, F1_70time, 1, 1);

Read_DATA_All(Application.StartupPath + "\\In\\F2\\30.txt",
 F2_30sizeR, F2_30timeR, F2_30size, F2_30time, 1, 1);
Read_DATA_All(Application.StartupPath + "\\In\\F2\\30.txt",
 F2_50sizeR, F2_50timeR, F2_50size, F2_50time, 1, 1);
Read_DATA_All(Application.StartupPath + "\\In\\F2\\30.txt",
 F2_70sizeR, F2_70timeR, F2_70size, F2_70time, 1, 1);

Read_DATA_All(Application.StartupPath + "\\In\\C\\30.txt",
 C1_30sizeR, C1_30timeR, C1_30size, C1_30time, 1, 1);
```

```
Read_DATA_All(Application.StartupPath + "\\In\\C\\50.txt",
 C1_50sizeR, C1_50timeR, C1_50size, C1_50time, 1, 1);
Read_DATA_All(Application.StartupPath + "\\In\\C\\70.txt",
 C1_70sizeR, C1_70timeR, C1_70size, C1_70time, 1, 1);

//Rand part
string []randNumberF1 = new string[200];
string[] randNumberF2 = new string[200];
string[] randNumberC1 = new string[200];
/* for (int i = 0; i < 200; i++)
{
randNumberF1[i] = RandomGen(4, 8, 12).ToString();
randNumberF2[i] = RandomGen(4, 8, 12).ToString();
randNumberC3[i] = RandomGen(4, 8, 12).ToString();
}
for (int i = 0; i < randNumberF1.Length; i++)
{
Save(randNumberF1[i],"\\RandomUseCPU_F1.txt", 1);
Save(randNumberF2[i],"\\RandomUseCPU_F2.txt", 1);
Save(randNumberC3[i],"\\RandomUseCPU_C1.txt", 1);
}*/
Read_DATA_All(Application.StartupPath + "\\RandomUseCPU_F1N13.txt",
 randNumberF1, null, null, null, 2, 1);
Read_DATA_All(Application.StartupPath + "\\RandomUseCPU_F2N13.txt",
 randNumberF2, null, null, null, 2, 1);
Read_DATA_All(Application.StartupPath + "\\RandomUseCPU_C1N13.txt",
 randNumberC1, null, null, null, 2, 1);

double FR1 = 0, FR2 = 0, CR1 = 0;
double SizeC = 0;
double Delay = 0;
int PROBES = 0;
double ORDER = 4;
double DECIY=0.3;
double h_resp = 0, p_resp = 0, c_resp = 0;
double probe_prob = 0.5;
int homeCount=0;
double responseTime=0;
double []ArrayResponseTime=new double[200];
int indexArrayResponseTime=0;
int poolCount=0;
int cloudCount = 0;
```

```
double T_min=0,T_max=0;
double cap_level=0.8;
double inc=0.1;
bool probe_cloud=false;
Random RandNO = new Random();
double[] timeRun = new double[200];
int NOF2 = 0,NOC=0;
double LastDelay=0;
for (int i = 0; i < randNumberF1.LongLength; i++)
{
switch (randNumberF1[i])
{
case "1":
FR1 = Convert.ToDouble(F1_30timeR[i]);
//F1 = Convert.ToDouble(F1_30time[i]);
break;
case "2":
FR1 = Convert.ToDouble(F1_50timeR[i]);
// F1 = Convert.ToDouble(F1_50time[i]);

break;
case "3":
FR1 = Convert.ToDouble(F1_70timeR[i]);
// F1 = Convert.ToDouble(F1_70time[i]);

break;
}
switch (randNumberF2[i])
{
case "1":
FR2 = Convert.ToDouble(F2_30timeR[i]);
//F2 = Convert.ToDouble(F2_30time[i]);

break;
case "2":
FR2 = Convert.ToDouble(F2_50timeR[i]);
//F2 = Convert.ToDouble(F2_50time[i]);

break;
case "3":
FR2 = Convert.ToDouble(F2_70timeR[i]);
//F2 = Convert.ToDouble(F2_70time[i]);
break;
}
switch (randNumberC1[i])
{
case "1":
CR1 = Convert.ToDouble(C1_30timeR[i]);
// C1 = Convert.ToDouble(C1_30time[i]);
break;
case "2":
CR1 = Convert.ToDouble(C1_50timeR[i]);
// C1 = Convert.ToDouble(C1_50time[i]);
break;
case "3":
CR1 = Convert.ToDouble(C1_70timeR[i]);
// C1 = Convert.ToDouble(C1_70time[i]);


break;
}
// F1 += FDelay;
// F2 += FDelay;
// C1 += CDelay;
FR1 += FDelay;
FR2 += FDelay;
CR1 += CDelay;
////////////////////////////////////////////////////////////////////////////////
if(indexArrayResponseTime>0)
{
double Max = ArrayResponseTime[0];
double mean = 0;
for (int k = 0; k < indexArrayResponseTime; k++)
{
if (ArrayResponseTime[k] < Max)
Max = ArrayResponseTime[k];
mean += ArrayResponseTime[k];
}
T_min = mean / indexArrayResponseTime;
T_max = Max;
}
h_resp = FR1;
if (h_resp > Delay)
```

```
{

p_resp = FR2;
NOF2++;
PROBES++;
LastDelay=Delay;
Delay = Math.Abs(p_resp - h_resp) * (Math.Pow(PROBES, ORDER));

}
else
{
Delay *= DECIY;
}
if (probe_cloud)
{
c_resp = CR1;
SizeC += Convert.ToDouble(C1_30size[i]);
NOC++;
if (h_resp < c_resp || p_resp < c_resp)
{
probe_prob *= DECIY;
if (RandNO.NextDouble() <= (1 - probe_prob))
probe_cloud = false;
if (h_resp < p_resp)
{
homeCount++;
responseTime = h_resp;
ArrayResponseTime[indexArrayResponseTime++] = responseTime;
}
else
{
poolCount++;
responseTime = p_resp + LastDelay;
ArrayResponseTime[indexArrayResponseTime++] = responseTime;

}
}
else
{
cloudCount++;
responseTime = c_resp + LastDelay;
//SizeC +=Convert.ToDouble( C1_30size[i]);
```

```
ArrayResponseTime[indexArrayResponseTime++] = responseTime;

}
}
else
{
if (h_resp < T_min || p_resp < T_min)
{
if (probe_prob < cap_level)
probe_prob += inc;
else
probe_prob = cap_level;
// if (h_resp < p_resp)
// {
homeCount++;
responseTime = h_resp;
ArrayResponseTime[indexArrayResponseTime++] = responseTime;
//  }
// else
// {
//     responseTime = p_resp;
//      ArrayResponseTime[indexArrayResponseTime++] = responseTime;
// }

}
else
if (h_resp > T_min && p_resp > T_min)
{
if (RandNO.NextDouble() <= (probe_prob))
probe_cloud = true;
c_resp = CR1;
NOC++;
cloudCount++;
responseTime = c_resp + LastDelay;
SizeC += Convert.ToDouble(C1_30size[i]);
ArrayResponseTime[indexArrayResponseTime++] = responseTime;
}
else
{
if (h_resp > T_max && p_resp > T_max)
{
probe_cloud = true;
```

```
cloudCount++;
responseTime = c_resp + LastDelay;
c_resp = CR1;
SizeC += Convert.ToDouble(C1_30size[i]);
ArrayResponseTime[indexArrayResponseTime++] = responseTime;
}
}
}


//////////////////////////////////////////////////////////////////////

}
}
public static void Read_DATA_All(String PathF, string[] Par1, string[] 
{

string path = PathF;
if (!File.Exists(path))
{

using (FileStream fs = File.Create(path))
{
Byte[] info =
new UTF8Encoding(true).GetBytes("error in Write/Read NO:531");


fs.Write(info, 0, info.Length);
}
}
int index = 0,indexLine=0;

string _temp = "";

using (StreamReader sr = File.OpenText(path))
{
string s = "";
while ((s = sr.ReadLine()) != null)
{
if (mode == 2)
{
Par1[index++] = s.ToString();
```

```
continue;
}

for (int k = 0; k < s.Length; k++)
if (s[k] != '\t')
_temp += s[k];
else
{

switch (indexLine)
{
case 0:
Par1[index] = _temp;
break;
case 1:
Par2[index] = _temp;
break;
case 2:
Par3[index] = _temp;
break;
}
_temp="";
indexLine++;
}
Par4[index] = _temp;
_temp = "";
index++;
indexLine = 0;


}
}

}
public static void Save(string Data,string Path, int mood)
{

string path = Application.StartupPath + Path;
/*  if (mood == 5)
{
path = Application.StartupPath + "\\Temp"+Convert.ToString(n)+".bat";
//  File.Delete(path);
// File.Create(path);
//  return;
}*/
/*if (mood == 6)
{
path = Application.StartupPath + "\\Temp" + Convert.ToString(n) + ".bat
mood = 1;
}*/


//  sialog1.ShowDialog();
//  string path = sialog1.FileName;

// This text is added only once to the file.
/* if (!File.Exists(path))
{
// Create a file to write to.
string[] createText = { "" };
File.WriteAllLines(path, createText);
}*/

// This text is always added, making the file longer over time
// if it is not deleted.
//  string appendText = textBox1.Text + Environment.NewLine;
//  if (mood == 0||mood==2)
// File.AppendAllText(path, Data);
if (mood == 2)
{

File.Delete(path);
// File.Create(path);
return;
}
if (1 == mood)
Data += Environment.NewLine;
File.AppendAllText(path, Data);

string[] readText = File.ReadAllLines(path);
foreach (string s in readText)
{
Console.WriteLine(s);
}
```

```
//  Refresh();

// Open the file to read from.

/* string[] readText = File.ReadAllLines(path);

foreach (string s in readText)
{

Console.WriteLine(s);

}*/

//  MessageBox.Show("Chenging Password.", "Admin", MessageBoxButtons.OK);
}
public static Random RandNo = new Random();
public static int RandomGen(int N1,int N2,int N3)
{


int N=RandNo.Next(1,12);
if (N<N1)
return 1;
else
if(N<N2)
return 2;
else
return 3;

}
}
}
```

B.4 Better Workload

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;
using System.Windows.Forms;
namespace Failure_prevention
{
class Program
{
public static double FDelay = 1.4;
public static double CDelay = 2.6;
static void Main(string[] args)
{
string[] Save1 = new string[210];
for (int kk = 1; kk < 14; kk++)
{
string[] F1_30size = new string[200];
string[] F1_30time = new string[200];
string[] F1_50size = new string[200];
string[] F1_50time = new string[200];
string[] F1_70size = new string[200];
string[] F1_70time = new string[200];

string[] F2_30size = new string[200];
string[] F2_30time = new string[200];
string[] F2_50size = new string[200];
string[] F2_50time = new string[200];
string[] F2_70size = new string[200];
string[] F2_70time = new string[200];


string[] C1_30size = new string[200];
string[] C1_30time = new string[200];
string[] C1_50size = new string[200];
string[] C1_50time = new string[200];
string[] C1_70size = new string[200];
string[] C1_70time = new string[200];

string[] F1_30sizeR = new string[200];
string[] F1_30timeR = new string[200];
string[] F1_50sizeR = new string[200];
string[] F1_50timeR = new string[200];
string[] F1_70sizeR = new string[200];
string[] F1_70timeR = new string[200];

string[] F2_30sizeR = new string[200];
```

```
string[] F2_30timeR = new string[200];
string[] F2_50sizeR = new string[200];
string[] F2_50timeR = new string[200];
string[] F2_70sizeR = new string[200];
string[] F2_70timeR = new string[200];


string[] C1_30sizeR = new string[200];
string[] C1_30timeR = new string[200];
string[] C1_50sizeR = new string[200];
string[] C1_50timeR = new string[200];
string[] C1_70sizeR = new string[200];
string[] C1_70timeR = new string[200];

Read_DATA_All(Application.StartupPath + "\\In\\F1\\30.txt",
 F1_30sizeR, F1_30timeR, F1_30size, F1_30time, 1, 1);
Read_DATA_All(Application.StartupPath + "\\In\\F1\\50.txt",
 F1_50sizeR, F1_50timeR, F1_50size, F1_50time, 1, 1);
Read_DATA_All(Application.StartupPath + "\\In\\F1\\70.txt",
 F1_70sizeR, F1_70timeR, F1_70size, F1_70time, 1, 1);

Read_DATA_All(Application.StartupPath + "\\In\\F1\\30.txt",
 F2_30sizeR, F2_30timeR, F2_30size, F2_30time, 1, 1);
Read_DATA_All(Application.StartupPath + "\\In\\F1\\50.txt",
 F2_50sizeR, F2_50timeR, F2_50size, F2_50time, 1, 1);
Read_DATA_All(Application.StartupPath + "\\In\\F1\\70.txt",
 F2_70sizeR, F2_70timeR, F2_70size, F2_70time, 1, 1);

Read_DATA_All(Application.StartupPath + "\\In\\C\\30.txt",
 C1_30sizeR, C1_30timeR, C1_30size, C1_30time, 1, 1);
Read_DATA_All(Application.StartupPath + "\\In\\C\\50.txt",
 C1_50sizeR, C1_50timeR, C1_50size, C1_50time, 1, 1);
Read_DATA_All(Application.StartupPath + "\\In\\C\\70.txt",
 C1_70sizeR, C1_70timeR, C1_70size, C1_70time, 1, 1);

//Rand part select
string[] randNumberF1 = new string[200];
string[] randNumberF2 = new string[200];
string[] randNumberC1 = new string[200];
/*for (int N = 3; N < 4; N++)
{
for (int i = 0; i < 200; i++)
{
randNumberF1[i] = RandomGen(4, 8, 12).ToString();
randNumberF2[i] = RandomGen(4, 8, 12).ToString();
randNumberC1[i] = RandomGen(4, 8, 12).ToString();
}
for (int i = 0; i < 200; i = i + N)
{
for (int k = i; k < 200 && k < i + N; k++)
{
randNumberF1[k] = randNumberF1[i];
randNumberF2[k] = randNumberF2[i];
randNumberC1[k] = randNumberC1[i];
}
}

for (int i = 0; i < randNumberF1.Length; i++)
{
Save(randNumberF1[i], "\\RandomUseCPU_F1N" + N.ToString() + ".txt", 1);
Save(randNumberF2[i], "\\RandomUseCPU_F2N" + N.ToString() + ".txt", 1);
Save(randNumberC1[i], "\\RandomUseCPU_C1N" + N.ToString() + ".txt", 1);
}
}*/
/////////////////////////////////////////
string[] AllC1 = new string[200];
// double SumC1 = 0;
string[] RandomF1F2 = new string[200];
Random RandVal = new Random();
string[] RandomF1F2C1 = new string[200];
// double SumC1_In_F1F2C1 = 0;
////////////////////////////////////////
Read_DATA_All(Application.StartupPath + "\\RandomUseCPU_F1N" +
 kk.ToString() + ".txt",randNumberF1, null, null, null, 2, 1);
Read_DATA_All(Application.StartupPath + "\\RandomUseCPU_F2N" +
 kk.ToString() + ".txt", randNumberF2, null, null, null, 2, 1);
Read_DATA_All(Application.StartupPath + "\\RandomUseCPU_C1N" +
 kk.ToString() + ".txt", randNumberC1, null, null, null, 2, 1);

double FR1 = 0, FR2 = 0, CR1 = 0;
int F1Count = 0, F2Count = 0, C1Count = 0;
double SizeC = 0;
double[] timeRun = new double[200];
for (int i = 0; i < randNumberF1.LongLength; i++)
```

```
{
switch (randNumberF1[i])
{
case "1":
FR1 = Convert.ToDouble(F1_30timeR[i]);
// F1 = Convert.ToDouble(F1_30time[i]);
break;
case "2":
FR1 = Convert.ToDouble(F1_50timeR[i]);
//   F1 = Convert.ToDouble(F1_50time[i]);

break;
case "3":
FR1 = Convert.ToDouble(F1_70timeR[i]);
//   F1 = Convert.ToDouble(F1_70time[i]);

break;
}
switch (randNumberF2[i])
{
case "1":
FR2 = Convert.ToDouble(F2_30timeR[i]);
// F2 = Convert.ToDouble(F2_30time[i]);

break;
case "2":
FR2 = Convert.ToDouble(F2_50timeR[i]);
//    F2 = Convert.ToDouble(F2_50time[i]);

break;
case "3":
FR2 = Convert.ToDouble(F2_70timeR[i]);
//   F2 = Convert.ToDouble(F2_70time[i]);

break;
}
switch (randNumberC1[i])
{
case "1":
CR1 = Convert.ToDouble(C1_30timeR[i]);
//   C1 = Convert.ToDouble(C1_30time[i]);
break;
```

```
case "2":
CR1 = Convert.ToDouble(C1_50timeR[i]);
//    C1 = Convert.ToDouble(C1_50time[i]);
break;
case "3":
CR1 = Convert.ToDouble(C1_70timeR[i]);
//    C1 = Convert.ToDouble(C1_70time[i]);


break;
}
//      F1 += FDelay;
//    F2 += FDelay;
//    C1 += CDelay;
FR1 += FDelay;
FR2 += FDelay;
CR1 += CDelay;
//////////////////////////////////////////////////////////////////
if ((Convert.ToInt32(randNumberF1[i]) <= Convert.ToInt32
(randNumberF2[i])) &&
(Convert.ToInt32(randNumberF1[i]) <= Convert.ToInt32(randNumberC1[i])))
{
timeRun[i] = FR1;
F1Count++;
}
else
if (Convert.ToInt32(randNumberF2[i]) <= Convert.ToInt32(randNumberC1[i]
{
timeRun[i] = FR2;
F2Count++;
}
else
{
timeRun[i] = CR1;
C1Count++;
SizeC += Convert.ToDouble(C1_30sizeR[i]);
}
}
int ll = 0;
Save1[ll++] += kk.ToString() + "     " + "     ";//+ "     " + "
" + "     ";
Save1[ll++] += "SizeC" + "     " + SizeC.ToString() + "          ";//+
"SumC1_In_F1F2C1    " + SumC1_In_F1F2C1.ToString() + "    ";
Save1[ll++] += "F1Count    " + F1Count.ToString() + "    ";// + "
" + "    " + "    ";
Save1[ll++] += "F2Count    " + F2Count.ToString() + "    ";//+ "
" + "    " + "    ";
Save1[ll++] += "C1Count    " + C1Count.ToString() + "    ";// + "
" + "    " + "    ";
Save1[ll++] += "    " + "    ";//+ "AllC1" + "    " + "RandomF1F2" + "
" + "RandomF1F2C1" + "    ";
for (int i = 0; i < randNumberC1.Length; i++)
Save1[ll++] += "    " + timeRun[i] + "    ";//+ AllC1[i] + "
" + RandomF1F2[i] + "    " + RandomF1F2C1[i] + "    ";
}
for (int i = 0; i <= 205; i++)
Save(Save1[i], "\\out\\All.txt", 1);
}
public static void Read_DATA_All(String PathF, string[] Par1,
 string[] Par2, string[] Par3, string[] Par4, int mode, int indexFile)
{

string path = PathF;
if (!File.Exists(path))
{

using (FileStream fs = File.Create(path))
{
Byte[] info =
new UTF8Encoding(true).GetBytes("error in Write/Read NO:531");


fs.Write(info, 0, info.Length);
}
}
int index = 0, indexLine = 0;

string _temp = "";

using (StreamReader sr = File.OpenText(path))
{
string s = "";
while ((s = sr.ReadLine()) != null)
{
```

```
if (mode == 2)
{
Par1[index++] = s.ToString();
continue;
}

for (int k = 0; k < s.Length; k++)
if (s[k] != '\t')
_temp += s[k];
else
{

switch (indexLine)
{
case 0:
Par1[index] = _temp;
break;
case 1:
Par2[index] = _temp;
break;
case 2:
Par3[index] = _temp;
break;
}
_temp = "";
indexLine++;
}
Par4[index] = _temp;
_temp = "";
index++;
indexLine = 0;


}
}

}
public static void Save(string Data, string Path, int mood)
{

string path = Application.StartupPath + Path;
/*  if (mood == 5)
```

```
{
path = Application.StartupPath + "\\Temp"+Convert.ToString(n)+".bat";
//  File.Delete(path);
// File.Create(path);
//  return;
}*/
/*if (mood == 6)
{
path = Application.StartupPath + "\\Temp" + Convert.ToString(n) + ".bat";
mood = 1;
}*/


//  sialog1.ShowDialog();
//  string path = sialog1.FileName;

// This text is added only once to the file.
/* if (!File.Exists(path))
{
// Create a file to write to.
string[] createText = { "" };
File.WriteAllLines(path, createText);
}*/

// This text is always added, making the file longer over time
// if it is not deleted.
//  string appendText = textBox1.Text + Environment.NewLine;
//  if (mood == 0||mood==2)
// File.AppendAllText(path, Data);
if (mood == 2)
{

File.Delete(path);
// File.Create(path);
return;
}
if (1 == mood)
Data += Environment.NewLine;
File.AppendAllText(path, Data);

string[] readText = File.ReadAllLines(path);
foreach (string s in readText)
```

```
{
Console.WriteLine(s);
}
//  Refresh();

// Open the file to read from.

/* string[] readText = File.ReadAllLines(path);

foreach (string s in readText)
{

Console.WriteLine(s);

}*/

//  MessageBox.Show("Chenging Password.",
 "Admin", MessageBoxButtons.OK);
}
public static Random RandNo = new Random();
public static int RandomGen(int N1, int N2, int N3)
{


int N = RandNo.Next(1, 12);
if (N < N1)
return 1;
else
if (N < N2)
return 2;
else
return 3;

}
}
}
```

B.5 Random Fit

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;
using System.Windows.Forms;
namespace Failure_prevention
{
class Program
{
public static double FDelay = 1.4;
public static double CDelay = 2.6;
static void Main(string[] args)
{
string[] Save1 = new string[210];
for (int kk = 1; kk < 14; kk++)
{
string[] F1_30size = new string[200];
string[] F1_30time = new string[200];
string[] F1_50size = new string[200];
string[] F1_50time = new string[200];
string[] F1_70size = new string[200];
string[] F1_70time = new string[200];

string[] F2_30size = new string[200];
string[] F2_30time = new string[200];
string[] F2_50size = new string[200];
string[] F2_50time = new string[200];
string[] F2_70size = new string[200];
string[] F2_70time = new string[200];


string[] C1_30size = new string[200];
string[] C1_30time = new string[200];
string[] C1_50size = new string[200];
string[] C1_50time = new string[200];
string[] C1_70size = new string[200];
string[] C1_70time = new string[200];

string[] F1_30sizeR = new string[200];
string[] F1_30timeR = new string[200];
string[] F1_50sizeR = new string[200];
string[] F1_50timeR = new string[200];
```

```
string[] F1_70sizeR = new string[200];
string[] F1_70timeR = new string[200];

string[] F2_30sizeR = new string[200];
string[] F2_30timeR = new string[200];
string[] F2_50sizeR = new string[200];
string[] F2_50timeR = new string[200];
string[] F2_70sizeR = new string[200];
string[] F2_70timeR = new string[200];


string[] C1_30sizeR = new string[200];
string[] C1_30timeR = new string[200];
string[] C1_50sizeR = new string[200];
string[] C1_50timeR = new string[200];
string[] C1_70sizeR = new string[200];
string[] C1_70timeR = new string[200];

Read_DATA_All(Application.StartupPath + "\\In\\F1\\30.txt",
 F1_30sizeR, F1_30timeR, F1_30size, F1_30time, 1, 1);
Read_DATA_All(Application.StartupPath + "\\In\\F1\\30.txt",
 F1_50sizeR, F1_50timeR, F1_50size, F1_50time, 1, 1);
Read_DATA_All(Application.StartupPath + "\\In\\F1\\30.txt",
 F1_70sizeR, F1_70timeR, F1_70size, F1_70time, 1, 1);

Read_DATA_All(Application.StartupPath + "\\In\\F2\\30.txt",
 F2_30sizeR, F2_30timeR, F2_30size, F2_30time, 1, 1);
Read_DATA_All(Application.StartupPath + "\\In\\F2\\30.txt",
 F2_50sizeR, F2_50timeR, F2_50size, F2_50time, 1, 1);
Read_DATA_All(Application.StartupPath + "\\In\\F2\\30.txt",
 F2_70sizeR, F2_70timeR, F2_70size, F2_70time, 1, 1);

Read_DATA_All(Application.StartupPath + "\\In\\C\\30.txt",
 C1_30sizeR, C1_30timeR, C1_30size, C1_30time, 1, 1);
Read_DATA_All(Application.StartupPath + "\\In\\C\\50.txt",
 C1_50sizeR, C1_50timeR, C1_50size, C1_50time, 1, 1);
Read_DATA_All(Application.StartupPath + "\\In\\C\\70.txt",
 C1_70sizeR, C1_70timeR, C1_70size, C1_70time, 1, 1);

//Rand part select
string[] randNumberF1 = new string[200];
string[] randNumberF2 = new string[200];
```

```
string[] randNumberC1 = new string[200];
/*for (int N = 3; N < 4; N++)
{
for (int i = 0; i < 200; i++)
{
randNumberF1[i] = RandomGen(4, 8, 12).ToString();
randNumberF2[i] = RandomGen(4, 8, 12).ToString();
randNumberC1[i] = RandomGen(4, 8, 12).ToString();
}
for (int i = 0; i < 200; i = i + N)
{
for (int k = i; k < 200 && k < i + N; k++)
{
randNumberF1[k] = randNumberF1[i];
randNumberF2[k] = randNumberF2[i];
randNumberC1[k] = randNumberC1[i];
}
}

for (int i = 0; i < randNumberF1.Length; i++)
{
Save(randNumberF1[i], "\\RandomUseCPU_F1N" + N.ToString() + ".txt", 1);
Save(randNumberF2[i], "\\RandomUseCPU_F2N" + N.ToString() + ".txt", 1);
Save(randNumberC1[i], "\\RandomUseCPU_C1N" + N.ToString() + ".txt", 1);
}
}*/
/////////////////////////////////////////
string[] AllC1 = new string[200];
double SumC1 = 0;
string[] RandomF1F2 = new string[200];
Random RandVal = new Random();
string[] RandomF1F2C1 = new string[200];
double SumC1_In_F1F2C1 = 0;
/////////////////////////////////////////
Read_DATA_All(Application.StartupPath + "\\RandomUseCPU_F1N" +
 kk.ToString() + ".txt", randNumberF1, null, null, null, 2, 1);
Read_DATA_All(Application.StartupPath + "\\RandomUseCPU_F2N" +
 kk.ToString() + ".txt", randNumberF2, null, null, null, 2, 1);
Read_DATA_All(Application.StartupPath + "\\RandomUseCPU_C1N" +
 kk.ToString() + ".txt", randNumberC1, null, null, null, 2, 1);

double F1 = 0, F2 = 0, C1 = 0, FR1 = 0, FR2 = 0, CR1 = 0;
```

```
int F1Count = 0, F2Count = 0, C1Count = 0;
double SizeC = 0;
double[] timeRun = new double[200];
for (int i = 0; i < randNumberF1.LongLength; i++)
{
switch (randNumberF1[i])
{
case "1":
FR1 = Convert.ToDouble(F1_30timeR[i]);
F1 = Convert.ToDouble(F1_30time[i]);
break;
case "2":
FR1 = Convert.ToDouble(F1_50timeR[i]);
F1 = Convert.ToDouble(F1_50time[i]);

break;
case "3":
FR1 = Convert.ToDouble(F1_70timeR[i]);
F1 = Convert.ToDouble(F1_70time[i]);

break;
}
switch (randNumberF2[i])
{
case "1":
FR2 = Convert.ToDouble(F2_30timeR[i]);
F2 = Convert.ToDouble(F2_30time[i]);

break;
case "2":
FR2 = Convert.ToDouble(F2_50timeR[i]);
F2 = Convert.ToDouble(F2_50time[i]);

break;
case "3":
FR2 = Convert.ToDouble(F2_70timeR[i]);
F2 = Convert.ToDouble(F2_70time[i]);

break;
}
switch (randNumberC1[i])
{
```

```
case "1":
CR1 = Convert.ToDouble(C1_30timeR[i]);
C1 = Convert.ToDouble(C1_30time[i]);
break;
case "2":
CR1 = Convert.ToDouble(C1_50timeR[i]);
C1 = Convert.ToDouble(C1_50time[i]);
break;
case "3":
CR1 = Convert.ToDouble(C1_70timeR[i]);
C1 = Convert.ToDouble(C1_70time[i]);


break;
}
F1 += FDelay;
F2 += FDelay;
C1 += CDelay;
FR1 += FDelay;
FR2 += FDelay;
CR1 += CDelay;
////////////////////////////////////////////////////////////////////
AllC1[i] = CR1.ToString();
SumC1 += Convert.ToDouble(C1_30size[i]);
///////////////////////////////////////////////////////////////////
{
int valRand = RandVal.Next(1, 3);
if (valRand == 1)
RandomF1F2[i] = F1.ToString();
else
RandomF1F2[i] = F2.ToString();
}
///////////////////////////////////////////////////////////////////
{
int valRand = RandVal.Next(1, 4);
switch (valRand)
{
case 1:
RandomF1F2C1[i] = F1.ToString();
break;
case 2:
RandomF1F2C1[i] = F2.ToString();
break;
case 3:
RandomF1F2C1[i] = C1.ToString();
SumC1_In_F1F2C1 += Convert.ToDouble(F1_30size[i]);
break;
}

}
/////////////////////////////////////////////////////////////////
if (F1 <= F2 && F1 <= C1)
{
timeRun[i] = FR1;
F1Count++;
}
else
if (F2 <= F1 && F2 <= C1)
{
timeRun[i] = FR2;
F2Count++;
}
else
{
timeRun[i] = CR1;
C1Count++;
SizeC += Convert.ToDouble(C1_30sizeR[i]);
}
}
int ll = 0;
Save1[ll++] += kk.ToString() + "    " + "    " + "    " + "    " + "
";
Save1[ll++] += "SizeC" + "    " + SizeC.ToString() +
 "    " + "SumC1_In_F1F2C1 " + SumC1_In_F1F2C1.ToString() + " ";
Save1[ll++] += "F1Count    " + F1Count.ToString() + "    " + "
" + "    " + "    ";
Save1[ll++] += "F2Count    " + F2Count.ToString() + "    " + "
" +  "    " + "    ";
Save1[ll++] += "C1Count    " + C1Count.ToString() + "    " + "
" + "    " + "    ";
Save1[ll++] += "    " + "    " + "AllC1" + "    " + "RandomF1F2" + "
" + "RandomF1F2C1" + "    ";
for (int i = 0; i < randNumberC1.Length; i++)
Save1[ll++] += "    " + timeRun[i] + "    " + AllC1[i] + "    " +
 RandomF1F2[i] + "    " + RandomF1F2C1[i] + "    ";
}
for (int i = 0; i <= 205; i++)
Save(Save1[i], "\\out\\All.txt", 1);
}
public static void Read_DATA_All(String PathF, string[]
Par1, string[] Par2, string[] Par3, string[] Par4, int mode,
 int indexFile)
{

string path = PathF;
if (!File.Exists(path))
{

using (FileStream fs = File.Create(path))
{
Byte[] info =
new UTF8Encoding(true).GetBytes("error in Write/Read NO:531");


fs.Write(info, 0, info.Length);
}
}
int index = 0, indexLine = 0;

string _temp = "";

using (StreamReader sr = File.OpenText(path))
{
string s = "";
while ((s = sr.ReadLine()) != null)
{
if (mode == 2)
{
Par1[index++] = s.ToString();
continue;
}

for (int k = 0; k < s.Length; k++)
if (s[k] != '\t')
_temp += s[k];
else
{

switch (indexLine)
{
case 0:
Par1[index] = _temp;
break;
case 1:
Par2[index] = _temp;
break;
case 2:
Par3[index] = _temp;
break;
}
_temp = "";
indexLine++;
}
Par4[index] = _temp;
_temp = "";
index++;
indexLine = 0;


}
}

}
public static void Save(string Data, string Path, int mood)
{

string path = Application.StartupPath + Path;
/*   if (mood == 5)
{
path = Application.StartupPath + "\\Temp"+Convert.ToString(n)+".bat";
//   File.Delete(path);
// File.Create(path);
//   return;
}*/
/*if (mood == 6)
{
path = Application.StartupPath + "\\Temp" +
Convert.ToString(n) + ".bat";
```

```
mood = 1;
}*/


//  sialog1.ShowDialog();
//  string path = sialog1.FileName;

// This text is added only once to the file.
/* if (!File.Exists(path))
{
// Create a file to write to.
string[] createText = { "" };
File.WriteAllLines(path, createText);
}*/

// This text is always added, making the file longer over time
// if it is not deleted.
//  string appendText = textBox1.Text + Environment.NewLine;
//  if (mood == 0||mood==2)
// File.AppendAllText(path, Data);
if (mood == 2)
{

File.Delete(path);
// File.Create(path);
return;
}
if (1 == mood)
Data += Environment.NewLine;
File.AppendAllText(path, Data);

string[] readText = File.ReadAllLines(path);
foreach (string s in readText)
{
Console.WriteLine(s);
}
//  Refresh();

// Open the file to read from.

/* string[] readText = File.ReadAllLines(path);
```

```
foreach (string s in readText)
{

Console.WriteLine(s);

}*/

//  MessageBox.Show("Chenging Password.", "Admin",
 MessageBoxButtons.OK);
}
public static Random RandNo = new Random();
public static int RandomGen(int N1, int N2, int N3)
{


int N = RandNo.Next(1, 12);
if (N < N1)
return 1;
else
if (N < N2)
return 2;
else
return 3;

}
}
}
```

B.6 MLTD

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;
using System.Windows.Forms;
namespace Failure_prevention
{
class Program
{
```

```
\\delays are set
public static double FDelay = 1.4;
public static double CDelay = 2.6;
static void Main(string[] args)
{
string[] Save1 = new string[210];
\\TSWC is set
for (int kk = 1; kk < 11; kk++)
{
\\the definitions of functions
string[] F1_30size = new string[200];
string[] F1_30time = new string[200];
string[] F1_50size = new string[200];
string[] F1_50time = new string[200];
string[] F1_70size = new string[200];
string[] F1_70time = new string[200];

string[] F2_30size = new string[200];
string[] F2_30time = new string[200];
string[] F2_50size = new string[200];
string[] F2_50time = new string[200];
string[] F2_70size = new string[200];
string[] F2_70time = new string[200];


string[] C1_30size = new string[200];
string[] C1_30time = new string[200];
string[] C1_50size = new string[200];
string[] C1_50time = new string[200];
string[] C1_70size = new string[200];
string[] C1_70time = new string[200];

string[] F1_30sizeR = new string[200];
string[] F1_30timeR = new string[200];
string[] F1_50sizeR = new string[200];
string[] F1_50timeR = new string[200];
string[] F1_70sizeR = new string[200];
string[] F1_70timeR = new string[200];

string[] F2_30sizeR = new string[200];
string[] F2_30timeR = new string[200];
string[] F2_50sizeR = new string[200];
```

```
string[] F2_50timeR = new string[200];
string[] F2_70sizeR = new string[200];
string[] F2_70timeR = new string[200];


string[] C1_30sizeR = new string[200];
string[] C1_30timeR = new string[200];
string[] C1_50sizeR = new string[200];
string[] C1_50timeR = new string[200];
string[] C1_70sizeR = new string[200];
string[] C1_70timeR = new string[200];

\\real and estimated size and response times are loaded
Read_DATA_All(Application.StartupPath + "\\In\\F1\\30.txt",
 F1_30sizeR, F1_30timeR, F1_30size, F1_30time, 1, 1);
Read_DATA_All(Application.StartupPath + "\\In\\F1\\30.txt",
 F1_50sizeR, F1_50timeR, F1_50size, F1_50time, 1, 1);
Read_DATA_All(Application.StartupPath + "\\In\\F1\\30.txt",
 F1_70sizeR, F1_70timeR, F1_70size, F1_70time, 1, 1);

Read_DATA_All(Application.StartupPath + "\\In\\F2\\30.txt",
 F2_30sizeR, F2_30timeR, F2_30size, F2_30time, 1, 1);
Read_DATA_All(Application.StartupPath + "\\In\\F2\\30.txt",
 F2_50sizeR, F2_50timeR, F2_50size, F2_50time, 1, 1);
Read_DATA_All(Application.StartupPath + "\\In\\F2\\30.txt",
 F2_70sizeR, F2_70timeR, F2_70size, F2_70time, 1, 1);

Read_DATA_All(Application.StartupPath + "\\In\\C\\30.txt",
 C1_30sizeR, C1_30timeR, C1_30size, C1_30time, 1, 1);
Read_DATA_All(Application.StartupPath + "\\In\\C\\50.txt",
 C1_50sizeR, C1_50timeR, C1_50size, C1_50time, 1, 1);
Read_DATA_All(Application.StartupPath + "\\In\\C\\70.txt",
 C1_70sizeR, C1_70timeR, C1_70size, C1_70time, 1, 1);

//The workloads are randomly set
string[] randNumberF1 = new string[200];
string[] randNumberF2 = new string[200];
string[] randNumberC1 = new string[200];

//definitions of variables that will be used later
string[] AllC1 = new string[200];
double SumC1 = 0;
```

```
string[] RandomF1F2 = new string[200];
Random RandVal = new Random();
string[] RandomF1F2C1 = new string[200];
double SumC1_In_F1F2C1 = 0;
//the workloads are loaded
Read_DATA_All(Application.StartupPath + "\\RandomUseCPU_F1N" +
 kk.ToString() + ".txt", randNumberF1, null, null, null, 2, 1);
Read_DATA_All(Application.StartupPath + "\\RandomUseCPU_F2N" +
 kk.ToString() + ".txt", randNumberF2, null, null, null, 2, 1);
Read_DATA_All(Application.StartupPath + "\\RandomUseCPU_C1N" +
 kk.ToString() + ".txt", randNumberC1, null, null, null, 2, 1);

\\the processing times are set with regard to the
 workloads of servers

double F1 = 0, F2 = 0, C1 = 0, FR1 = 0, FR2 = 0, CR1 = 0;
int F1Count = 0, F2Count = 0, C1Count = 0;
double SizeC = 0;
double[] timeRun = new double[200];
for (int i = 0; i < randNumberF1.LongLength; i++)
{
switch (randNumberF1[i])
{
case "1":
FR1 = Convert.ToDouble(F1_30timeR[i]);
F1 = Convert.ToDouble(F1_30time[i]);
break;
case "2":
FR1 = Convert.ToDouble(F1_50timeR[i]);
F1 = Convert.ToDouble(F1_50time[i]);

break;
case "3":
FR1 = Convert.ToDouble(F1_70timeR[i]);
F1 = Convert.ToDouble(F1_70time[i]);

break;
}
switch (randNumberF2[i])
{
case "1":
FR2 = Convert.ToDouble(F2_30timeR[i]);
```

```
F2 = Convert.ToDouble(F2_30time[i]);

break;
case "2":
FR2 = Convert.ToDouble(F2_50timeR[i]);
F2 = Convert.ToDouble(F2_50time[i]);

break;
case "3":
FR2 = Convert.ToDouble(F2_70timeR[i]);
F2 = Convert.ToDouble(F2_70time[i]);

break;
}
switch (randNumberC1[i])
{
case "1":
CR1 = Convert.ToDouble(C1_30timeR[i]);
C1 = Convert.ToDouble(C1_30time[i]);
break;
case "2":
CR1 = Convert.ToDouble(C1_50timeR[i]);
C1 = Convert.ToDouble(C1_50time[i]);
break;
case "3":
CR1 = Convert.ToDouble(C1_70timeR[i]);
C1 = Convert.ToDouble(C1_70time[i]);


break;
}
F1 += FDelay;
F2 += FDelay;
C1 += CDelay;
FR1 += FDelay;
FR2 += FDelay;
CR1 += CDelay;
////////////////////////////////////////////////////////////////////////
AllC1[i] = CR1.ToString();
SumC1 += Convert.ToDouble(C1_30size[i]);
///////////////////////////////////////////////////////////////////////
{
```

```
int valRand = RandVal.Next(1, 3);
if (valRand == 1)
RandomF1F2[i] = F1.ToString();
else
RandomF1F2[i] = F2.ToString();
}
/////////////////////////////////////////////////////////////////
{
int valRand = RandVal.Next(1, 4);
switch (valRand)
{
case 1:
RandomF1F2C1[i] = F1.ToString();
break;
case 2:
RandomF1F2C1[i] = F2.ToString();
break;
case 3:
RandomF1F2C1[i] = C1.ToString();
SumC1_In_F1F2C1 += Convert.ToDouble(F1_30size[i]);
break;
}

}
/////////////////////////////////////////////////////////////////
\\ Task distribution with regard to predicted response times
if (F1 <= F2 && F1 <= C1)
{
timeRun[i] = FR1;
F1Count++;
}
else
if (F2 <= F1 && F2 <= C1)
{
timeRun[i] = FR2;
F2Count++;
}
else
{
timeRun[i] = CR1;
C1Count++;
SizeC += Convert.ToDouble(C1_30sizeR[i]);
```

```
}
}
int ll = 0;
Save1[ll++] += kk.ToString() + "    " + "    " + "    " + "    " + "
";
Save1[ll++] += "SizeC" + "    " + SizeC.ToString() + "        " +
 "SumC1_In_F1F2C1    " + SumC1_In_F1F2C1.ToString() + "    ";
Save1[ll++] += "F1Count    " + F1Count.ToString() + "    " + "
" + "    " + "    ";
Save1[ll++] += "F2Count    " + F2Count.ToString() + "    " + "
" +  "    " + "    ";
Save1[ll++] += "C1Count    " + C1Count.ToString() + "    " + "
" + "    " + "    ";
Save1[ll++] += "    " + "    " + "AllC1" + "    " + "RandomF1F2" + "
" + "RandomF1F2C1" + "    ";
for (int i = 0; i < randNumberC1.Length; i++)
Save1[ll++] += "    " + timeRun[i] + "    " + AllC1[i] + "    " +
 RandomF1F2[i] + "    " + RandomF1F2C1[i] + "    ";
}
for (int i = 0; i <= 205; i++)
Save(Save1[i], "\\out\\All.txt", 1);
}
public static void Read_DATA_All(String PathF, string[] Par1,
 string[] Par2, string[] Par3, string[] Par4, int mode, int indexFile)
{

string path = PathF;
if (!File.Exists(path))
{

using (FileStream fs = File.Create(path))
{
Byte[] info =
new UTF8Encoding(true).GetBytes("error in Write/Read NO:531");


fs.Write(info, 0, info.Length);
}
}
int index = 0, indexLine = 0;

string _temp = "";
```

```
using (StreamReader sr = File.OpenText(path))
{
string s = "";
while ((s = sr.ReadLine()) != null)
{
if (mode == 2)
{
Par1[index++] = s.ToString();
continue;
}

for (int k = 0; k < s.Length; k++)
if (s[k] != '\t')
_temp += s[k];
else
{

switch (indexLine)
{
case 0:
Par1[index] = _temp;
break;
case 1:
Par2[index] = _temp;
break;
case 2:
Par3[index] = _temp;
break;
}
_temp = "";
indexLine++;
}
Par4[index] = _temp;
_temp = "";
index++;
indexLine = 0;


}
}
```

```
}
public static void Save(string Data, string Path, int mood)
{

string path = Application.StartupPath + Path;
/*  if (mood == 5)
{
path = Application.StartupPath + "\\Temp"+Convert.ToString(n)+".bat";
//  File.Delete(path);
// File.Create(path);
//  return;
}*/
/*if (mood == 6)
{
path = Application.StartupPath + "\\Temp" +
Convert.ToString(n) + ".bat";
mood = 1;
}*/


//  sialog1.ShowDialog();
//  string path = sialog1.FileName;

// This text is added only once to the file.
/* if (!File.Exists(path))
{
// Create a file to write to.
string[] createText = { "" };
File.WriteAllLines(path, createText);
}*/

// This text is always added, making the file longer over time
// if it is not deleted.
//  string appendText = textBox1.Text + Environment.NewLine;
//  if (mood == 0||mood==2)
// File.AppendAllText(path, Data);
if (mood == 2)
{

File.Delete(path);
// File.Create(path);
return;
}
if (1 == mood)
Data += Environment.NewLine;
File.AppendAllText(path, Data);

string[] readText = File.ReadAllLines(path);
foreach (string s in readText)
{
Console.WriteLine(s);
}
//  Refresh();

// Open the file to read from.

/* string[] readText = File.ReadAllLines(path);
foreach (string s in readText)
{

Console.WriteLine(s);

}*/

//  MessageBox.Show("Chenging Password.", "Admin",
 MessageBoxButtons.OK);
}
public static Random RandNo = new Random();
public static int RandomGen(int N1, int N2, int N3)
{


int N = RandNo.Next(1, 12);
if (N < N1)
return 1;
else
if (N < N2)
return 2;
else
return 3;

}
}
}
```

Bibliography

[1] Mohammadreza Pourkiani and Masoud Abedi. Using machine learning for task distribution in fog-cloud scenarios: A deep performance analysis. In *35th International Conference on Information Networking (ICOIN)*. IEEE, 2021. URL `https://doi.org/10.1109/ICOIN50884.2021.9333929`.

[2] Pallavi Sethi and Smruti R Sarangi. Internet of things: architectures, protocols, and applications. *Journal of Electrical and Computer Engineering*, 2017, 2017. URL `https://doi.org/10.1155/2017/9324035`.

[3] Kristof Van Laerhoven and Albrecht Schmidt. How to build smart appliances. *IEEE Personal Communications*, 8(4):66–71, 2001. URL `https://doi.org/10.1109/98.944006`.

[4] Bhavneesh Malik and V. R. Singh. A survey of research in wban for biomedical and scientific applications. *Health and Technology*, 3(3):227–235, 2013. doi: https://doi.org/10.1007/s12553-013-0056-5.

[5] Marwa Salayma, Ahmed Al-Dubai, Imed Romdhani, and Youssef Nasser. Wireless body area network (wban) a survey on reliability, fault tolerance, and technologies coexistence. *ACM Computing Surveys (CSUR)*, 50(1):1–38, 2017. URL `https://doi.org/10.1145/3041956`.

[6] Samaneh Movassaghi, Mehran Abolhasan, Justin Lipman, David Smith, and Abbas Jamalipour. Wireless body area networks: A survey. *IEEE Communications surveys & tutorials*, 16(3):1658–1686, 2014. URL `https://doi.org/10.1109/SURV.2013.121313.00064`.

[7] Ali Hassan Sodhro, Faisal K Shaikh, Sandeep Pirbhulal, Mir Muhammad Lodro, and Madad Ali Shah. Medical-qos based telemedicine service selection using analytic hierarchy process. In *Handbook of large-scale distributed computing in smart healthcare*, pages 589–609. Springer, 2017. URL `https://doi.org/10.1007/978-3-319-58280-1_21`.

[8] Ali Hassan Sodhro, Zongwei Luo, Arun Kumar Sangaiah, and Sung Wook Baik. Mobile edge computing based qos optimization in medical healthcare applications. *International Journal of Information Management*, 45:308–318, 2019. URL `https://doi.org/10.1016/j.ijinfomgt.2018.08.004`.

[9] Ali Hassan Sodhro, Abdul Sattar Malokani, Gul Hassan Sodhro, Muhammad Muzammal, and Luo Zongwei. An adaptive qos computation for medical data processing in intelligent healthcare applications. *Neural computing and applications*, 32(3):723–734, 2020. URL `https://doi.org/10.1007/s00521-018-3931-1`.

[10] Healthcare – reliability, security, and longevity. URL `https://kruse.de/healthcare-reliability-security-and-longevity-industrial-flash-memory/`. Accessed: 09.12.2021.

[11] Olamilekan Fadahunsi and Muthucumaru Maheswaran. Locality sensitive request distribution for fog and cloud servers. *Service Oriented Computing and Applications*, 13(2):127–140, 2019. URL `https://doi.org/10.1007/s11761-019-00260-2`.

[12] Jiuyun Xu, Zhuangyuan Hao, Ruru Zhang, and Xiaoting Sun. A method based on the combination of laxity and ant colony system for cloud-fog task scheduling. *IEEE Access*, 7:116218–116226, 2019. URL `https://doi.org/10.1109/ACCESS.2019.2936116`.

[13] Tina Samizadeh Nikoui, Ali Balador, Amir Masoud Rahmani, and Zeinab Bakhshi. Cost-aware task scheduling in fog-cloud environment. In *2020 CSI/CPSSI International Symposium on Real-Time and Embedded Systems and Technologies (RTEST)*, pages 1–8. IEEE, 2020. URL `https://doi.org/10.1109/RTEST49666.2020.9140118`.

[14] Suchintan Mishra, Manmath Narayan Sahoo, Sambit Bakshi, and Joel JPC Rodrigues. Dynamic resource allocation in fog-cloud hybrid systems using multicriteria ahp techniques. *IEEE Internet of Things Journal*, 7(9):8993–9000, 2020. URL `https://doi.org/10.1109/JIOT.2020.3001603`.

[15] Flávia C Delicato, Paulo F Pires, Thais Batista, et al. *Resource management for Internet of Things*. Springer, 2017. ISBN 978-3-319-54247-8.

[16] Mohammadreza Pourkiani and Masoud Abedi. Machine learning based task distribution in heterogeneous fog-cloud environments. In *28 th International Conference on Software, Telecommunications and Computer Networks (SoftCOM 2020)*. IEEE, 2020. URL `https://doi.org/10.23919/SoftCOM50211.2020.9238309`.

[17] Alessio Botta, Walter De Donato, Valerio Persico, and Antonio Pescapé. Integration of cloud computing and internet of things: a survey. *Future generation computer systems*, 56:684–700, 2016. URL `https://doi.org/10.1016/j.future.2015.09.021`.

[18] Shaik Masthan Babu, A Jaya Lakshmi, and B Thirumala Rao. A study on cloud based internet of things: Cloudiot. In *2015 global conference on communication technologies (GCCT)*, pages 60–65. IEEE, 2015. URL `https://doi.org/10.1109/GCCT.2015.7342624`.

[19] Paolo Bellavista, Javier Berrocal, Antonio Corradi, Sajal K Das, Luca Foschini, and Alessandro Zanni. A survey on fog computing for the internet of things. *Pervasive and mobile computing*, 52:71–99, 2019. URL `https://doi.org/10.1016/j.pmcj.2018.12.007`.

[20] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pages 13–16, 2012. URL `https://doi.org/10.1145/2342509.2342513`.

[21] Foteini Andriopoulou, Tasos Dagiuklas, and Theofanis Orphanoudakis. Integrating iot and fog computing for healthcare service delivery. In *Components and services for IoT platforms*, pages 213–232. Springer, 2017. URL `https://doi.org/10.1007/978-3-319-42304-3_11`.

[22] Tariq Ahamed Ahanger, Usman Tariq, Muneer Nusir, Abdulaziz Aldaej, Imdad Ullah, and Abdullah Sulman. A novel iot–fog–cloud-based healthcare system for monitoring and predicting covid-19 outspread. *The Journal of Supercomputing*, pages 1–24, 2021.

[23] Redowan Mahmud, Fernando Luiz Koch, and Rajkumar Buyya. Cloud-fog interoperability in iot-enabled healthcare solutions. In *Proceedings of the 19th international conference on distributed computing and networking*, pages 1–10, 2018.

[24] Mohammadreza Pourkiani and Masoud Abedi. An introduction to a dynamic data size reduction approach in fog servers. In *2019 International Conference on Information and Communications Technology (ICOIACT)*, pages 261–265. IEEE, 2019. URL `https://doi.org/10.1109/ICOIACT46704.2019.8938494`.

[25] Ranesh Kumar Naha, Saurabh Garg, Dimitrios Georgakopoulos, Prem Prakash Jayaraman, Longxiang Gao, Yong Xiang, and Rajiv Ranjan. Fog computing: Survey of trends, architectures, requirements, and research directions. *IEEE access*, 6:47980–48009, 2018. URL `https://doi.org/10.1109/ACCESS.2018.2866491`.

[26] Doan Hoang and Thanh Dat Dang. Fbrc: Optimization of task scheduling in fog-based region and cloud. In *2017 IEEE Trustcom/BigDataSE/ICESS*,

pages 1109–1114. IEEE, 2017. URL https://doi.org/10.1109/Trustcom/BigDataSE/ICESS.2017.360.

[27] Mohammadreza Pourkiani, Masoud Abedi, and Mohammad Amin Tahavori. Improving the quality of service in wbsn based healthcare applications by using fog computing. In *2019 International Conference on Information and Communications Technology (ICOIACT)*, pages 266–270. IEEE, 2019. URL https://doi.org/10.1109/ICOIACT46704.2019.8938448.

[28] Mohammadreza Pourkiani and Masoud Abedi. Fcstd: Fog-cloud smart task distribution by exploiting the artificial neural networks. In *11th IEEE International Conference on Networks of the Future (NoF 2020)*. IEEE, Accepted, 2020. URL https://doi.org/10.1109/NoF50125.2020.9249167.

[29] Blesson Varghese, Nan Wang, Dimitrios S Nikolopoulos, and Rajkumar Buyya. Feasibility of fog computing. In *Handbook of Integration of Cloud Computing, Cyber Physical Systems and Internet of Things*, pages 127–146. Springer, 2020.

[30] Masoud Abedi and Mohammadreza Pourkiani. Resource allocation in combined fog-cloud scenarios by using artificial intelligence. In *2020 Fifth International Conference on Fog and Mobile Edge Computing (FMEC)*, pages 218–222. IEEE, 2020. URL https://doi.org/10.1109/FMEC49853.2020.9144693.

[31] Karim Foughali, Karim Fathallah, and Ali Frihida. Using cloud iot for disease prevention in precision agriculture. *Procedia computer science*, 130:575–582, 2018. URL https://doi.org/10.1016/j.procs.2018.04.106.

[32] Krittin Intharawijitr, Katsuyoshi Iida, and Hiroyuki 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)*, pages 1–4. IEEE, 2016. URL https://doi.org/10.1109/PERCOMW.2016.7457059.

[33] Vitor Barbosa Carlos de Souza. Mechanisms for service-oriented resource allocation in iot. Universitat Politècnica de Catalunya, 2018. URL http://hdl.handle.net/2117/113988.

[34] Michael Mitzenmacher. On the analysis of randomized load balancing schemes. *Theory of Computing Systems*, 32(3):361–386, 1999. URL https://doi.org/10.1007/s002240000122.

[35] Jason Brownlee. Neural networks are function approximation algorithms, 2020. URL https://machinelearningmastery.com/neural-networks-are-function-approximators/. Accessed: 09.12.2021.

[36] Alem Čolaković and Mesud Hadžialić. Internet of things (iot): A review of enabling technologies, challenges, and open research issues. *Computer Networks*, 144:17–39, 2018. URL https://doi.org/10.1016/j.comnet.2018.07.017.

[37] Rob Van Kranenburg. *The Internet of Things: A critique of ambient technology and the all-seeing network of RFID.* Institute of Network Cultures, 2008. ISBN 978-90-78146-06-3.

[38] Cosmas Zavazava. Itu work on internet of things. In *Presentation at ICTP Workshop*, 2015. URL http://wireless.ictp.it/school_2015/presentations/secondweek/ITU-WORK-ON-IOT.pdf.

[39] Ovidiu Vermesan, Peter Friess, Patrick Guillemin, Sergio Gusmeroli, Harald Sundmaeker, Alessandro Bassi, Ignacio Soler Jubert, Margaretha Mazura, Mark Harrison, Markus Eisenhauer, et al. Internet of things strategic research roadmap. *Internet of things-global technological and societal trends*, 1(2011): 9–52, 2011. URL http://hdl.handle.net/11250/2430372.

[40] Ibrahim Mashal, Osama Alsaryrah, Tein-Yaw Chung, Cheng-Zen Yang, Wen-Hsing Kuo, and Dharma P Agrawal. Choices for interaction with things on internet and underlying issues. *Ad Hoc Networks*, 28:68–90, 2015. URL https://doi.org/10.1016/j.adhoc.2014.12.006.

[41] Omar Said and Mehedi Masud. Towards internet of things: Survey and future vision. *International Journal of Computer Networks*, 5(1):1–17, 2013. URL https://www.cscjournals.org/library/manuscriptinfo.php?mc=IJCN-265.

[42] Miao Wu, Ting-Jie Lu, Fei-Yang Ling, Jing Sun, and Hui-Ying Du. Research on the architecture of internet of things. In *2010 3rd International Conference on Advanced Computer Theory and Engineering (ICACTE)*, volume 5, pages V5–484. IEEE, 2010. URL https://doi.org/10.1109/ICACTE.2010.5579493.

[43] Sandra Sendra Compte. Deployment of efficient wireless sensor nodes for monitoring in rural, indoor and underwater environments. PhD Thesis. 2013. URL https://doi.org/10.4995/Thesis/10251/32279.

[44] Sourav Banerjee, Chinmay Chakraborty, and Sumit Chatterjee. A survey on iot based traffic control and prediction mechanism. In *Internet of Things and Big Data Analytics for Smart Generation*, pages 53–75. Springer, 2019. URL https://doi.org/10.1007/978-3-030-04203-5_4.

[45] L Minh Dang, Md Piran, Dongil Han, Kyungbok Min, Hyeonjoon Moon, et al. A survey on internet of things and cloud computing for healthcare. *Electronics*, 8(7):768, 2019. URL `https://doi.org/10.3390/electronics8070768`.

[46] Zhibo Pang. Technologies and architectures of the internet-of-things (iot) for health and well-being. PhD Thesis. 2013. ISBN 978-91-7501-736-5.

[47] Redefine connectivity by building a network to support the internet of things. URL `https://www.cisco.com/c/en/us/solutions/service-provider/a-network-to-support-iot.html`. Accessed: 09.12.2021.

[48] Ibrar Yaqoob, Ejaz Ahmed, Ibrahim Abaker Targio Hashem, Abdelmuttlib Ibrahim Abdalla Ahmed, Abdullah Gani, Muhammad Imran, and Mohsen Guizani. Internet of things architecture: Recent advances, taxonomy, requirements, and open challenges. *IEEE wireless communications*, 24(3):10–16, 2017. URL `https://doi.org/10.1109/MWC.2017.1600421`.

[49] Rafiullah Khan, Sarmad Ullah Khan, Rifaqat Zaheer, and Shahid Khan. Future internet: the internet of things architecture, possible applications and key challenges. In *2012 10th international conference on frontiers of information technology*, pages 257–260. IEEE, 2012. URL `https://doi.org/10.1109/FIT.2012.53`.

[50] Muneer Bani Yassein, Shadi Aljawarneh, and Walaa Al-Sarayrah. Mobility management of internet of things: Protocols, challenges and open issues. In *2017 International Conference on Engineering & MIS (ICEMIS)*, pages 1–8. IEEE, 2017. URL `https://doi.org/10.1109/ICEMIS.2017.8273021`.

[51] In Lee and Kyoochun Lee. The internet of things (iot): Applications, investments, and challenges for enterprises. *Business Horizons*, 58(4):431–440, 2015. URL `https://doi.org/10.1016/j.bushor.2015.03.008`.

[52] Internet of things at a glance, 2020. URL `https://emarsonindia.com/wp-content/uploads/2020/02/Internet-of-Things.pdf`.

[53] Oyster Bay. Data captured by iot connections to top 1.6 zettabytes in 2020, as analytics evolve from cloud to edge, April 2015. URL `https://www.abiresearch.com/press/data-captured-by-iot-connections-to-top-16-zettaby/`.

[54] Changyan Yi, Jun Cai, and Zhou Su. A multi-user mobile computation offloading and transmission scheduling mechanism for delay-sensitive applications. *IEEE Transactions on Mobile Computing*, 19(1):29–43, 2019. URL `https://doi.org/10.1109/TMC.2019.2891736`.

[55] Jie Ding, Mahyar Nemati, Chathurika Ranaweera, and Jinho Choi. Iot connectivity technologies and applications: A survey. *arXiv preprint arXiv:2002.12646*, pages 67646–67673, 2020. URL `https://doi.org/10.1109/ACCESS.2020.2985932`.

[56] Ian Foster, Yong Zhao, Ioan Raicu, and Shiyong Lu. Cloud computing and grid computing 360-degree compared. In *2008 grid computing environments workshop*, pages 1–10. IEEE, 2008. URL `https://doi.org/10.1109/GCE.2008.4738445`.

[57] Peter Mell, Tim Grance, et al. The nist definition of cloud computing. 2011. URL `https://csrc.nist.gov/publications/detail/sp/800-145/final`.

[58] Hanan Elazhary. Internet of things (iot), mobile cloud, cloudlet, mobile iot, iot cloud, fog, mobile edge, and edge emerging computing paradigms: Disambiguation and research directions. *Journal of Network and Computer Applications*, 128:105–140, 2019. URL `https://doi.org/10.1016/j.jnca.2018.10.021`.

[59] Jie Lin, Wei Yu, Nan Zhang, Xinyu Yang, Hanlin Zhang, and Wei Zhao. A survey on internet of things: Architecture, enabling technologies, security and privacy, and applications. *IEEE Internet of Things Journal*, 4(5):1125–1142, 2017. URL `https://doi.org/10.1109/JIOT.2017.2683200`.

[60] Carlo Puliafito, Enzo Mingozzi, Francesco Longo, Antonio Puliafito, and Omer Rana. Fog computing for the internet of things: A survey. *ACM Transactions on Internet Technology (TOIT)*, 19(2):1–41, 2019. URL `https://doi.org/10.1145/3301443`.

[61] Pengfei Hu, Sahraoui Dhelim, Huansheng Ning, and Tie Qiu. Survey on fog computing: architecture, key technologies, applications and open issues. *Journal of network and computer applications*, 98:27–42, 2017. URL `https://doi.org/10.1016/j.jnca.2017.09.002`.

[62] Matthew NO Sadiku, Sarhan M Musa, and Omonowo D Momoh. Cloud computing: opportunities and challenges. *IEEE potentials*, 33(1):34–36, 2014. URL `https://doi.org/10.1109/MPOT.2013.2279684`.

[63] Jun Zhou, Zhenfu Cao, Xiaolei Dong, and Athanasios V Vasilakos. Security and privacy for cloud-based iot: Challenges. *IEEE Communications Magazine*, 55(1):26–33, 2017. URL `https://doi.org/10.1109/MCOM.2017.1600363CM`.

[64] Mahadev Satyanarayanan, Grace Lewis, Edwin Morris, Soumya Simanta, Jeff Boleng, and Kiryong Ha. The role of cloudlets in hostile environments. *IEEE*

Pervasive Computing, 12(4):40–49, 2013. URL https://doi.org/10.1109/MPRV.2013.77.

[65] Luis M Vaquero and Luis Rodero-Merino. Finding your way in the fog: Towards a comprehensive definition of fog computing. *ACM SIGCOMM Computer Communication Review*, 44(5):27–32, 2014. URL https://doi.org/10.1145/2677046.2677052.

[66] What is fog computing?, 2014. URL https://www.ibm.com/blogs/cloud-computing/2014/08/25/fog-computing/.

[67] The openfog consortium reference architecture: Executive summary, 2017. URL https://www.iiconsortium.org/pdf/OpenFog-Reference-Architecture-Executive-Summary.pdf.

[68] Soumya Kanti Datta, Christian Bonnet, and Jerome Haerri. Fog computing architecture to enable consumer centric internet of things services. In *2015 International Symposium on Consumer Electronics (ISCE)*, pages 1–2. IEEE, 2015. URL https://doi.org/10.1109/ISCE.2015.7177778.

[69] Shanhe Yi, Zhengrui Qin, and Qun Li. Security and privacy issues of fog computing: A survey. In *International conference on wireless algorithms, systems, and applications*, pages 685–695, 2015. URL https://doi.org/10.1007/978-3-319-21837-3_67.

[70] Mung Chiang and Tao Zhang. Fog and iot: An overview of research opportunities. *IEEE Internet of Things Journal*, 3(6):854–864, 2016. URL https://doi.org/10.1109/JIOT.2016.2584538.

[71] Pengfei Hu, Huansheng Ning, Tie Qiu, Yanfei Zhang, and Xiong Luo. Fog computing based face identification and resolution scheme in internet of things. *IEEE transactions on industrial informatics*, 13(4):1910–1920, 2016. URL https://doi.org/10.1109/TII.2016.2607178.

[72] Prateeksha Varshney and Yogesh Simmhan. Demystifying fog computing: Characterizing architectures, applications and abstractions. In *2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC)*, pages 115–124. IEEE, 2017. URL https://doi.org/10.1109/ICFEC.2017.20.

[73] Tom H Luan, Longxiang Gao, Zhi Li, Yang Xiang, Guiyi Wei, and Limin Sun. Fog computing: Focusing on mobile users at the edge. *arXiv preprint arXiv:1502.01815*, 2015. URL https://arxiv.org/abs/1502.01815.

[74] M Shohrab Hossain and Mohammed Atiquzzaman. Cost analysis of mobility protocols. *Telecommunication Systems*, 52(4):2271–2285, 2013. URL https://doi.org/10.1007/s11235-011-9532-2.

[75] Mohammed A Hassan, Mengbai Xiao, Qi Wei, and Songqing Chen. Help your mobile applications with fog computing. In *2015 12th Annual IEEE International Conference on Sensing, Communication, and Networking-Workshops (SECON Workshops)*, pages 1–6. IEEE, 2015. URL `https://doi.org/10.1109/SECONW.2015.7328146`.

[76] Kang Kai, Wang Cong, and Luo Tao. Fog computing for vehicular ad-hoc networks: paradigms, scenarios, and issues. *the journal of China Universities of Posts and Telecommunications*, 23(2):56–96, 2016. URL `https://doi.org/10.1016/S1005-8885(16)60021-3`.

[77] Flavio Bonomi, Rodolfo Milito, Preethi Natarajan, and Jiang Zhu. Fog computing: A platform for internet of things and analytics. In *Big data and internet of things: A roadmap for smart environments*, pages 169–186. Springer, 2014. URL `https://doi.org/10.1007/978-3-319-05029-4_7`.

[78] Clinton Dsouza, Gail-Joon Ahn, and Marthony Taguinod. Policy-driven security management for fog computing: Preliminary framework and a case study. In *Proceedings of the 2014 IEEE 15th international conference on information reuse and integration (IEEE IRI 2014)*, pages 16–23. IEEE, 2014. URL `https://doi.org/10.1109/IRI.2014.7051866`.

[79] Yang Zhang, Dusit Niyato, Ping Wang, and Dong In Kim. Optimal energy management policy of mobile energy gateway. *IEEE Transactions on Vehicular Technology*, 65(5):3685–3699, 2015. URL `https://doi.org/10.1109/TVT.2015.2445833`.

[80] Subhadeep Sarkar and Sudip Misra. Theoretical modelling of fog computing: a green computing paradigm to support iot applications. *Iet Networks*, 5(2):23–29, 2016. URL `https://doi.org/10.1049/iet-net.2015.0034`.

[81] Fatemeh Jalali, Kerry Hinton, Robert Ayre, Tansu Alpcan, and Rodney S Tucker. Fog computing may help to save energy in cloud computing. *IEEE Journal on Selected Areas in Communications*, 34(5):1728–1739, 2016. URL `https://doi.org/10.1109/JSAC.2016.2545559`.

[82] Xavi Masip-Bruin, Eva Marín-Tordera, Ghazal Tashakor, Admela Jukan, and Guang-Jie Ren. Foggy clouds and cloudy fogs: a real need for coordinated management of fog-to-cloud computing systems. *IEEE Wireless Communications*, 23(5):120–128, 2016. URL `https://doi.org/10.1109/MWC.2016.7721750`.

[83] Ruilong Deng, Rongxing Lu, Chengzhe Lai, Tom H Luan, and Hao Liang. Optimal workload allocation in fog-cloud computing toward balanced delay

and power consumption. *IEEE internet of things journal*, 3(6):1171–1181, 2016. URL https://doi.org/10.1109/JIOT.2016.2565516.

[84] Jianbo Du, Liqiang Zhao, Jie Feng, and Xiaoli Chu. Computation offloading and resource allocation in mixed fog/cloud computing systems with min-max fairness guarantee. *IEEE Transactions on Communications*, 66(4):1594–1608, 2018. URL https://doi.org/10.1109/TCOMM.2017.2787700.

[85] Mohit Taneja and Alan 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)*, pages 1222–1228. IEEE, 2017. URL https://doi.org/10.23919/INM.2017.7987464.

[86] Hamed Shah-Mansouri and Vincent WS Wong. Hierarchical fog-cloud computing for iot systems: A computation offloading game. *IEEE Internet of Things Journal*, 5(4):3246–3257, 2018. URL https://doi.org/10.1109/JIOT.2018.2838022.

[87] Vitor Barbosa C Souza, Wilson Ramírez, Xavier Masip-Bruin, Eva Marín-Tordera, G Ren, and Ghazal Tashakor. Handling service allocation in combined fog-cloud scenarios. In *2016 IEEE international conference on communications (ICC)*, pages 1–5. IEEE, 2016. URL https://doi.org/10.1109/ICC.2016.7511465.

[88] Adam A Alli and Muhammad Mahbub Alam. The fog cloud of things: A survey on concepts, architecture, standards, tools, and applications. *Internet of Things*, 9:100177, 2020. URL https://doi.org/10.1016/j.iot.2020.100177.

[89] Elarbi Badidi and Karima Moumane. Enhancing the processing of healthcare data streams using fog computing. In *2019 IEEE Symposium on Computers and Communications (ISCC)*, pages 1113–1118. IEEE, 2019. URL https://doi.org/10.1109/ISCC47284.2019.8969736.

[90] Xin Li and Cui Tang. Makespan minimization for batch tasks in data centers. In *International Conference on Security, Privacy and Anonymity in Computation, Communication and Storage*, pages 115–123. Springer, 2016. URL https://doi.org/10.1007/978-3-319-49145-5_12.

[91] Ashkan Yousefpour, Genya Ishigaki, and Jason P Jue. Fog computing: Towards minimizing delay in the internet of things. In *2017 IEEE international conference on edge computing (EDGE)*, pages 17–24. IEEE, 2017. URL https://doi.org/10.1109/IEEE.EDGE.2017.12.

[92] Amira Rayane Benamer, Hana Teyeb, and Nejib Ben Hadj-Alouane. Latency-aware placement heuristic in fog computing environment. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, pages 241–257. Springer, 2018. URL `https://doi.org/10.1007/978-3-030-02671-4_14`.

[93] Michael Mitzenmacher. The power of two choices in randomized load balancing. *IEEE Transactions on Parallel and Distributed Systems*, 12(10):1094–1104, 2001. URL `https://doi.org/10.1109/71.963420`.

[94] Huaiying Sun, Huiqun Yu, and Guisheng Fan. Contract-based resource sharing for time effective task scheduling in fog-cloud environment. *IEEE Transactions on Network and Service Management*, 2020. URL `https://doi.org/10.1109/TNSM.2020.2977843`.

[95] Mahdi Abbasi, Ehsan Mohammadi Pasand, and Mohammad R Khosravi. Workload allocation in iot-fog-cloud architecture using a multi-objective genetic algorithm. *Journal of Grid Computing*, pages 1–14, 2020. URL `https://doi.org/10.1007/s10723-020-09507-1`.

[96] Stamatios-Aggelos N Alexandropoulos, Christos K Aridas, Sotiris B Kotsiantis, and Michael N Vrahatis. Multi-objective evolutionary optimization algorithms for machine learning: A recent survey. In *Approximation and Optimization*, pages 35–55. Springer, 2019. URL `https://doi.org/10.1007/978-3-030-12767-1_4`.

[97] Rodrigo AC da Silva and Nelson LS da Fonseca. Resource allocation mechanism for a fog-cloud infrastructure. In *2018 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2018. URL `https://doi.org/10.1109/ICC.2018.8422237`.

[98] Lei Li, Mian Guo, Lihong Ma, Huiyun Mao, and Quansheng Guan. Online workload allocation via fog-fog-cloud cooperation to reduce iot task service delay. *Sensors*, 19(18):3830, 2019. URL `https://doi.org/10.3390/s19183830`.

[99] Vitor Barbosa Souza, Xavi Masip-Bruin, Eva Marín-Tordera, Wilson Ramírez, and Sergio Sanchez. Towards distributed service allocation in fog-to-cloud (f2c) scenarios. In *2016 IEEE global communications conference (GLOBECOM)*, pages 1–6. IEEE, 2016. URL `https://doi.org/10.1109/GLOCOM.2016.7842341`.

[100] Ranesh Kumar Naha, Saurabh Garg, Andrew Chan, and Sudheer Kumar Battula. Deadline-based dynamic resource allocation and provisioning algorithms in fog-cloud environment. *Future Generation Computer Systems*, 104:131–141, 2020. URL `https://doi.org/10.1016/j.future.2019.10.018`.

[101] Mohammed Anis Benblidia, Bouziane Brik, Leila Merghem-Boulahia, and Moez Esseghir. Ranking fog nodes for tasks scheduling in fog-cloud environments: A fuzzy logic approach. In *2019 15th International Wireless Communications & Mobile Computing Conference (IWCMC)*, pages 1451–1457. IEEE, 2019. URL `https://doi.org/10.1109/IWCMC.2019.8766437`.

[102] AP Gagge, JAJ Stolwijk, and B Saltin. Comfort and thermal sensations and associated physiological responses during exercise at various ambient temperatures. *Environmental Research*, 2(3):209–229, 1969. URL `https://doi.org/10.1016/0013-9351(69)90037-1`.

[103] Roger Haslam. *An evaluation of models of human response to hot and cold environments.* PhD thesis, Loughborough University, 1989. URL `https://hdl.handle.net/2134/7027`.

[104] RA Haslam and KC Parsons. Computer-based models of human responses to the thermal environment: Are their predictions accurate enough for practical use? *Thermal Physiology, JB Mercer, Ed*, pages 763–768, 1989.

[105] M Salloum, N Ghaddar, and K Ghali. A new transient bioheat model of the human body and its integration to clothing models. *International journal of thermal sciences*, 46(4):371–384, 2007. URL `https://doi.org/10.1016/j.ijthermalsci.2006.06.017`.

[106] Jinming Zou, Yi Han, and Sung-Sau So. Overview of artificial neural networks. In *Artificial Neural Networks*, pages 14–22. Springer, 2008. URL `https://doi.org/10.1007/978-1-60327-101-1_2`.

[107] Steven Walczak. Neural network models for a resource allocation problem. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 28(2):276–284, 1998. URL `https://doi.org/10.1109/3477.662769`.

[108] Rashid Mijumbi, Juan-Luis Gorricho, Joan Serrat, Maxim Claeys, Jeroen Famaey, and Filip De Turck. Neural network-based autonomous allocation of resources in virtual networks. In *2014 European Conference on Networks and Communications (EuCNC)*, pages 1–6. IEEE, 2014. URL `https://doi.org/10.1109/EuCNC.2014.6882668`.

[109] Ingo Steinwart and Andreas Christmann. *Support vector machines.* Springer Science & Business Media, 2008. ISBN 978-0-387-77242-4.

[110] The core conference ranking portal. URL `https://www.core.edu.au/conference-portal`. Accessed: 09.12.2021.

[111] Qualis conference ranking portal. URL `https://qualis.ic.ufmt.br`. Accessed: 09.12.2021.

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Dissertation mit dem Titel “Smart Task Distribution in Combined Fog-Cloud Scenarios” selbstständig und ohne fremde Hilfe verfasst, andere als die von mir angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Werken wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Rostock, 1. April 2022

Mohammadreza Pourkiani